What is Visual Basic?

The word “Visual” refers to the way the Graphical User Interface (GUI) is designed. With some programming languages you have to design the GUI’s by writing lots of code to describe everything about the GUI (for example appearance, location of controls, how to display the controls, etc). But with Visual Basic all you have to do is select the control you want and draw it on the form. This is also known as object orientated programming. The word basic stands for Beginners All-Purpose Symbolic Instruction Code which refers to the coding section of the software.

Structure of a Visual Basic Application

Application (Project) is made up of:

Forms:

Windows that you create for user interface

Controls:

Graphical features drawn on forms to allow user interaction (text boxes, labels, scroll bars, command buttons, etc.) (Forms and Controls are **objects**.)

Properties:

Every characteristic of a form or control is specified by a property. Example properties include names, captions, size, colour, position, and contents. Visual Basic applies default properties. You can change properties at design time or run time.

Methods:

Built-in procedure that can be invoked to impart some action to a particular object

Event Procedures:

Code related to some object. This is the code that is executed when a certain event occurs.

General Procedures:

Code not related to objects. This code must be invoked by the application.

Modules:

Collection of general procedures, variable declarations, and constant definitions used by application

Steps in Developing Application:

There are three primary steps involved in building a Visual Basic application:

1. **Draw** the user **interface**
2. **Assign properties** to controls
3. **Attach code** to controls

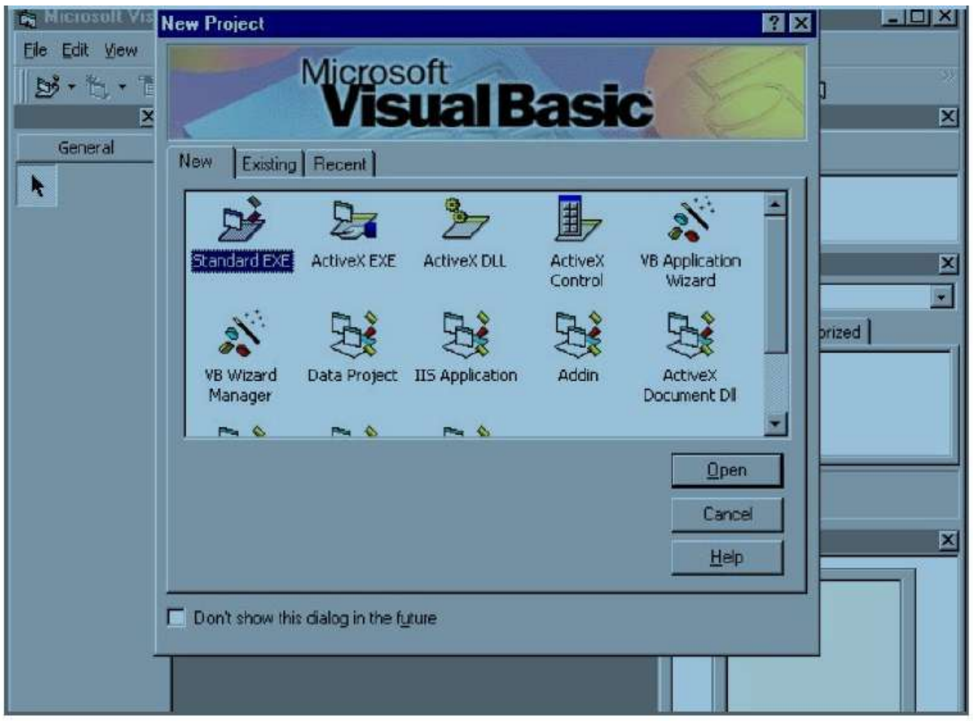
Visual Basic operates in three modes:

1. **Design** mode - used to build application
2. **Run** mode - used to run the application
3. **Break** mode - application halted and debugger is available

The Visual Basic Environment

Upon start up, Visual Basic 6.0 will display the following dialog box as shown in Figure. You can choose to start a new project, open an existing project or select a list of recently opened programs. A project is a collection of files that make up your application. There are various types of applications that can be created; however, we shall concentrate on creating Standard EXE programs (EXE means executable program). Now, click on the Standard EXE icon to go into the VB programming environment.

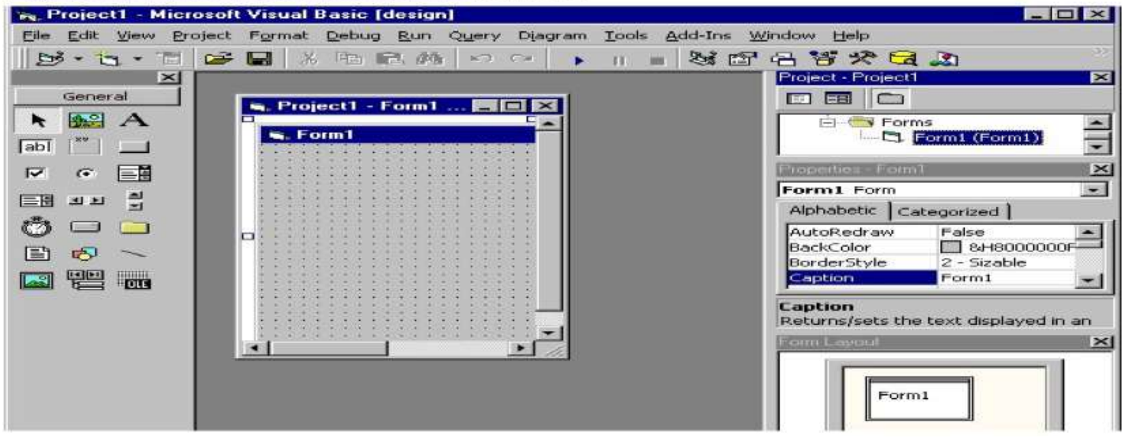
Figure 1.1 The Visual Basic Start-up Dialog Box



In Figure 1.2, the Visual Basic Environment consists of

1. A blank form for you to design your application's interface.
2. The project window, which displays the files that are created in your application.
3. The properties window which displays the properties of various controls and objects that are created in your application.

Figure 1.2: The Visual Basic Environment



The Integrated Development Environment - IDE

One of the most significant changes in Visual Basic 6.0 is the Integrated Development Environment (IDE). IDE is a term commonly used in the programming world to describe the interface and environment that we use to create our applications. It is called integrated because we can access virtually all of the development tools that we need from one screen called an interface. The IDE is also commonly referred to as the design environment, or the program.

Tha Visual Basic IDE is made up of a number of components

1. Menu Bar
2. Tool Bar
3. Project Explorer
4. Properties window
5. Form Layout Window
6. Toolbox
7. Form Designer
8. Object Browser

Menu Bar

This Menu Bar displays the commands that are required to build an application. The main menu items have sub menu items that can be chosen when needed.

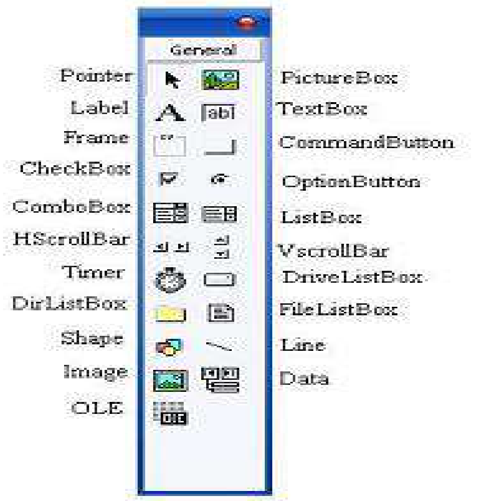
Tool Bar

The toolbars in the menu bar provide quick access to the commonly used commands and a button in the toolbar is clicked once to carry out the action represented by it.

Toolbox

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

Toolbox window with its controls available commonly



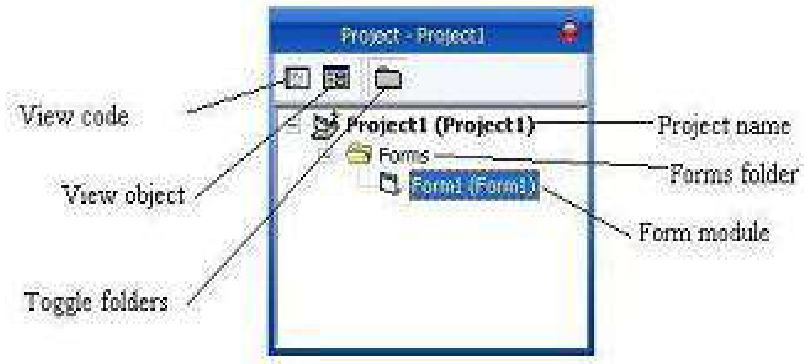
Control	Description
Pointer	Provides a way to move and resize the controls form
Picture Box	Displays icons/bitmaps and metafiles. It displays text or acts as a visual container for other controls.
Text Box	Used to display message and enter text.
Frame	Serves as a visual and functional container for controls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

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

Project Explorer

The Project Explorer as shown in the following figure serves as a quick reference to the various elements of a project namely form, classes and modules. All of the object that make up the application are packed in a project. A simple project will typically contain one form, which is a window that is designed as part of a program's interface. It is possible to develop any number of forms for use in a program, although a program may consist of a single form. In addition to forms, the Project Explorer window also lists code modules and classes.

Project Explorer



Properties Window

The Properties Window exposes the various characteristics of selected objects. Each and every form in an application is considered an object. All the characteristics of an object are called its properties. Thus, a form has properties and any controls placed on it will have properties too. All of these properties are displayed in the Properties Window.

Object Browser

The Object Browser allows us to browse through the various properties, events and methods that are made available to us. It is accessed by selecting Object Browser from the View menu or pressing the key F2.

Working with Forms

The most basic object you will be working with in Visual Basic is the form object, which is the visual foundation for building an application. It is basically a window that you can add different elements to in order to create a complete application.

Setting Properties

Any VB application program uses the Form to gather input from the user and to display the output information to the user.
There are a few Special Properties given in the following sections.

The Border

Depending on the type of form you want to display, you can program the border to be fixed, sizable or even nonexistent. These features can be set with the Border Style property.

The Caption

The form's caption is the text you see in the form's title bar. It can be used to identify the name of the application, the current function of the form, or as a status bar. What you put in the caption depends on what your program is trying to achieve

The Title Bar

The title bar is the colored bar on the top of most forms. If your desktop colour scheme is set to the Windows default scheme, this bar will be blue. You can use the title bar to drag the window around the screen. In addition, double-clicking on it will alternately maximize and restore the form.

Maximize/Restore Button (Boolean)

The Maximize button has two purposes. If the form is in its normal state, that is its normal size, you can click the Maximize button to stretch the current form to the size of the screen or container of the form.

Min Button (Boolean)

The Minimize button is used to minimize the current form, that is, move it out of the way. To enable this button on your form, set the form's Minbutton property to True in the form's Properties window.

The Close Button

The Close button's sole purpose is to close the current window. In Visual Basic, you can control whether the Close button is visible to the user with the Control Box property. The Close button will not be visible if the Control box, is not visible. If you decide not to enable the Close button or the Control box, they you must use a menu or a button to close the form.

Moveable (Boolean)

Determines whether the Form can be moved. That is, if it is set to False, the Form will remain in its initial start-up position. User can't move it around screen. However, it can be resized.

Start up Position

Specifies the position of the Form when it first appears with one the values given in the following table

Value	Description
0-Manual	**No initial is setting specified.**
1-Center Owner	**Center on the item to which the User Form belongs.**
2-Center Screen	**Center on the whole screen.**
3-Windows Default	**Position in upper-left corner of screen.**

Window State

Set the visual state such as Maximized, Minimized or Normal of the Form at run time with one the value given in the following table.

Value	Description
0-Normal	**(Default) Normal.**
1-Minimized	**Minimized (minimized to an icon)**
2-Maximized	**Maximized (enlarged to maximum size)**

A Form can be in one of the following states:

Not loaded – The Form lives on a disk file and doesn't take up any resources.

Loaded but not shown – The Form is loaded into memory, takes up the required resources, and is ready to be displayed.

Loaded and shown – The Form is shown, and the user can interact with it.

The Load and Unload to load and unload the Forms

The Load statement has the following syntax:
Load form Name
And the Unload statement has this syntax:
Unload form Name

The form Name is the name of the Form to be loaded or unloaded. When Visual Basic starts, it loads the default Form, say Form1, and displays it. To load other Forms, say Form2, the load statement is to be issued as in below:

Load Form2

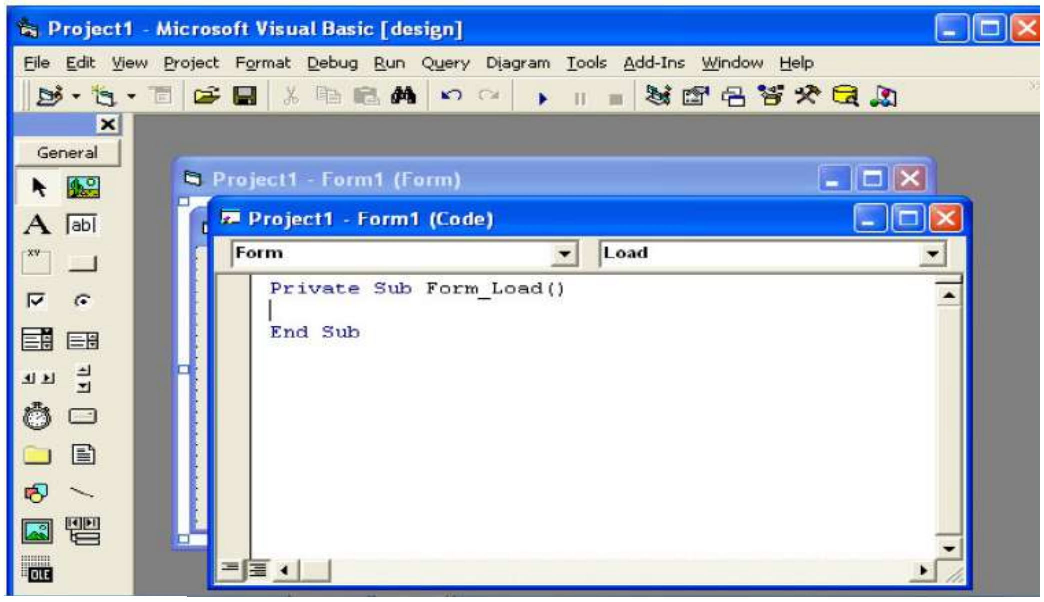
This statement will load the Form2 into the memory. But the Form2 will not be visible immediately. To make it visible its Show method (explained shortly) must be called. Once a Form is loaded, it takes over the required resources, so when the Form is not needed it must be unloaded as in below:

Unload Form2

When a Form is unloaded, the resources it occupies are returned to the system and can be used by other Forms and/or applications.

Creating Simple Visual Basic Applications

First of all, launch Microsoft Visual Basic. Normally, a default form Form1 will be available. To start a new project. Double click on Form1, and the source code window for it as shown in the following Figure. The top of the source code window consists of a list of objects (on the left) and their associated events or procedures (on the right).

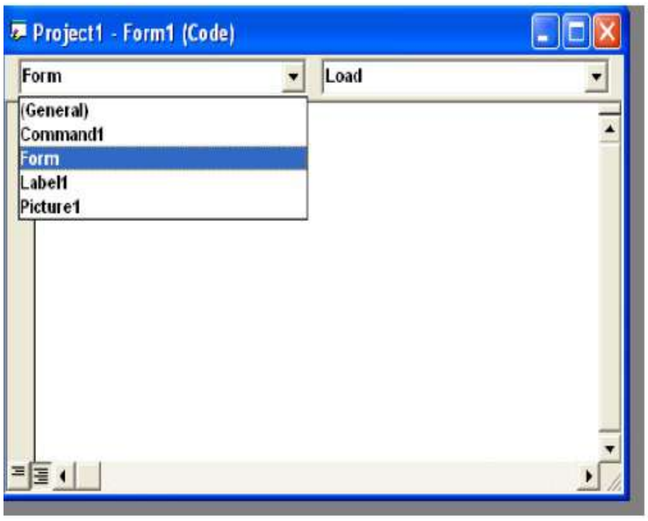

Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
General
Project1 - Form1 (Form)
Project1 - Form1 (Code)
Form
Load

```
Private Sub Form_Load()

End Sub
```

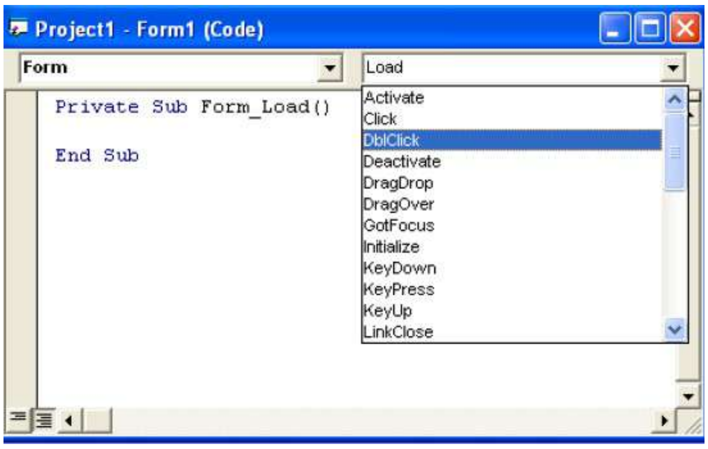
The Source code Wimdow

Click on the object box, the drop-down list will display a list of objects that have been inserted into the form.



List Of Objects

Similarly, click on the procedure box, a list of procedures associated with the object will be displayed.



```
Private Sub Form_Load()

End Sub
```

List of Procedures

In order to display the output of the program, you have to add the Form1.show statement.

Now, press F5 or click on the run button to run the program.

Example1:

```
Private Sub Form_Load ( )
Form1.show
Print "Welcome to Visual Basic tutorial"
End Sub
```

Example 2 :-

```
Private Sub Form_Activate ( )
Print 20 + 10
Print 20 - 10
Print 20 * 10
Print 20 / 10
End Sub
```

Properties, Methods, Events:

All the controls in the Tool Box except the Pointer are objects in Visual Basic. These objects have associated properties, methods and events.

PROPERTIES

Properties define the characteristics of an object such as Size, Colour etc. or sometimes the way in which it behaves. For example, a Text Box accepts properties such as Enabled, Font, Multi Line, Text, Visible, Width, etc. These design-time properties can be set at the design tme by selecting the Properties Window. But certain properties cannot be set at design time. For example, the Current X and Current Y properties of a Form cannot be set at the design time. These are Run time properties

METHODS

A method is an action that can be performed on objects. For Example, the statement

Text1. Move 700, 400 it performs a very precise action to move a textbox.

The Textbox control has other associated methods such as Refresh, Set Focus, etc.

1. The Refresh method enforces a complete repaint of the control or a Form.
   For example, Text1.Refresh-- refreshes the Textbox.
2. The Set focus method moves the focus on the control.
   For Example Text1.SetFocus-- sets the focus to Textbox control Text1.

EVENTS

Visual Basic programs are built around events. Events are various things that can happen in a program. In an event driven application, the program statements are executed only when a particular event calls a specific part of the code that is assigned to the event.

Let us consider a Textbox control and a few of its associated events to understand the concept of event driven programming. The Textbox control supports various events such as Change, Click, Mouse Move and many more that will be listed in the Properties dropdown list in the code window for the Textbox control. We will look into a few of them as given below.

1. The code entered in the Change event fires when there is a change in the contents of the Textbox
2. The Click event fires when the Textbox control is clicked.
3. The Mouse Move event fires when the mouse is moved over the Textbox.

Variables, Data Types and Modules

Visual Basic uses building blocks such as variables, Datatypes, Procedures, Functions and control structures in its programming environment.

Modules

Module is a separate code file in Visual Basic that contains procedures and declarations. The three kind of modules are Form Modules, Standard Modules and Class Modules.

Form Module

The complete code including procedures and declarations part of a form is referred to as the form module.

Standard Modules

Standard modules (.BAS file name extension) are containers for procedures and declarations commonly accessed by other modules within the application.

They can contain global (available to the whole application) or module-level declarations of variables, constants, types, external procedures, and global procedures.

Class Modules

Class modules (.CLS file name extension) are the foundation of object-oriented programming in Visual Basic. New objects can be created in class modulus. These new objects can include own customized properties and methods. Actually, forms are just class modules that can have controls placed on them and can display form windows.

Data types

The data type of a programming element refers to what kind of data it can hold and how it stores that data. Data types apply to all values that can be stored in computer memory or participate in the evaluation of an expression

Table 6.1:Numeric Data Types

Type	Storage	Range of Values
Byte	1 byte	0 to 255
Integer	2 bytes	-32, 768 to 32, 767
Long	4 bytes	-2, 147, 483, 648 to 2, 147, 483, 648
Single	4 bytes	-3.402823E+38 to -1.401298E-45 for negative values 1.401298E-45 to 3.402823E + 38 for positive values.
Double	8 bytes	-1.79769313486232e+308 to -4.94065645841247E-324 for negative values 4.94065645841247E-324 to 1.79769313486232e+308 for positive values.
Currency	8 bytes	-922, 337, 203, 685, 477.5808 to 922, 337, 203, 685, 477.5807
Decimal	12 bytes	+/-79,228, 162, 514, 264, 337, 593, 543, 950, 335 if no decimal is use +/-7.9228162514264337593543950335 (28 decimal places).

Variables

1. Variables are the memory locations which are used to store values temporarily.
2. A variable name must begin with an alphabet letter and should not exceed 255 characters.
3. It must be unique within the same scope. It should not contain any special character like %, &, !, #, @ or $.

There are many ways of declaring variables in Visual Basic.
That is listed below:

1. Explicit Declaration
2. Using Option Explicit statement
3. Scope of Variables

Explicit Declaration

Declaring a variable tells Visual Basic to reserve space in memory.

The variables are declared with a Dim statement to name the variable and its type. The As type clause in the Dim statement allows to define the data type or object type of the variable. This is called explicit declaration.

Syntax

Dim variable [As Type]
For example,
Dim strName As String
Dim intCounter As Integer

Using Option Explicit statement

It may be convenient to declare variables implicitly, but it can lead to errors that may not be recognized at run time. Say, for example a variable by name intcount is used implicitly and is assigned to a value. In the next step, this field is incremented by 1 by the following statement

Intcount = Intcount + 1

This calculation will result in intcount yielding a value of 1 as intcount would have been initialized to zero. This is because the intcount variable has been mityped as incont in the right hand side of the second variable. But Visual Basic does not see this as a mistake and consider it to be new variable and therefore gives a wrong result.

In Visual Basic, to prevent errors of this nature, a variable can be declared by adding the following statement to the general declaration section of the Form.

Option Explicit

This forces the user to declare all the variables. The Option Explicit statement checks in the module for usage of any undeclared variables and reports an error to the user. The user can thus rectify the error on seeing this error message.

The Option Explicit statement can be explicitly placed in the general declaration section of each module using the following steps.

1. Click Options item in the Tools menu
2. Click the Editor tab in the Options dialog box
3. Check Require Variable Declaration option and then click the OK button

Scope of variables

A variable is scoped to a procedure-level (local) or module-level variable depending on how it is declared.

Local Variables

A local variable is one that is declared inside a procedure. This variable is only available to the code inside the procedure and can be declared using the Dim statements as given below.

Dim sum As Integer

The local variables exist as long as the procedure in which they are declared, is executing.

Once a procedure is executed, the values of its local variables are lost and the memory used by these variables is freed and can be reclaimed.

Static Variables

Static variables are not reinitialized each time Visual Invokes a procedure and therefore retain or preserve value even when a procedure ends.

A static variable is declared as given below.

Static impermanent As Integer

The following is an example of an event procedure for a Command Button that counts and displays the number of clicks made.

```
Private Sub Command1_Click ( )
Static Counter As Integer
Counter = Counter + 1
Print Counter
End Sub
```

When the Command Button, is clicked on the first time, the Counter starts with its default value of zero. Visual Basic then adds 1 to it and prints the result.

Module Level Variables

A module level variable is available to all the procedures in the module. They are declared using the Public or the Private keyword.

Variables that are visible only from within the module they belong to and can't be accessed from the outside. In general, these variables are useful for sharing data among procedures in the same module:

```
Private Login Time As Date ' A private module-level variable
Dim Login Password As String ' Another private module-level variable
```

The Public attribute can also be used for module-level variables, for all module types except BAS modules.

Procedures

Visual Basic offers different types of procedures to execute small sections of coding in applications. Visual Basic programs can be broken into smaller logical components called Procedures.

Sub Procedures

A sub procedure can be placed in standard, class and form modules. Each time the procedure is called, the statements between Sub and End Sub are executed. The syntax for a sub procedure is as follows:

```
[Private | Public] [Static] Sub Procedure name [( arglist)]
[ statements]
End Sub
```

Arglist is a list of argument names separated by commas. Each argument acts like a variable in the procedure. There are two types of Sub Procedures namely general procedures and event procedures.

Event Procedures

An event procedure is a procedure block that contains the control's actual name, an underscore(_), and the event name. The following syntax represents the event procedure for a Form_Load event.

```
Private Sub Form_Load()
....statement block..
End Sub
```

Event Procedures acquire the declarations as Private by default.

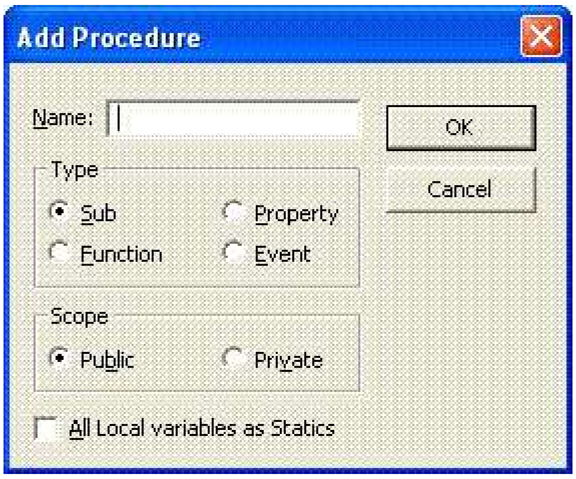
General Procedures

A general procedure is declared when several event procedures perform the same actions. It is a good programming practice to write common statements in a separate procedure (general procedure) and then call them in the event procedure.

In order to add General procedure:

1. The Code window is opened for the module to which the procedure is to be added.

2. The Add Procedure option is chosen from the Tools menu, which opens an Add Procedure dialog box as shown in the figure given below.
3. The name of the procedure is typed in the Name textbox
4. Under Type, Sub is selected to create a Sub procedure, Function to create a Function procedure or Property to create a Property procedure.
5. Under Scope, Public is selected to create a procedure that can be invoked outside the module, or Private to create a procedure that can be invoked only from within the module.


Add Procedure

Name:

Type: [● Sub] Property Function Event

Scope: [● Public] Private

All Local variables as Statics

A new procedure can also be created in the current module by typing Sub Procedure Name, Function Procedure Name, or Property Procedure Name in the Code window. A Function procedure returns a value and a Sub Procedure does not return a value.

Function Procedures
Functions are like sub procedures, except they return a value to the calling procedure.

They are especially useful for taking one or more pieces of data, called arguments and performing some tasks with them. Then the functions return a value that indicates the results of the tasks complete within the function.

The following function procedure calculates the third side or hypotenuse of a right triangle, where A and B are the other two sides. It takes two arguments A and B (of data type Double) and finally returns the results.

```
Function Hypotenuse (A As Double, B As Double) As Double
Hypotenuse = sqr (A^2 + B^2)
End Function
```

The above function procedure is written in the general declarations section of the Code window. A function can also be written by selecting the Add Procedure dialog box from the Tools menu and by choosing the required scope and type.

Property Procedures
A property procedure is used to create and manipulate custom properties.
It is used to create read only properties for Forms, Standard modules and Class modules.
Visual Basic provides three kind of property procedures

1. Property Let procedure that sets the value of a property,
2. Property Get procedure that returns the value of a property and
3. Property Set procedure that sets the references to an object.

Control Structures In Visual Basic 6.0

Programs are not monolithic sets of commands that carry out the same calculations every time they are executed. Instead, they adjust their behavior depending on the data supplied or based on the result of a test condition. Visual Basic provides three control flow, or decision, structures to take a course of action depending on the outcome of the test.

They are:
1. If..Then
2. If...Then...Else
3. Select Case

If...Then...End If

The If structure test the condition specified and, if it it's True, executes the statements(s) that follow. The If structure can have a single-line or multiple-line

Syntax:
If condition Then Statement

In the above statement, Visual Basic evaluates the condition and, if it's True, executes the statement that follows. If the condition is not True, it continues with the statement following the If structure. Multiple statements can also be supplied, provided they must be separated by colon as in the following:

If condition Then Statment1:Statement2:Statement3
Here is an example of single-line If statement:
If Salary > 3000 Then DA Percent = 100

This statement can be broken into Multiple lines, as it will be easier to read as follows:
If Salary > 3000 Then
DA Percent = 100
End If

A variation of the If...Then is the **If...Then...Else** statement, which executes one block of statements if the condition is true and another if the condition is false.

Syntax :-

```
If condition Then
Statement block - 1
Else
Statement block – 2
End If
```

Another variation of the If...Then...Else statement uses several conditions, with the **Else if** keyword:

e.g.: Assume you have to find the grade using nested if and display in a text box

```
If average > 75 Then
txtGrade.Text = "A"
ElseIf average > 65 Then
txtGrade.Text = "B"
ElseIf average > 55 Then
txtGrade.text = "C"
ElseIf average > 45 Then
txtGrade.Text = "S"
Else
txtGrade.Text = "F"
End If
```

Select...Case selection structure

Select...Case structure is an alternative to If...Then...ElseIf for selectively executing a single block of statements from among multiple block of statements. Select...case is more convenient to use than the If...Else...End If. The following program block illustrate the working of Select...Case.

Syntax

```
Select Case Index
Case 0
Statements
Case 1
Statements
End Select
```

e.g.: Assume you have to find the grade using select...case and display in the text box

Dim average as Integer

```
average = txtAverage.Text
Select Case average
Case 100 To 75
txtGrade.Text ="A"
Case 74 To 65
txtGrade.Text ="B"
Case 64 To 55
txtGrade.Text ="C"
Case 54 To 45
txtGrade.Text ="S"
Case 44 To 0
txtGrade.Text ="F"
```

```
Case Else
MsgBox "Invalid average marks"
End Select
```

Do While... Loop Statement:

The **Do While...Loop** is used to execute statements until a certain condition is met. The following Do Loop counts from 1 to 100.

```
Dim number As Integer
number = 1
Do While number <= 100
number = number + 1
```

Loop

A variable number is initialized to 1 and then the Do While Loop starts. First, the condition is tested; if condition is True, then the statements are executed. When it gets to the Loop it goes back to the Do and tests condition again. If condition is False on the first pass, the statements are never executed.

While... Wend Statement

A While...Wend statement behaves like the Do While...Loop statement. The following While...Wend counts from 1 to 100

```
Dim number As Integer
number = 1
While number <=100
number = number + 1
Wend
```

Do...Loop While Statement

The **Do...Loop** While statements first executes the statements and then test the condition after each execution. The following program block illustrates the structure:

```
Dim number As Long
number = 0
Do
number = number + 1
Loop While number < 201
```

The programs execute the statements between Do and Loop While structure in any case. Then it determines whether the counter is less than 501. If so, the program again executes the statements between Do and Loop While else exits the Loop.

Do Until...Loop Statement

Unlike the **Do While...Loop** and **While...Wend** repetition structures, the **Do Until... Loop** structure tests a condition for falsity. Statements in the body of a **Do Until...Loop** are executed repeatedly as long as the loop-continuation test evaluates to False.

An example for **Do Until...Loop** statement. The coding is typed inside the click event of the command button

```
Dim number As Long
number=0
Do Until number > 1000
number = number + 1
Print number
Loop
```

Numbers between 1 to 1000 will be displayed on the form as soon as you click on the command button.

The For...Next Loop

The **For...Next** Loop is another way to make loops in Visual Basic. **For...Next** repetition structure handles all the details of counter-controlled repetition. The following loop counts the numbers from 1 to 100:

```
Dim x As Integer
For x = 1 To 50
Print x
Next
```

In order to count the numbers from 1 yo 50 in steps of 2, the following loop can be used

```
For x = 1 To 50 Step 2
Print x
Next
```

The following loop counts numbers as 1, 3, 5, 7..etc.

The above coding will display numbers vertically on the form. In order to display numbers horizontally the following method can be used.

```
For x = 1 To 50
Print x & Space$ (2);
Next
```

To increase the space between the numbers increase the value inside the brackets after the & Space$.

Following example is a **For...Next** repetition structure which is with the If condition used.

```
Dim number As Integer
For number = 1 To 10
If number = 4 Then
Print "This is number 4"
Else
Print number
End If
Next
```

In the output instead of number 4 you will get the "This is number 4".

A **For...Next** loop condition can be terminated by an **Exit For** statement. Consider the following statement block.

```
Dim x As Integer
For x = 1 To 10
Print x
If x = 5 Then
Print "The program exited at x=5"
Exit For
End If
Next
```

The preceding code increments the value of x by 1 until it reaches the condition x = 5. The **Exit For** statement is executed and it terminates the **For...Next** loop. The Following statement block containing **Do...While** loop is terminated using **Exit Do** statement.

```
Dim x As Integer
Do While x < 10
Print x
x = x + 1
If x = 5 Then
Print "The program is exited at x=5"
Exit Do
End If
Loop
```

With...End With statement

When properties are set for objects or methods are called, a lot of coding is included that acts on the same object. It is easier to read the code by implementing the **With...End With** statement. Multiple properties can be set and multiple methods can be called by using the **With...End With** statement. The code is executed more quickly and efficiently as the object is evaluated only once. The concept can be clearly understood with following example.

```
With Text1
.Font.Size = 14
.Font.Bold = True
.ForeColor = vbRed
.Height = 230
.Text = "Hello World"
End With
```

In the above coding, the object Text1, which is a text box is evaluated only once instead of every associated property or method. This makes the coding simpler and efficient.

Introduction to Arrays

By definition, an array is a variable with a single name that represents many different items. When we work with a single item, we only need to use one variable. However, if we have a list of items which are of similar type to deal with, we need to declare an array of variables instead of using a variable for each item.

We differentiate each item in the array by using subscript, the index value of each item, for example, name(1), name(2), name(3) ....etc. , makes declaring variables more streamline.

Dimension of an Array

An array can be one-dimensional or multidimensional. A one-dimensional array is like a list of items or a table that consists of one row of items or one column of items.

A two-dimensional array is a table of items that make up of rows and columns. The format for a one-dimensional array is ArrayName(x), the format for a two dimensional array is ArrayName(x,y) and a three-dimensional array is ArrayName(x,y,z) . Normally it is sufficient to use a one-dimensional and two-dimensional array, you only need to use higher dimensional arrays if you need to deal with more complex problems. Let me illustrate the arrays with tables

One dimensional Array

Student Name	Name(1)	Name(2)	Name(3)	Name(4)

Two Dimensional Array

Name(1,1)	Name(1,2)	Name(1,3)	Name(1,4)
Name(2,1)	Name(2,2)	Name(2,3)	Name(2,4)
Name(3,1)	Name(3,2)	Name(3,3)	Name(3,4)

Declaring Array

We can use Public or Dim statement to declare an array just as the way we declare a single variable. The Public statement declares an array that can be used throughout an application while the Dim statement declares an array that could be used only in a local procedure.

Declaring one dimensional Array

The general syntax to declare a one dimensional array is as follow:
Dim array Name(subscript) as data Type where subs indicates the last subscript in the array.

When you declare an array, you need to be aware of the number of elements created by the Dim keyword. In the Dim arrayName(subscript) statement, subscript actually is a constant that defines the maximum number of elements allowed. More importantly, subs start with 0 instead of 1. Therefore, the Dim arrangeName(10) statement creates 11 elements numbered 0 to 11. There are two ways to overcome this problem, the first way is by uisng the keyword Option Base 1, as shown in Example 16.1.

Option Base 1
Dim CusName(10) as String
will declare an array that consists of 10 elements if the statement Option Base 1 appear in the declaration area, starting from CusName(1) to CusName(10). Otherwise, there will be 11 elements in the array starting from CusName(0) through to CusName(10)

CusName(1)	CusName(2)	CusName(3)	CusName(4)	CusName(5)
CusName(6)	CusName(7)	CusName(8)	CusName(9)	CusName(10)

The second way is to specify the lower bound and the upper bound of the subscript using To keyword. The syntax is

Dim array Name (lower bound To upper bound) As data Type
Dim Count (100 to 500) as Integer declares an array that consists of the first element starting from Count(100) and ends at Count(500)

Example
```
Dim studentName(1 to 10) As String
Dim num As Integer
Private Sub addName()
For num = 1 To 10
Student Name(num) = InputBox("Enter the student name","Enter Name", "", 1500, 4500)
If student Name(num)<>"" Then
Form1.Print student Name(num)
Else
End
End If
Next
End Sub
```

Declaring two dimensional Array
The general syntax to declare a two dimensional array is as follow:
Dim Array Name (Sub1,Sub2) as data Type

Example
If you wish to compute a summary of students involve in games according to different year in a high school, you need to declare a two dimensional array. In this example, let's say we have 4 games, football, basketball, tennis and hockey and the classes are from year 7 to year 12. We can create an array as follows:

Dim Stu Games (1 to 4,7 to 12) As Integer will create an array of four rows and six columns, as shown in the following table:

Year	7	8	9	10	11	12
Football	StuGames(1,7)	StuGames(1,8)	StuGames(1,9)	StuGames(1,10)	StuGames(1,11)	StuGames(1,12)
Basketball	StuGames(2,7)	StuGames(2,8)	StuGames(2,9)	StuGames(2,10)	StuGames(2,11)	StuGames(2,12)
Tennis	StuGames(3,7)	StuGames(3,8)	StuGames(3,9)	StuGames(3,10)	StuGames(3,11)	StuGames(3,12)
Hockey	StuGames(4,7)	StuGames(4,8)	StuGames(4,9)	StuGames(4,10)	StuGames(4,11)	StuGames(4,12)

Example
In this example, we want to summarize the first half-yearly sales volume for four products. Therefore, we declare a two dimension array as follows:
Dim saleVol(1 To 4, 1 To 6) As Integer
Besides that, we want to display the output in a table form. Therefore, we use a list box. We named the list box listVolume. AddItem is a listbox method to populate the listbox.

The code

```
Private Sub cmdAdd_Click()
Dim prod, mth As Integer ' prod is product and mth is month
Dim saleVol(1 To 4, 1 To 6) As Integer
Const j = 1
listVolume.AddItem vbTab & "January" & vbTab & "February" & vbTab & "March" _& vbTab &
"Apr" & vbTab & "May" & vbTab & "June"
listVolume.AddItem vbTab & "_______________________________"
For prod = 1 To 4
For mth = 1 To 6
saleVol(prod, mth) = InputBox("Enter the sale volume for" & " " & "product" & " " & prod & " "
& "month" & " " & mth)
Next mth
Next prod
For i = 1 To 4

List Volume. Add Item "Product" & "" & i & vb Tab & sale Vol(i, j) & vbTab & sale Vol(i, j + 1) &
vb Tab & sale Vol(i, j + 2) & vb Tab & sale Vol(i, j + 3) & vbTab & sale Vol(i, j + 4) & vb Tab & sale
Vol(i, j + 5)
Next i
End Sub
```

Dynamic Array

Defining the number of elements in an array during design time, is known as static array. However, the problem is sometimes we might not know how many data items to store during run time may not be known. In this case, to use dynamic array where the number of elements will be decided during run time. In VB6, the dynamic array can be resized when the program is executing. The first step in declaring a dynamic array is by using the Dim statement without specifying the dimension list, as follows:

Dim my Array()

Then at run time we can specify the actual array size using the ReDim statement, as follows:

Re Dim my Array(1 to n) when n is decided during run time

A two dimensional array can also be declared using ReDim statement, as follows:

Re Dim my Array(1 to n, 1 to m) when m and n are known during run time

User – Defined Data Types in Visual Basic - 6

Variables of different data types when combined as a single variable to hold several related information's is called a User-Defined data type

A Type statement is used to define a user-defined type in the General declaration section of a form or module.

User-defined data types can only be private in form while in standard modules can be public or private.

An example for a user defined data type to hold the product details is as given below.

```
Private Type Product Details
Prod ID as String
Prod Name as String
Price as Currency
End Type
```

The user defined data type can be declared with a variable using the Dim statement as in any other variable declaration statement. An array of these user-defined data types can also be declared. An example to consolidate these two features is given below.

```
Dim Electronic Goods as Product Details ' One Record
Dim Electronic Goods (10) as Product Details ' An array of 11 records
```

A User-Defined data type can be referenced in an application by using the variable name in the procedure along with the item name in the Type block. Say, for example if the text property of a Text Box namely text1 is to be assigned the name of the electronic good, the statement can be written as given below.

```
Text1.Text = Electronic Goods. ProdName
```

If the same is implemented as an array, then the statement becomes

```
Text1.Text = Electronic Goods (i). Prod Name
```

User-defined data types can also be passed to procedures to allow many related items as one argument.

```
Sub Prod Data ( Electronic Goods as Product Details)
Text1.Text = Electronic Goods. ProdName
Text1.Text = Electronic Goods. Price
End Sub
```

Data Type conversion

In some situation variables has to be converted from one type to another. For example, an automatically generated number may have to be concatenated with a prefix to produce an another string say Employee Id (something like AAA1001). Concatenation can take place only on strings. So, first the number has to be converted into a string, and then it can be concatenated with the prefix. The CStr function converts an expression into a sting. The following example explains this process.

```
Dim Employee Id as String
Dim Auto Num as Integer
Auto Num = (some expression that automatically generates a number)
Employee Id = “AAA” + CStr (Auto Num)
```

The following are the list of type conversion functions.

Conversion Function	**Converts an expression to**
CBool	Boolean
CByte	Byte
CCur	Currency
CDate	Date
CDbl	Double
CInt	Integer
CLng	Long
CSng	Single
CStr	String

Visual Basic Built-in Functions

Many built-in functions are offered by Visual Basic fall under various categories. These functions are procedures that return a value.

1. Date and Time Functions
2. Format Function
3. String Functions

Date and Time Functions

The system's current date and time can be retrieved using the Now, Date and Time functions in Visual Basic. The Now function retrieves the date and time, while Date function retrieves only date and Time function retrieves only the time.

To display both the date and time together a message box is displayed use the statement given below.

MsgBox "The current date and time of the system is" & Now

Here & is used as a concatenation operator to concentrate the string and the Now function. Selective portions of the date and time value can be extracted using the below listed functions.

Funtion	**Extracted Portion**
Year ( )	Year (Now)
Month ( )	Month (Now)
Day ( )	Day (Now)
Week Day ( )	Week Day (Now)
Hour ( )	Hour (Now)
Minute ( )	Minute (Now)
Second ( )	Second (Now)

The calculation and conversion functions related to date and time functions are listed below.

Function	**Description**
DatedAdd ( )	Returns a date to which a specific interval has been added
DateDiff ( )	Returns a Long data type value specifying the interval between the two values
DatePart ( )	Returns an Integer containing the specified part of a given date
DateValue ( )	Converts a string to a Date
DateValue ( )	Converts a string to a Time
DateSerial ( )	Returns a date for specified year, month and day

DateDiff Function

The Date Diff function returns the intervals between two dates in terms of years, months or days. The syntax for this is given below.
Date Diff (interval, date1, date2 [, first day of week[, first week of year]])

Format Function

The format function accepts a numeric value and converts it to a string in the format specified by the format argument. The syntax for this is given below.
Format (expression[, format[, first day of week[, first week of year]]])

The Format function syntax has these parts:

Part	**Description**
Expression	Required any value expression
Format	Optional.A valid named or user-defined format expression.
Firstdayofweek	Optional.A contant that specifies the first day of the week.
Firstweekofyear	Optional.A contant that specifies the first week of the year.

String functions

Function or Statement	**Purpose**
Get Field Value	Returns a substring from a delimited source string
Hex	Returns the hexadecimal representation of a number, as a string
InStr	Returns the position of one string within another
LCase	Converts a string to lower case
Left	Returns the left portion of a string
Len	Returns the length of a string or size of a variable
Like Operator	Compares a string against a pattern
LTrim	Removes leading spaces from a string
Mid Function	Returns a portion of a string
Mid Statement	Replaces a portion of a string with another string
Oct	Returns the octal representation of a number, as a string
Right	Returns the right portion of a string
RTrim	Removes trailing spaces from a string
SetField	Replaces a substring within a delimited target string
Space	Returns a string of space
Str	Returns the string representation of a number
StrComp	Compares two strings
String	Returns a string consisting of a repeated character
Trim	Removes leading and trailing spaces from a string
UCase	Converts a string to upper case

Working with Control

A control is an object that can be drawn on a Form object to enable or enhance user interaction with an application. Controls have properties that define aspects their appearance, such as position, size and colour, and aspects of their behaviour, such as their response to the user input.

Here are some important points about setting up the properties

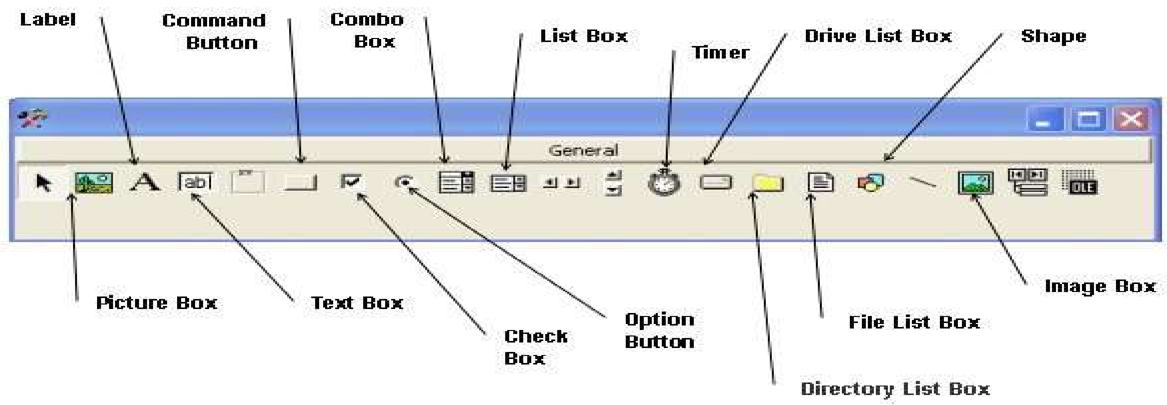
1. Set the Caption Property of a control clearly so that a user knows what to do with that command. For example, in the calculator program, all the captions of the command buttons such as +, - , MC, MR are commonly found in an ordinary calculator, a user should have no problem in manipulating the buttons.
2. Set a meaningful name for the Name Property because it is easier to write and read the event procedure and easier to debug or modify the programs later. The label off.
3. One more important property is whether the control is enabled or not
4. Finally, consider to make the control visible or invisible at runtime, or when should it become visible or invisible.

Tab Index property of Controls

Visual Basic uses the Tab Index property to determine the control that would receive the focus next when a tab key is pressed. Every time a tab key is pressed, Visual Basic looks at the value of the Tab Index for the control that has focus and then it scans through the controls searching for the next highest Tab Index number.

By default, Visual Basic assigns a tab order to control as we draw the controls on the Form, except for Menu, Timer, Data, Image, Line and Shape controls, which are not included in tab order. At run time, invisible or disabled controls also cannot receive the focus although a TabIndex value is given. Setting the TabIndex property of controls is compulsory in development environment.

The below figure is the VB6 toolbox that shows the basic controls.



The Text Box

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

Setting properties to a Text Box

Text can be entered into the text box by assigning the necessary string to the text property of the control
If the user needs to display multiple lines of text in a TextBox, set the MultiLine property to True

1. To customize the scroll bar combination on a Text Box, set the Scroll Bars property.
2. Scroll bars will always appear on the Text Box when it's MultiLine property is set to True and its Scroll Bars property is set to anything except None(0)
3. Using alignment properly ,set its multiline properly is true. If the Multi Line property is set to False, then setting the Alignment property has no effect.

Run-Time Properties of a Text Box control

The Text property is the one you'll reference most often in code, and conveniently it's the default property for the Text Box control. Three other frequently used properties are these:

1. **The Sel Start property** sets or returns the position of the blinking caret (the insertion point where the text you type appears
2. **The Sel Length property** returns the number of characters in the portion of text that has been highlighted by the user, or it returns 0 if there's no highlighted text.
3. **The SelText property** sets or returns the portion of the text that's currently selected, or it returns an empty string if no text is highlighted

The selected text can be copied to the Clipboard by using Sel Text:
Clipboard. SelText text, [format]

In the above syntax, text is the text that has to be placed into the Clipboard, and format has three possible values.

1. VbCFLink - conversation information
2. VbCFRTF - Rich Text Format
3. VbCFText - Text

Set text from the clipboard using the Get Text() function this way:

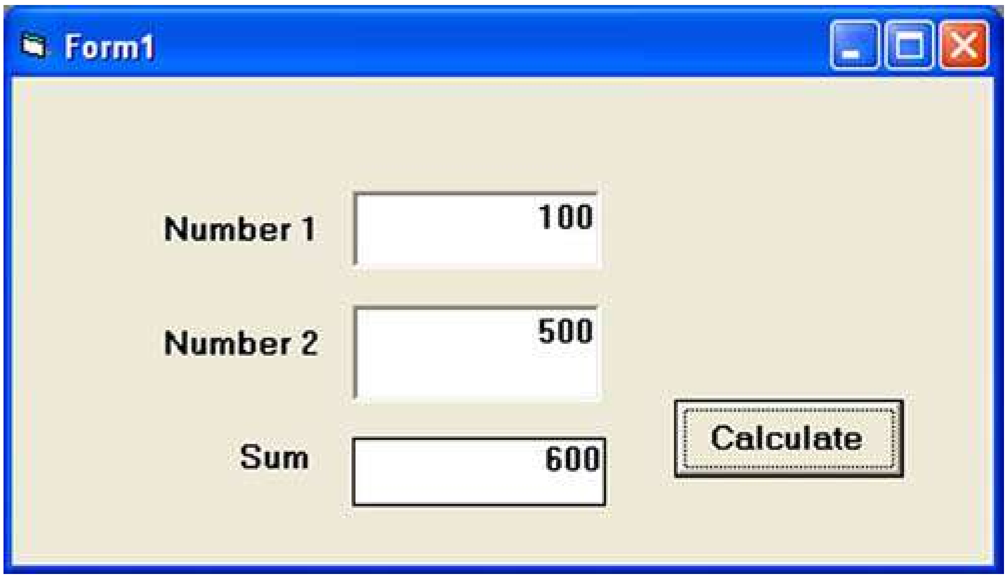
Clipboard. GetText ([format])

Example

In this program, two text boxes are inserted into the form together with a few labels. The two text boxes are used to accept inputs from the user and one of the labels will be used to display the sum of two numbers that are entered into the two text boxes. Besides, a command button is also programmed to calculate the sum of the two numbers using the plus operator. The program use creates a variable sum to accept the summation of values from text box 1 and text box 2.The procedure to calculate and to display the output on the label is shown below.

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

The output is shown in Figure



The Label

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

The Command Button

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

```
Private Sub Command1_Click ()
 Statements
End Sub
```

Properties of a Command Button control

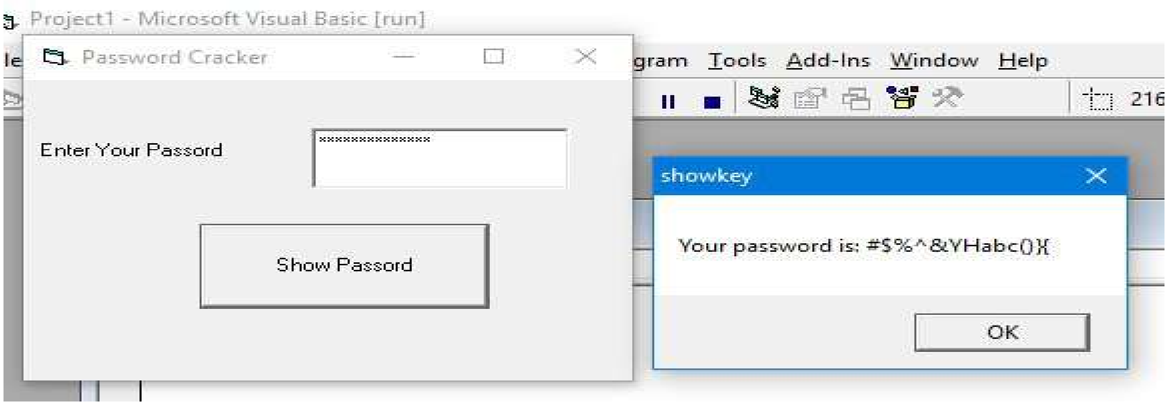
1. To display text on a Command Button control, set its Caption property.
2. An event can be activated by clicking on the Command Button.
3. To set the background colour of the Command Button, select a colour in the Back Color property.
4. To set the text colour set the Fore color property.
5. Font for the Command Button control can be selected using the Font property.
6. To enable or disable the buttons set the Enabled property to True or False
7. To make visible or invisible the buttons at run time, set the Visible property to True or False.
8. Tooltips can be added to a button by setting a text to the Tooltip property of the Command Button.

Example:A Simple Password Cracker

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

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

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


Project1 - Microsoft Visual Basic [run]
Password Cracker
gram Tools Add-Ins Window Help
showkey
OK

Enter Your Passord

Show Passord

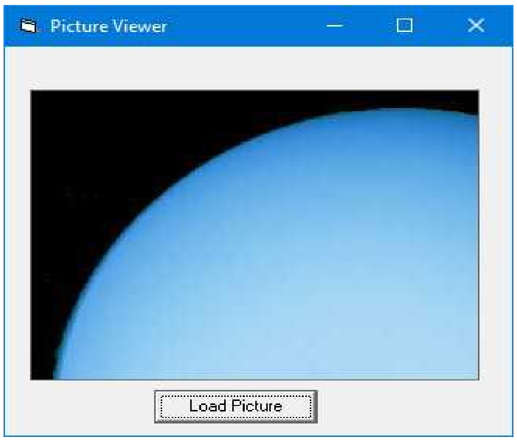
Your password is: #$%^&YHabc()}{

Password Cracker
The password can also be revealed by setting the Password Chr property back to normal mode, as follows:
Private Sub cmd_ShowPass_Click()
Dim yourpassword As String
Txt_Password.PasswordChar = ""
End Sub

The Picture Box
The Picture Box is one of the controls keys that is used to handle graphics. The picture be can loaded a at design phase by clicking on the picture item in the properties window and select the picture from the selected folder. The picture be loaded at runtime, using the **Load Picture** method. For example, the statement will load the picture grape.gif into the picture box.
Picture1.Picture=Load Picture ("C:\VBprogram\Images\grape.gif")

Example:
In this program, insert a command button and a picture box. Enter the following code:
Private Sub cmd_LoadPic_Click()
My Picture. Picture = LoadPicture("C:\Users\admin.DESKTOP-G1G4HEK\Documents\My Websites\vbtutor\vb6\images\uranus.jpg")
End Sub

The path to access the picture must be ensured by correct. Besides that, the image in the picture box is not resizable.



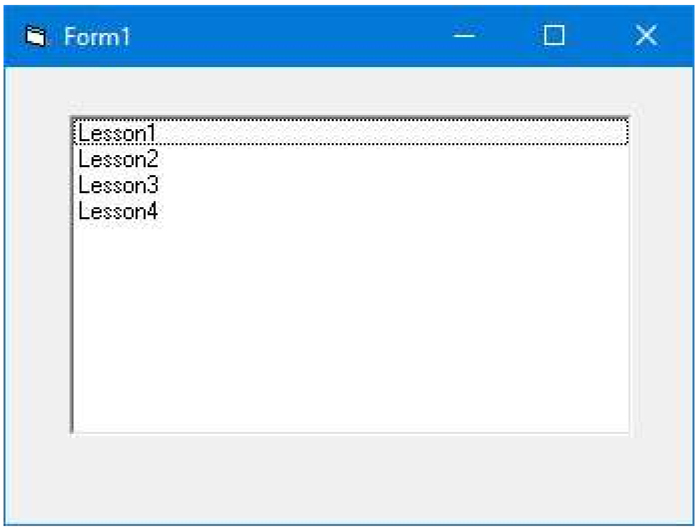
The ListBox

The function of the ListBox is to present a list of items where the user can click and select the items from the list. In order to add items to the list, we can use the **AddItem method**. Here is an example to add 4 items into list box.

Example

```
Private Sub Form_Load ( )
List1.AddItem “Lesson1”
List1.AddItem “Lesson2”
List1.AddItem “Lesson3”
List1.AddItem “Lesson4”
End Sub
```

The Output



```
Lesson1
Lesson2
Lesson3
Lesson4
```

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

The Combo Box

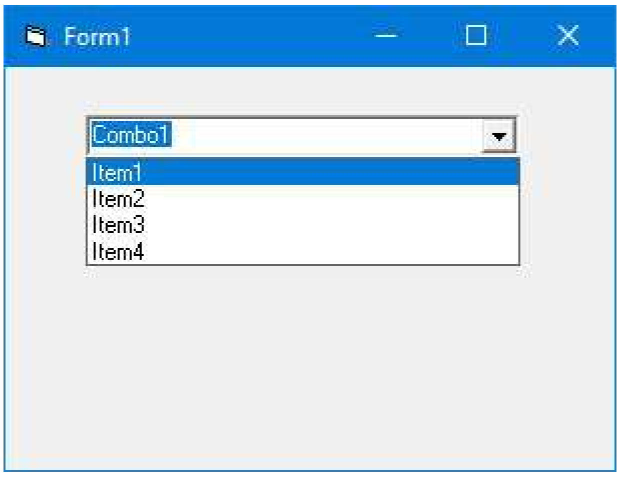
The function of the Combo Box is also to present a list of items where the user can click and select the items from the list.

However, the user needs to click on the small arrowhead on the right of the combo box to see the items which are presented in a drop-down list. In order to add items to the list, you can also use the **Add Item method**

Example

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

The Output



The Check Box

The Check Box control let the user selects or unselects an option. When the Check Box is checked, its value is set to 1 and when it is unchecked, the value is set to 0.

Include the statements Check1.Value=1, to mark the Check Box and Check1.Value=0, to unmark the Check Box, as well as use them to initiate certain actions.

For example, the program will change the background color of the form to red when the check box is unchecked and it will change to blue when the check box is checked.

Example

```
Private Sub Check1_Click ()
 If Check1.Value = 0 Then
 Form1.BackColor = vbRed
 ElseIf Check1.Value = 1 Then
Form1.BackColor = vbBlue
End If
End Sub
```

The Option Button

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

Example:

In the following example, to change the background color of the form according to the selected option.

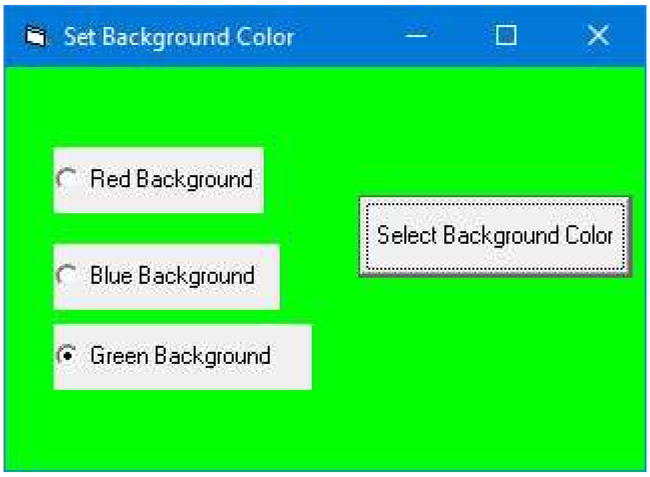
Insert three option buttons and change their captions to "Red Background","Blue Background" and "Green Background" respectively.

Next, insert a command button and change its name to cmd_SetColor and its caption to "Set Background Color". Now, click on the command button and enter the following code in the code window:

```
Private Sub cmd_SetColor_Click()
```

```
If Option1.Value = True Then
Form1.BackColor = vbRed
ElseIf Option2.Value = True Then
Form1.BackColor = vbBlue
Else
Form1.BackColor = vbGreen
End If
End Sub
```

Output :-



Working with Control Arrays:

A control array is a group of controls that share the same name type and the same event procedures.

A control array can be created only at design time, and at the very minimum at least one control must belong to it.

Creahing a Control Array ;

1. Create a control and then assign a numeric, non-negative value to its Index property;
2. Create two controls of the same class and assign them an identical Name property. Visual Basic shows a dialog box warning , that there's already a control with that name and asks whether to create a control array. Click on the Yes button.
3. Select a control on the form, press Ctrl+C to copy it to the clipboard, and then press Ctrl+V to paste a new instance of the control, which has the same Name property as the original one.

Advantages of Control Arrays

1. Controls that belong to the same control array share the same set of event procedures; this often dramatically reduces the amount of code you have to write to respond to a user's actions.
2. New elements can be dynamically added to a control array at run time; in other words, new controls will be created that didn't exist at design time.
3. Elements of control arrays consume fewer resources than regular controls and tend to produce smaller executables.

Creating Controls at Run Time
Control arrays can be created at run time using the following statements

1. Load object (Index %)
2. Unload object (Index %)

Where object is the name of the control to add or delete from the control array. Index % is the value of the index in the array. The control array to be added must be an element of the existing array created at design time with an index value of 0. When a new element of a control array is loaded, most of the property settings are copied from the lowest existing element in the array.

Example:
The application uses a set of Command Button Controls in an array to illustrate the working of a Calculator.

MENUS

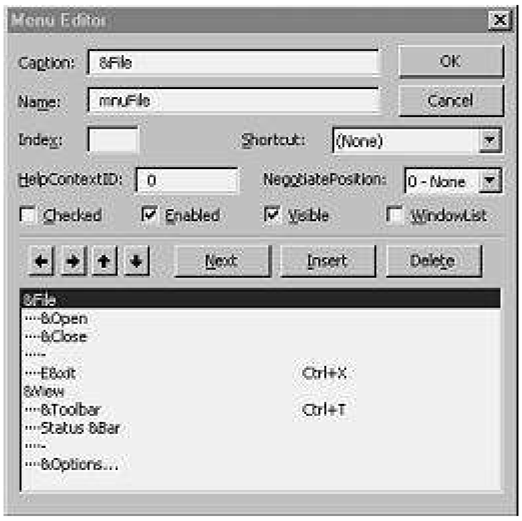
Menus

The menu bar is the standard feature of most Windows applications. The main purpose of the menus is for easy navigation and control of an application. Some of the most common menu items are File, Edit, View, Tools, Help and more. Each item on the main menu bar also provides a list of options in the form of a pull-down menu.
VB programmers can create menus by first selecting the form that will host the menu and then using the VB Menu Editor, as shown in the following Figure .

The Menu Editor is available only when a form is being designed. It is located on the Tools menu in VB6.

1. The first step in creating a menu is to enter the menu item's caption.
2. The caption can incorporate the ampersand (&) for an access key designation, also known as an accelerator key.
3. This enables the user to see an underlined letter in the menu and use the Alt key along with the accelerator key.
4. After the caption of the menu item has been set, the menu requires an object name for programmatic reference. The menu names can be any valid object name.

The Visual Basic Menu Editor



Caption: &File
Name: mnuFile
Index:
Shortcut: (None)
HelpContextID: 0
NegotiatePosition: 0 - None
☐ Checked ☑ Enabled ☑ Visible ☐ WindowList

```
&File
····&Open
····&Close
····-
····E&xit                    Ctrl+X
&View
····&Toolbar                 Ctrl+T
····Status &Bar
····-
····&Options...
```

To control the level and position of the menu item being entered, just use the four direction arrows in the Menu Editor. The up and down arrows reposition menu items for order. The left and right arrows allow menu items to be top-level, sub-level or sub–sub-level.

A separator bar can be implemented in a menu (a horizontal line that visually defines the boundary between two groups of menu items) by putting a single dash (-) as the entire caption for a menu item. This item then becomes the separator bar on the runtime menu.

Menu items can also have their appearance properties set at design time through the Menu Editor. Properties such as Checked, Enabled, Visible, Window List, and a shortcut key can all be specified.

In addition a drop-down list of possible shortcut key combinations allows the programmer to assign a shortcut key to this particular menu item. Windows automatically displays the shortcut key assignment when the menu item is displayed.

Adding a Pop-Up Menu

Pop-up menus, also known as "context menus" or "right-mouse menus," provide the user another convenient way to control the application. Pop-up menus are usually unique for various areas of an application.

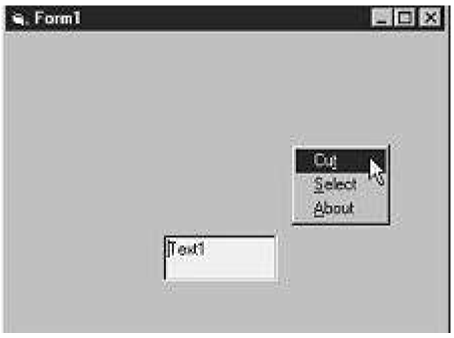
Different pop-up menus are found, depending on the object selected. If the user selects a set of cells in the spreadsheet, he gets a specific pop-up menu. If the user right-clicks on a worksheet tab, he gets a different pop-up menu.

When the pop-up menu is needed, the PopupMenu method is called to activate the pop-up menu. The following code demonstrates this:

Form1.PopupMenu mnuPopUp1

This preceding code calls the Popup Menu method of Form1, as shown in the following Figure:

The menu item mnuPopUp1 is passed as an argument and is displayed at the current mouse location.



Modifying the Appearance of a Menu

When a menu system is created at design time, the programmer can control property settings for menu items.

After these properties are set, they can be altered at runtime to assist the user in interpreting which selections have been made, which items are turned on, and which commands are available.

Syntax used in setting menu controls

```
mnuViewStatusBar.Checked = True
mnuFileOpen.Enabled = False
mnuFormatFontBold.Checked = True
mnuPopUp.Visible = False
mnuViewToolbar.Checked = True
```

Mouse Events

Visual Basic responds to various mouse events, which are recognized by most of the controls.

The main events are

1. Mouse Down
2. Mouse Up and
3. Mouse Move.

Mouse Down --- occurs when the user presses any mouse button
Mouse Up --- occurs when the user releases any mouse button.
These events use the arguments button, Shift, X, Y.
The first argument is an integer called Button. The value of the argument indicates whether the left, right or middle mouse button was clicked.
The second argument in an integer called shift. The value of this argumnet indicates whether the mouse button was clicked simultaneously with the Shift key, Ctrl key or Alt key.
The third and fourth arguments X and Y are the coordinates of the mouse location at the time the mouse button was clicked.
As the Form_MouseDown( ) is executed automatically whenever the mouse button is clicked inside the Form's area the X, Y co-ordinates are referenced to the form.

Graphical Mouse Applications

The mouse events can be combined with graphics methods and any number of customized drawing or paint applications can be created.

Example:-

The following code is in the Form_Mouse Down ( ) procedure, Form_Mouse Move ( ) procedure and cmdClear_Click ( ) procedures respectively.

```
Private Sub cmdClear_Click()
frmDraw.Cls
End Sub
Private Sub Form_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)
frmDraw.CurrentX = X
frmDraw.CurrentY = Y
End Sub
Private Sub Form_Mouse Move(Button As Integer, Shift As Integer, X As Single, Y As Single)
If Button = 1 Then
Line (frmDraw.CurrentX, frmDraw.CurrentY)-(X, Y)
End If
End Sub
```

Mouse Move application

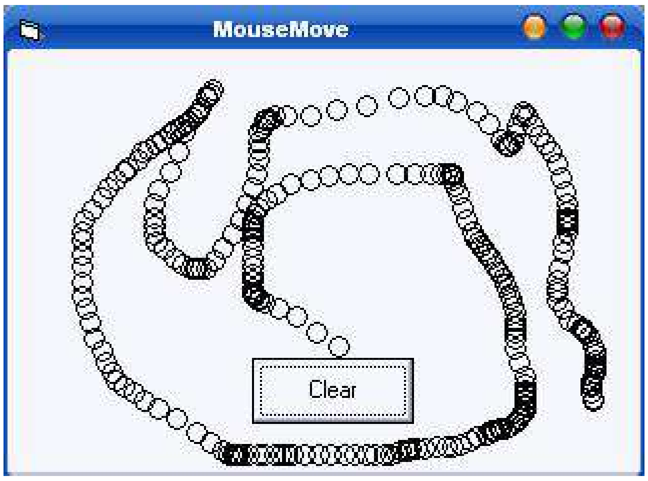
Visual Basic does not generate a MouseMove event for every pixel the mouse moves over and a limited number of mouse messages are generated per second by the operating environment.

The following application illustrates how often the Form_MouseMove ( ) event is executed.

```
Private Sub cmdClear_Click()
frm Mouse Move.Cls
End Sub
```

```
Private Sub Form_Mouse Move(Button As Integer, Shift As Integer, X As Single, Y As Single)
Circle (X, Y), 70
End Sub
```

The above procedure simply draws small circles at the mouse's current location using the Circle method. The parameter x, y represent the centre of the circle, and the second parameter represent the radius of the circle.



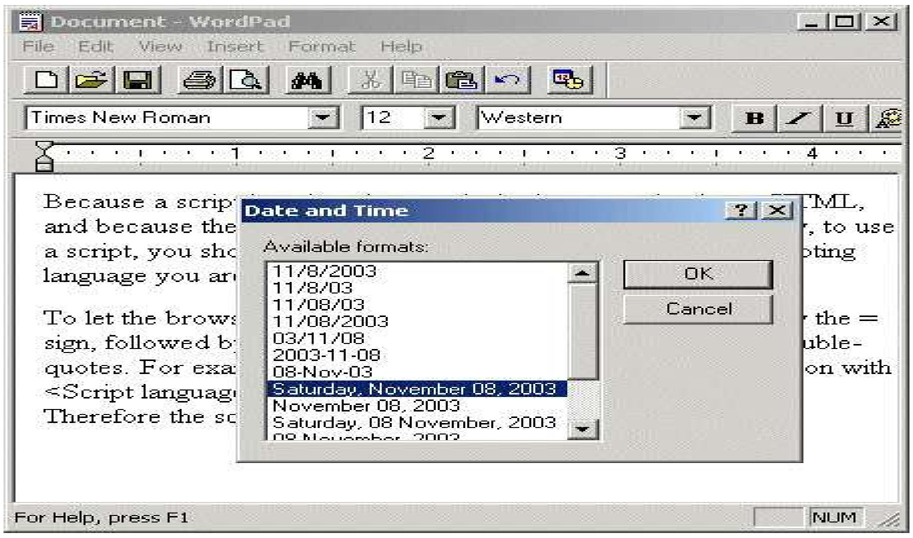
Dialog Boxes
Dialog boxes are used to interact with the user and retrieve information.
Dialog Boxes are either Model or Model less**.**

Modal dialog box

Modal dialog boxes are generally used inside a program, to display messages, and to set program parameters. Modal dialog boxes come to the front of the screen, and you may not use the program while the modal dialog box is open. To continue using the program, the modal dialog box must be closed.

Example:

The Date and Time dialog box of WordPad is modal: when it is displaying, the user cannot use any other part of WordPad unless he or she closes this object first



11/8/2003
11/8/03
11/08/03
11/08/2003
03/11/08
2003-11-08
08-Nov-03
Saturday, November 08, 2003
November 08, 2003
Saturday, 08 November, 2003

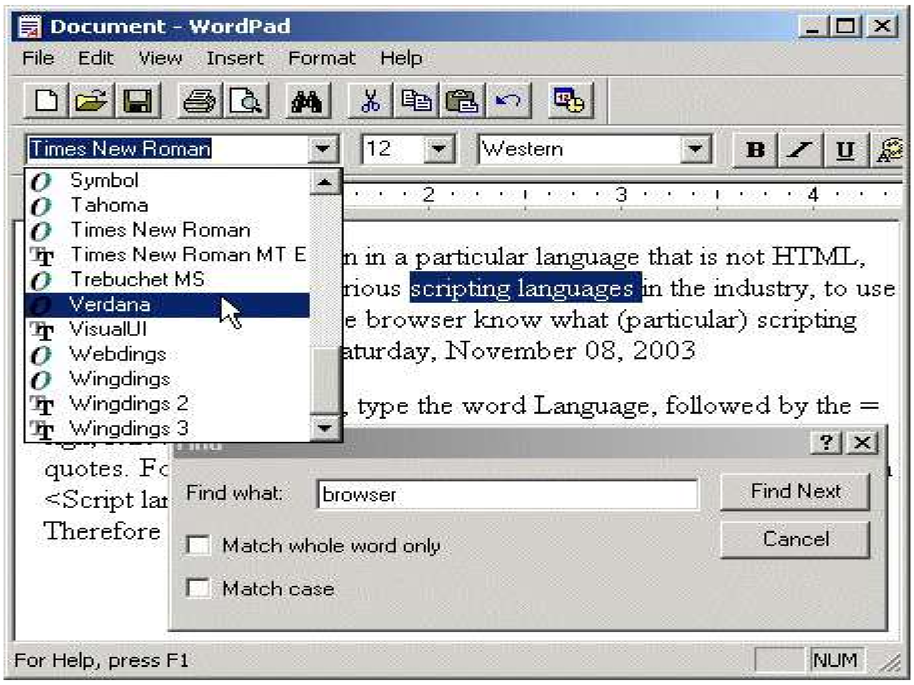
After creating a dialog used as an addition to an existing form or an existing dialog box, to call it as modal, use the **Show Dialog()** method.

Modeless dialog box

Modeless dialog boxes are able to be deselected, and control can be taken away from a modeless dialog box and transferred to some other window. Modeless dialog boxes are frequently used as a fast and easy way to create a window, without having to register a window class. Modeless dialog boxes are common in the Windows control panel.

Example:

The Find (and the Replace) dialog box of WordPad (also the Find and the Replace dialog boxes of most applications) is an example of a modeless dialog box. If it is opened, the user does not have to close it in order to use the application or the document in the background.



Since the modeless dialog box does not display its button on the task bar, the user should know that the dialog box is opened.

There are three ways of adding dialog boxes to an application. Which are as follows Predefined Dialog Boxes – created using Input Box () and Message Box () Function Custom Dialog Boxes – Created by adding controls to the form or by customizing an existing dialog box Standard Dialog Boxes – Created using Common Dialog Control

Predefined Dialog Boxes

1. Besides building your own dialog boxes, VB allows you to use some default dialog boxes of various kinds.
2. Some are predefined by Windows; others are simple dialog boxes.
3. These Dialog boxes are always model.
4. These uses the Msg Box and Input Box functions.

Using Messagebox() Function

The MessageBox method of the Application object allows to can also be provided specify both the message and the caption; Various buttons and features. This is a simple and direct encapsulation of the MessageBox function of the Windows API, which passes as a main window parameter the handle of the Application object. This handle is required to make the message box behave like a modal window.

Syntax :
MsgBox (Prompt [,icons+buttons] [,title] **)**
memory_variable = MsgBox (prompt [, icons+ buttons] [,title] **)**

Prompt:

String expressions displayed as the message in the dialog box. If prompt consist of more than one line, that can be separated by using the **vbrCrLf** constant.

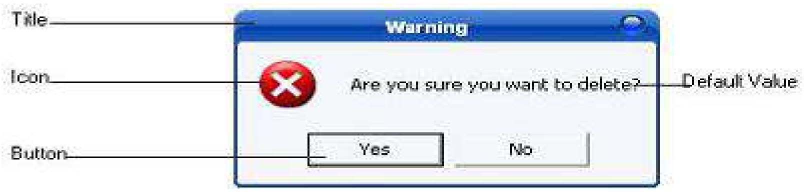
Icons + Buttons:

Numeric expression that is the sum of values specifying the number and type of buttons and icon to display.

Title:

String expression displayed in the title bar of the dialog box. If the title is omitted , the application name is placed in the title bar.

Following is an expanded Message Box:



Using Input box() Function

The Input Box function asks the user to input a string. You provide the caption, the query, and a default string. The Input Query function asks the user to input a string, too. The only difference between this and the Input Box function is in the syntax. The Input Query function has a Boolean return value that indicates whether the user has clicked OK or Cancel.

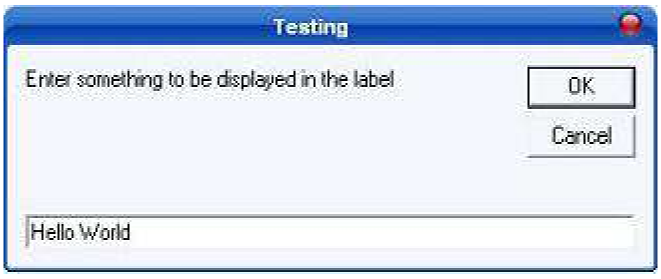
Syntax:
memory_variable = InputBox (prompt[,title][,default])
memory_variable is a variant data type but typically it is declared as string, which accept the message input by the users. The arguments are explained as follows:
Prompt:
String expression displayed as the message in the dialog box. If prompt consists of more than one line, it can be separated by using the vbCrLf constant

Title:
String expression displayed in the title bar of the dialog box. If the title is omitted , the application name is displayed in the title bar

Default - text:
The default text that appears in the input field where users can use it as his intended input or he may change to the message he wish to key in.
X - position and y – position: The position or the coordinate of the input box.
Following code is entered in **cmdOK_Click ( )** event

```
Private Sub cmdok_Click()
Dim ans As String
ans = InputBox("Enter something to be displayed in the label", "Testing", 0)
If ans = "" Then
lbl2.Caption = "No message"
Else
lbl2.Caption = ans
End If
End Sub
```

Save and run the application. As soon as you click the OK button you will get the following InputBox


Testing
Enter something to be displayed in the label
Hello World

Custom Dialog Boxes

1. CUSTOM Dialog boxes are customized by the user.
2. The appearance of the form is customized by setting the property values.
3. Custom dialog box is a form that is created containing control including command button, option button and textbox controls that supply information to the application

Common Dialog Control

1. The Common Dialog Box Library contains a set of dialog boxes for performing common application tasks, such as opening files, choosing color values, and printing documents.
2. A particular dialog box is displayed by using one of the six "Show..." methods of the Common Dialog control: **ShowOpen**, **ShowSave**, **ShowPrinter**, **ShowColor**, **ShowFont**, or **ShowHelp.**
3. It is an "Active X" control that must be added to the toolbox via the **Components** dialog box.
4. This dialog box is accessed via the **Project** menu, **Components** item.Once you check "Microsoft Common Dialog Control 6.0" and click OK, the control is added to your toolbox.
5. Then you can double-click it to make it appear on your form, as you would with any other control.
6. The Common Dialog control is not visible at run-time.

EXAMPLE:

To Change the font name, font size, Color using Common Dialog control

```
Private Sub Option1_Click()
Text1.FontBold = True
End Sub
Private Sub Option2_Click()
Text1.FontItalic = True
End Sub
Private Sub Option3_Click()
Text1.FontUnderline = True
End Sub
Private Sub Option4_Click()
Text1.FontBold = False
Text1.FontItalic = False
Text1.FontUnderline = False
End Sub
Private Sub Command1_Click()
CommonDialog1.ShowFont
Text1.FontName = CommonDialog1.FontName
End Sub
Private Sub Command2_Click()
CommonDialog1.ShowFont
Text1.FontSize = CommonDialog1.FontSize
End Sub
Private Sub Command3_Click()
CommonDialog1.ShowColor
Text1.ForeColor = CommonDialog1.Color
End Sub
Private Sub Command4_Click()
Text1.Text = ""
End Sub
Private Sub Command5_Click()
End

End Sub
```

GRAPHICS FOR APPLICATION:

Graphics forms a very important part of visual basic programming because an attractive user interface will be appealing to the users. In the old BASIC, drawing and designing graphics are considered difficult jobs, as they have to be programmed line by line in a text-based environment. However, in Visual Basic 6, these jobs have been made easy. There are two approaches for creating graphics for an application they are

1. Graphical Controls
2. Graphics methods

GRAPHICAL CONTROLS:

There are four basic controls in VB6 that can be used to draw graphics on form

1. The line control
2. The shape control
3. Image box
4. Picture box

The line controls:

To draw a straight line, just click on the line control and then use button mouse to draw the line on the form. After drawing the line, you can then change its color, width and style using the Border Color, Border Width and Border Style properties.

The shape controls:

To draw a shape, just click on the shape control and draw the shape on the form. The default shape is a rectangle, with the default shape property set at 0.The shape to square, oval, circle and rounded rectangle can be changed by changing the shape property's value to 1, 2, 3 , 4, and 5 respectively. In addition,Its background colour can be changed by using the Back Color property, its border style using the Border Style property, its border colour using the Border Color property as well its border width using the Border Width property.

The image box and picture box:

To load a picture or image into an image box or a picture box, you can click on the picture property in the properties window to launch a dialog box that will prompt you to select a certain picture file. At runtime by using the LoadPictrure ( ) method. The syntax is picture can be also be loaded

Image1.Picture= Load Picture ("C:\pathname\picture file name")
picture1.Picture= Load Picture ("C:\pathname\picture name")

Graphics methods:

There are four graphics methods available in visual basics. They are:

1. Pset method
2. Line method
3. CLS
4. Circle drawing methods
5. Point

The Pset method:

The Pset method draw a dot on the screen, it takes the syntax
Pset (x , y), color
The Pset method can also be used to draw a straight line on the form. The procedure is
Example Code:-

```
Private Sub cmdShow_Click()
DrawWidth = 10
PSet (100, 500)
 End Sub
```

Relative positioning with the Step keyword

The Step keyword allows you to draw in a position relative to the current position. See the example.

Code:

```
Private Sub cmdShow_Click()
 DrawWidth = 10
 CurrentX = 500
CurrentY = 500
PSet Step(0, 0)
 End Sub
```

The above code draws a point in the (0, 0) position relative to the current position that is (500, 500).

That means, the point is drawn in the (500, 500) position. But this is (0, 0) position relative to the current position.

The line method:

The line method is used to draw a straight line. This method is faster than the Pset method. The syntax for Line method is as follows. It draws a line from the point (x1, y1) to the point (x2, y2) and the color constant will determine the color of the line.

```
Line (x1, y1)-(x2, y2), color
```

Example Code:

```
Private Sub cmdShow_Click()
 DrawWidth = 5
'A hyphen is required between the points
 Line (0, 0)-(2000, 2000), vbBlue
 End Sub
```

The circle method:

The circle method uses the following syntax

```
Circle (x1, y1),radius, color
```

to draws a circle centered at (x1, y1), with a certain radius and a certain border color.

For example, the procedure

```
Circle (400, 400),500, VbRed
```

draws a circle centered at (400, 400) with a radius of 500 twips and a red border.

Example Code:-

```
Private Sub cmdShow_Click()
FillStyle = vbSolid
FillColor = &H80C0FF
Draw Width = 3
Circle (1800, 1800), 1000, vbRed
End Sub
```

CLS method:

The Cls method is another simple graphic method that is used to clear the surface of the form or the Picture Box control. If some texts are present, Cls method. It clears any drawing created by the graphic methods. It will be cleared by using

Form1.cls

Clears all the graphics present in form 1.

Point method:

The point method returns the colour of a particular pixel. For example the following statement is used to find the colour of the pixel at location 30,40.

Pixel color = point (30,40)

MULTIPLE DOCUMENT INTERFACE (MDI):

The Multiple Document Interface (MDI) was designed to simplify the exchange of information among documents, all under the same roof.

Creating MDI Form

1. The Multiple Document Interface (MDI) was designed to simplify the exchange of information among documents, all under the same roof.
2. With the main application, you can maintain multiple open windows, but not multiple copies of the application.
3. Data exchange is easier when you can view and compare many documents simultaneously. You almost certainly use Windows applications that can open multiple documents at the same time and allow the user to switch among them with a mouse-click.
4. The main Form, or MDI Form, isn't duplicated, but it acts as a container for all the windows, and it is called the parent window.
5. The windows in which the individual documents are displayed are called Child windows.

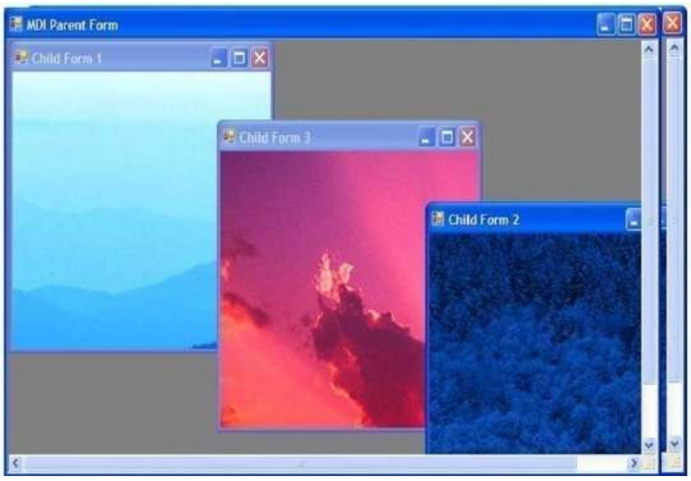
An MDI application must have at least two Form, the parent Form and one or more child Forms.

Each of these Forms has certain properties. There can be many child forms contained within the parent Form, but there can be only one parent Form.

Steps for creating an MDI application:

1. Start a new project and then choose Project >>> Add MDI Form to add the parent Form.
2. Set the Form's caption to MDI Window
3. Choose Project >>> Add Form to add a SDI Form.
4. Make this Form as child of MDI Form by setting the MDI Child property of the SDI Form to True.

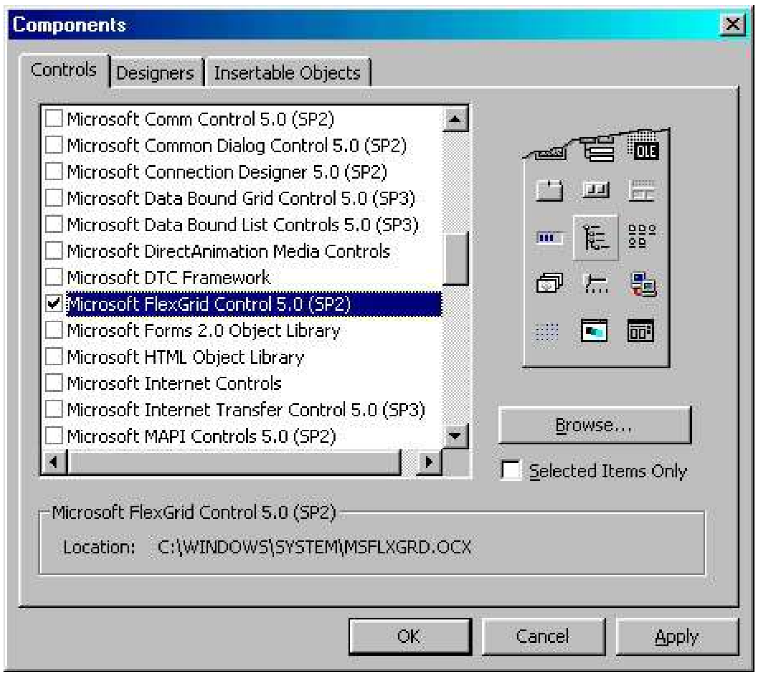
Set the caption property to MDI Child window.



Flex grids:
Using Microsoft's Flex Grid control (MSFLXGRD.OCX) create utilities to display, filter, edit, validate and update the data. For example, such utilities could include:

1. data entry & validation
2. high level reports
3. ported spreadsheet macro applications retaining cell layout & format

Using the Flex Grid
To use the Flex Grid in your application, right click the Toolbox or select the Project menu and choose 'Components'. The Components dialog box appears as shown in Figure 1.


Components

Controls | Designers | Insertable Objects

- [ ] Microsoft Comm Control 5.0 (SP2)
- [ ] Microsoft Common Dialog Control 5.0 (SP2)
- [ ] Microsoft Connection Designer 5.0 (SP2)
- [ ] Microsoft Data Bound Grid Control 5.0 (SP3)
- [ ] Microsoft Data Bound List Controls 5.0 (SP3)
- [ ] Microsoft DirectAnimation Media Controls
- [ ] Microsoft DTC Framework
- [x] Microsoft FlexGrid Control 5.0 (SP2)
- [ ] Microsoft Forms 2.0 Object Library
- [ ] Microsoft HTML Object Library
- [ ] Microsoft Internet Controls
- [ ] Microsoft Internet Transfer Control 5.0 (SP3)
- [ ] Microsoft MAPI Controls 5.0 (SP2)

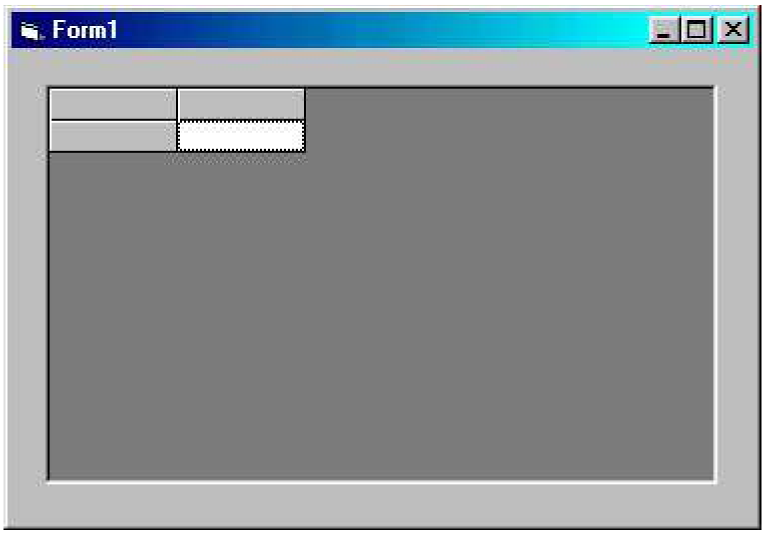
Browse...

- [ ] Selected Items Only

Microsoft FlexGrid Control 5.0 (SP2)
Location: C:\WINDOWS\SYSTEM\MSFLXGRD.OCX

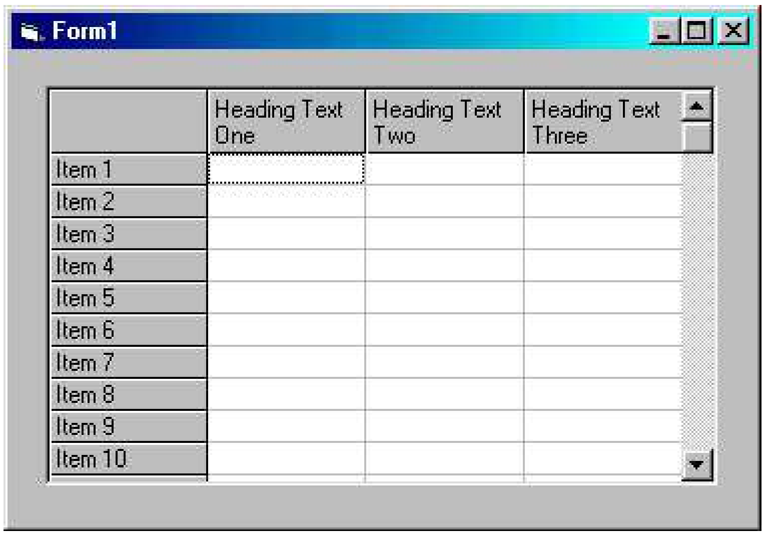
OK | Cancel | Apply

Select Microsoft Flex Grid Control and copy the control onto your form from the Toolbox. The default Flex Grid will appear on the form as shown in the following Figure.



An example of grid initialisation code would be as follows:

```
Dim lng Width As Long
Dim intLoop Count As Integer
Const SCROLL_BAR_WIDTH = 320
With MSFlex Grid
Lng Width = .Width - SCROLL_BAR_WIDTH
.Cols = 4
.FixedCols = 1
.Rows = 0
.Add Item vb Tab& "Heading Text One" &vbTab&"Heading Text Two" &vbTab& "Heading Text
Three" &vbTab& "Heading Text Four"
.Rows = 12
.FixedRows = 1
.WordWrap = True
.RowHeight(0) = .RowHeight(0) * 2
.ColWidth(0) = lngWidth / 4
.ColWidth(1) = lngWidth / 4
.ColWidth(2) = lngWidth / 4
.ColWidth(3) = lngWidth / 4
For intLoopCount = 1 To (.Rows - 1)
.TextMatrix(intLoopCount, 0) = "Item " &intLoopCount
Next intLoopCount
End With
```



	Heading Text One	Heading Text Two	Heading Text Three
Item 1			
Item 2			
Item 3			
Item 4			
Item 5			
Item 6			
Item 7			
Item 8			
Item 9			
Item 10			

Entering Values in the Columns:
The rest of the cells are filled with values by calling a procedure Fill values in the form_load procedure.
The following code is entered in the procedure Fill values ().

```
Public sub Fill values ()
Dim cc,rc
For cc = 1 to 3 step 1
Flexgrid.col = cc
For rc = 1 to 6 step 1
Fexgrid.row = rc
Flex grid. Text = "Nil"
Next
Next
End sub
```

Changing Cell Background & Foreground Colours:-
The ability to change the individual cell background colour and cell font colour is useful for highlighting particular data states, e.g. values exceeding a specified ceiling, negative values, cells where data cannot be entered etc.

To set the background colour of a cell, first reference the cell by setting the Row and Col properties to the appropriate cell, then use the Cell Back Color property to set its colour.

Similarly use the Cell Fore Color property to change the colour of the text displayed in a cell.

Examples of available colours are found in the Flex Grid's Properties Window under any of the colour properties, especially the palette tab as this displays a wider range of colours.

To set a range of cells to a selected colour each cell must be referenced in turn, unless all the cells in the grid are to have their colour set to the same colour in which case the Back Color and Fore Color property can be used which sets the entire grid's colour.

Rich Text Box Control

The Rich Text Box control allows the user to display, enter, and edit text while also providing more advanced formatting features than the conventional Text Box control.

General Description

The Rich Text Box control provides a number of properties. By using there properties formatting to any portion of text within the control. To change the formatting of text, it must first be selected. Only selected text can be assigned character and paragraph formatting. Using these properties, you can make text bold or italic, change the color, and create superscripts and subscripts. You can also adjust paragraph formatting by setting both left and right indents, as well as hanging indents.

The Rich Text Box control opens and saves files in both the RTF format and regular ASCII text format.Methods of the control can be used (Load File and Save File) to directly read and write files, or use properties of the control such as SelRTF and TextRTF in conjunction with Visual Basic's file input/output statements.

OPEN DATABASE CONNECTIVITY - ODBC

Open Database Connectivity

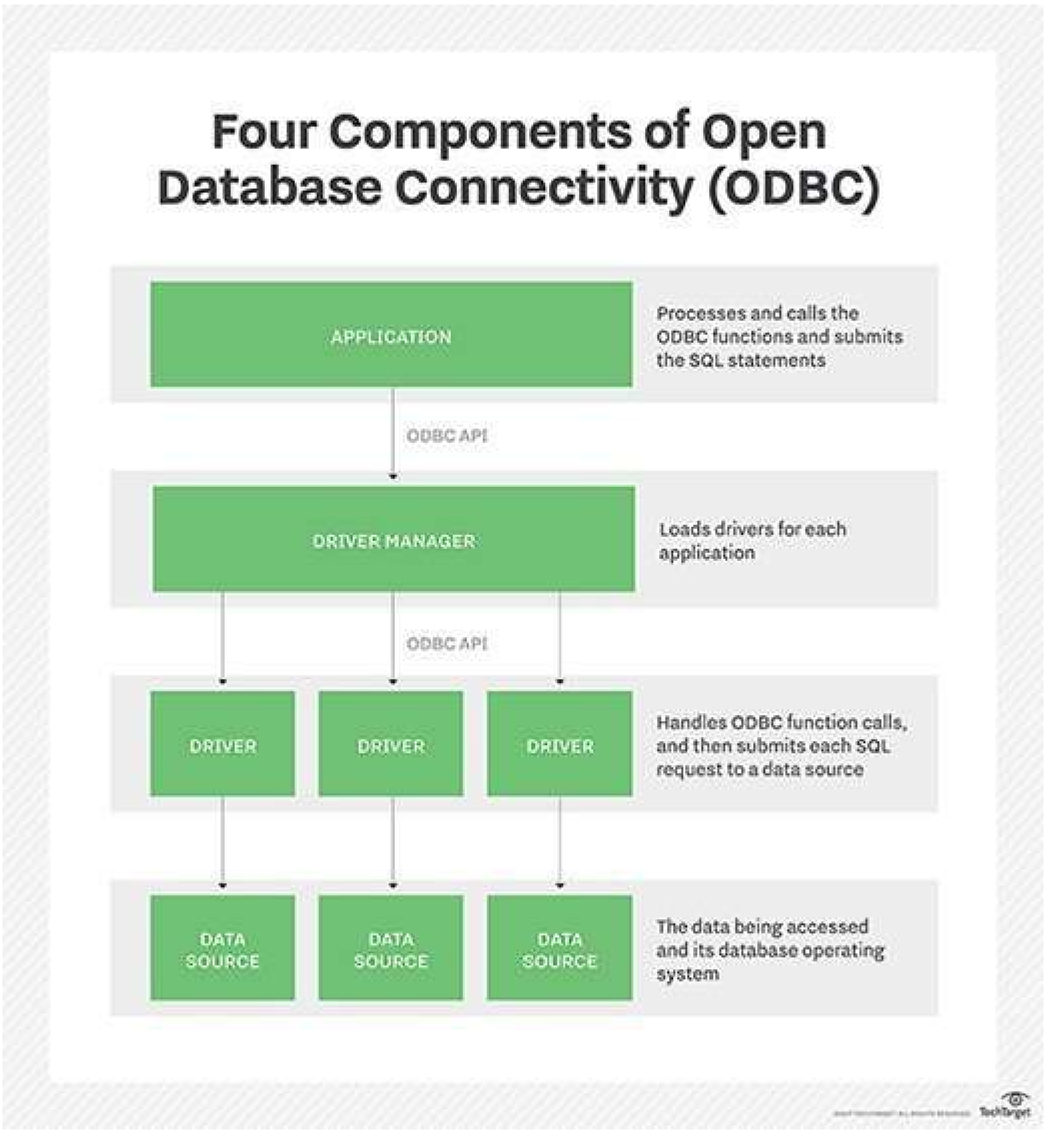
Open Database Connectivity (ODBC) is an open standard application programming interface (API) that allows application programmers to access any database.
The four different components of ODBC are:

Application:
Processes and calls the ODBC functions and submits the SQL statements;

Driver manager:
Loads drivers for each application;

Driver:
Handles ODBC function calls, and then submits each SQL request to a data source;

Data source:
The data being accessed and its database management system (DBMS) OS


Four Components of Open Database Connectivity (ODBC)
APPLICATION
Processes and calls the ODBC functions and submits the SQL statements
ODBC API
DRIVER MANAGER
Loads drivers for each application
ODBC API
DRIVER
DRIVER
DRIVER
Handles ODBC function calls, and then submits each SQL request to a data source
DATA SOURCE
DATA SOURCE
DATA SOURCE
The data being accessed and its database operating system
TechTarget

ODBC Driver

An ODBC driver uses the Open Database Connectivity (ODBC) interface by Microsoft that allows applications to access data in database management systems (DBMS) using SQL as a standard for accessing the data.

ODBC permits maximum interoperability, which means a single application can access different DBMS. Application end users can then add ODBC database drivers to link the application to their choice of DBMS.

ODBC data sources

A data source is a source of data combined with the connection information that is required to access that data. Examples of data sources are SQL Server, Oracle RDBMS, a spreadsheet, and a text file. Examples of connection information include server location, database name, logon ID, password, and various ODBC driver options that describe how to connect to the data source. This information can be obtained from the administrator of the database to which you want to connect.

In the ODBC architecture, an application such as Access connects to the ODBC Driver Manager, which in turn uses a specific ODBC driver (for example, Microsoft SQL ODBC driver) to connect to a data source. In Access, you use ODBC data sources to connect to data sources external to Access that do not have built-in drivers.

To connect to these data sources, to steps ;

1. Install the appropriate ODBC driver on the computer that contains the data source.
2. Define a data source name (DSN) by using either the **ODBC Data Source Administrator** to store the connection information in the Microsoft Windows registry or a DSN file, or a connect string in Visual Basic code to pass the connection information directly to the ODBC Driver Manager.

Add an ODBC Data Source

Before proceeding, obtain and install the appropriate ODBC driver for the data source to which you want to connect.

1. Click **Start**, and then click **Control Panel**.
2. In the Control Panel, double-click **Administrative Tools**.
3. In the Administrative Tools dialog box, double-click **Data Sources (ODBC)**.

The ODBC Data Source Administrator dialog box appears.

1. Click **User DSN**, **System DSN**, or **File DSN**, depending on the type of data source you want to add.
2. Click **Add**.
3. Select the driver that you want to use, and then click **Finish** or **Next**.

If the driver is not listed, contact the administrator of the database and get the information about how to obtain the correct driver.

Follow the instructions and enter the required connection information in any dialog boxes that follow.

Data Access Using DAO

A DAO-based application uses the following operations to access a data source:

1. **Create the workspace** Defines the user session, including user identification, password, and database type (such as Microsoft Jet or ODBC).
2. **Open the database** specifies a connection string for a particular Workspace object, with information such as data source name and database table.
3. **Open the record set** Runs an SQL query (with or without parameters) and populates the record set.
4. **Use the record set** The query result set is now available to your application. Depending on the cursor type, you can browse and change the row data.
5. **Close the record set** Drops the query results and closes the record set.
6. **Close the database** Closes the database and releases the connection.

Use DAO to perform DDL (Data Definition Language) operations that affect the structure of your database. For example, you can create, delete, and modify the table definitions.

Using ODBC with DAO

The properties and methods can be used in the conjuction with the ODBC type declaration. The Open Data base method creates a connection between the application and the ODBC data base.

Syntax:

Dim db as Database
Set db = OpenDatabase("<Data Source name>",<dbdriverpromptinformation>,
<Readnly>, "ODBC;UID=<User ID>;PWD=<Password>")
Db is a variable that represents the database object.

Example:

Finding a Specific record

A particular record from a table can also be viewed using Find method.
Each of the **Find** methods begins its search from the location and in the direction specified in the following table.

TABLE 2

Find method	**Begins searching at**	**Search direction**
Find First	Beginning of record set	End of record set
Find Last	End of record set	Beginning of record set
Find Next	Current record	End of record set
Find Previous	Current record	Beginning of record set

Using one of the **Find** methods isn't the same as using a **Move** method, however, which simply makes the first, last, next, or previous record current without specifying a condition.a Find operation can be followed with a Move operation.

Adding a field to a table:

A field can be added to an existing table of a database at run time.
The following example code adds a new field called Address to the emp table of the employee database.

Adding an index to a table:

The Append method can be used to add a new Index object. The following code adds a new index to the emp table for the column ename of the employee database.

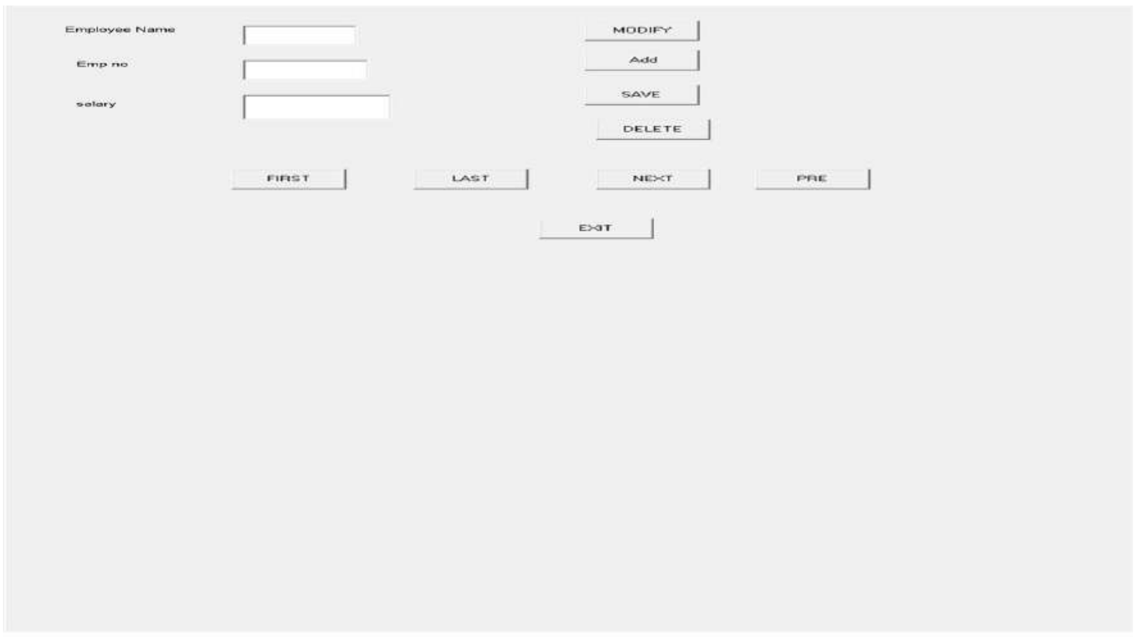
EXAMPLE CODE :-

```
Dim db As Database
Dim rs As Recordset
Public Sub movefields()
Text1 = rs("en")
Text2 = rs("eno")
Text3 = rs("sal")
End Sub
Private Sub cmdAdd_Click()
Text1.Text = ""
Text2.Text = ""
Text3.Text = ""
rs.AddNew
Text1.SetFocus
End Sub

Private Sub CMDDEL_Click()
rs.Delete
rs.MoveNext
If rs.EOF Then
rs.MoveLast
End If
movefields
End Sub
Private Sub CMDE_Click()
End
End Sub
Private Sub CMDMOD_Click()
If rs.EditMode = dbEditNone Then
rs.Edit
End If
End Sub
Private Sub CMDPRE_Click()
rs.MovePrevious
If rs.BOF Then
rs.MoveFirst
End If
movefields
End Sub
Private Sub cmdsave_Click()
If rs.EditMode = dbEditAdd Then

rs("en") = Text1.Text
rs("eno") = Text2.Text
rs("sal") = Text3.Text
```

```
End If
rs.Update
MsgBox ("RECORD SUCCESSFULLY ADDED")
End Sub
Private Sub Command2_Click()
rs.MoveFirst
movefields
End Sub
Private Sub Command3_Click()
rs.MoveLast
movefields
End Sub
Private Sub Command4_Click()
rs.MoveNext
If rs.EOF Then
rs.MoveLast
End If
movefields
End Sub
Private Sub Form_Load()
Dim NDB As TableDef
Dim NFLD As Field
Set db = OpenDatabase("c:\users\admin\documents\emplo.mdb")
'Set rs = db.OpenRecordset("emp", dbOpenDynaset)
Set NDB = db.TableDefs("emp")
Set NFLD = NDB.CreateField("address", dbText, 30)
NDB.Fields.Append NFLD
MsgBox ("FIELD ADDED")
db.Close
Text1.Text = rs.Fields("en")
Text2.Text = rs.Fields("eno")
Text3.Text = rs.Fields("sal")
End sub
```



Data Access Using Data Control

The data control is the primary interface between a Visual Basic application and a database. It can be used without writing any code at all. It can be a central part of a complex database management system. This icon may not appear in your Visual Basic toolbox. If it doesn't, select Project from the main menu, then click Components. The Components window will appear. Select Microsoft Data Control, then click OK. The control will be added into toolbox. This control is suitable for small databases.

The data control (or tool) can access databases created by several other programs besides Visual Basic (or Microsoft Access). Some other formats supported include Btrieve, dBase, FoxPro, and Paradox databases.

The data control can be used to perform the following tasks:

a. Connect to a database.
b. Open a specified database table.
c. Create a virtual table based on a database query.
d. Pass database fields to other Visual Basic tools, for display or editing. Such tools are bound tools (controls), or data aware.
e. Add new records or update a database.
f. Trap any errors that may occur while accessing data.
g. Close the database.

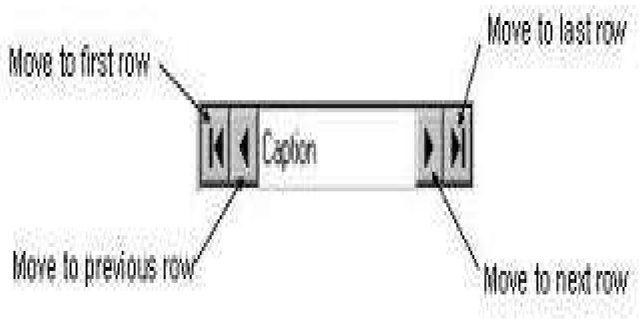
Data Control Properties:

Align	-	**Determines where data control is displayed.**
Caption	-	**Phrase displayed on the data control.**
Connection String	-	**Contains the information used to establish a connection to a Data base**
Lock Type	-	**Indicates the type of locks placed on records during editing (Default setting makes databases read-only).**
Record set	**-**	**A set of records defined by a data control's Connection String**

and Record Source properties. Run-time only

Record Source	-	**Determines the table and the data control is attached to.**

1. As a rule, you need one data control for every database table, or virtual table, you need access to. One row of a table is accessible to each data control at any one time. This is referred to as the current record.
2. When a data control is placed on a form, it appears with the assigned caption and four arrow buttons:



The arrows are used to navigate through the table rows (records). As indicated, the buttons can be used to move to the beginning of the table, the end of the table, or from record to record.

Using Data grid Control

Data Grid control is the not the default item in the Visual Basic control toolbox, you have to add it from the VB6 components.

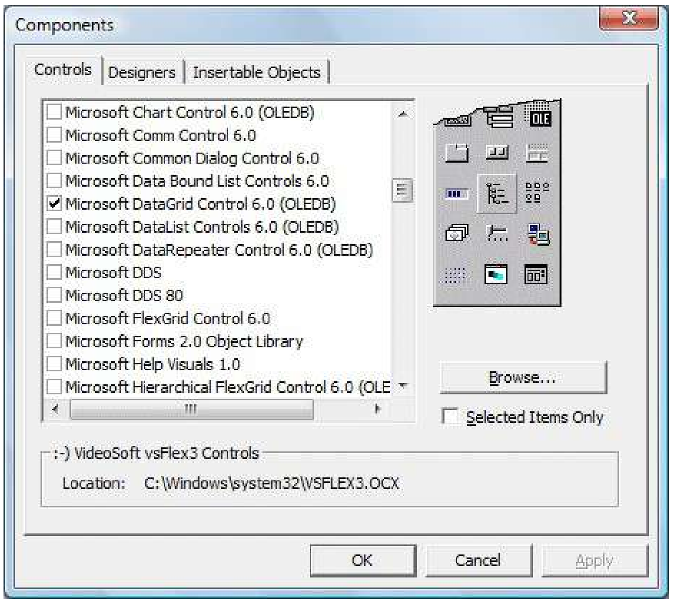
To add the Data Grid control,

1. Click on the project on the menu bar and select components to access the dialog box that displays all the available VB6 components, as shown in the diagram below. Select **Microsoft Data Grid Control 6.0** by clicking the checkbox beside this item. Before you exit the dialog box, you also need to select the **Microsoft ADO data control** so that you are able to access the database.

2. Last, click on the OK button to exit the dialog box. Now you should be able to see that the Data Grid control and the ADO data control are added to the toolbox.

3. The next step is to drag the Data Grid control and the ADO data control into the form.

The components dialog box is shown below:


Components

Controls | Designers | Insertable Objects

- ☐ Microsoft Chart Control 6.0 (OLEDB)
- ☐ Microsoft Comm Control 6.0
- ☐ Microsoft Common Dialog Control 6.0
- ☐ Microsoft Data Bound List Controls 6.0
- ☑ Microsoft DataGrid Control 6.0 (OLEDB)
- ☐ Microsoft DataList Controls 6.0 (OLEDB)
- ☐ Microsoft DataRepeater Control 6.0 (OLEDB)
- ☐ Microsoft DDS
- ☐ Microsoft DDS 80
- ☐ Microsoft FlexGrid Control 6.0
- ☐ Microsoft Forms 2.0 Object Library
- ☐ Microsoft Help Visuals 1.0
- ☐ Microsoft Hierarchical FlexGrid Control 6.0 (OLE

Browse...

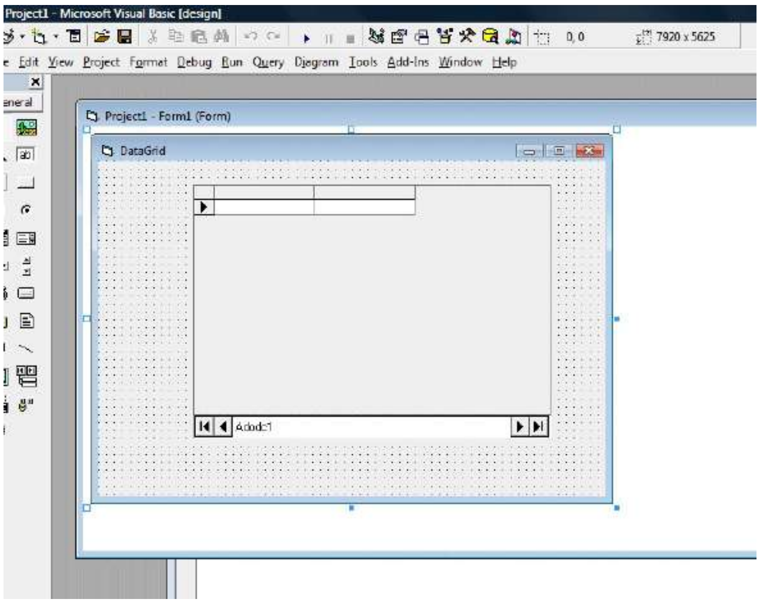
☐ Selected Items Only

;-) VideoSoft vsFlex3 Controls

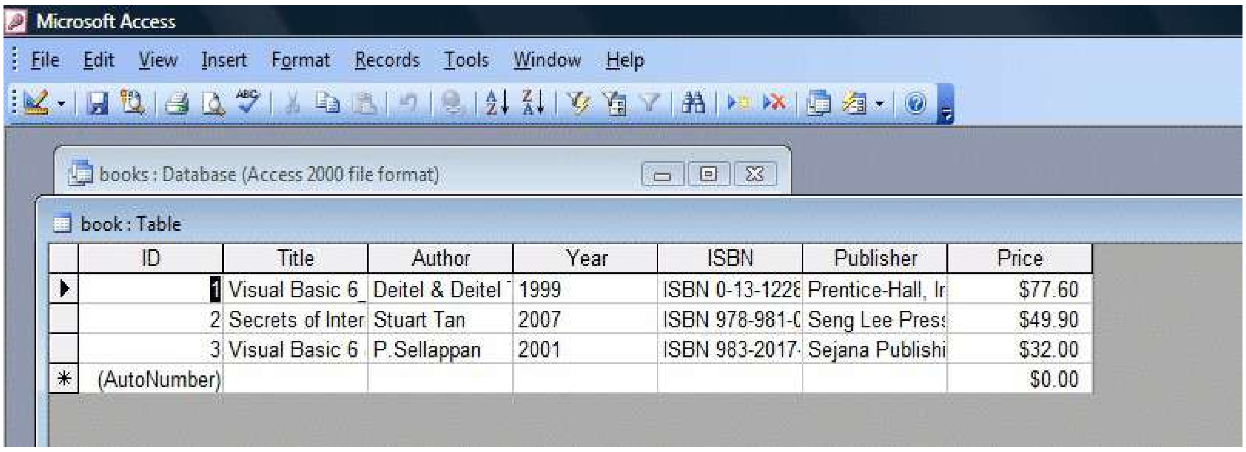
Location: C:\Windows\system32\VSFLEX3.OCX

OK | Cancel | Apply

Design Interface

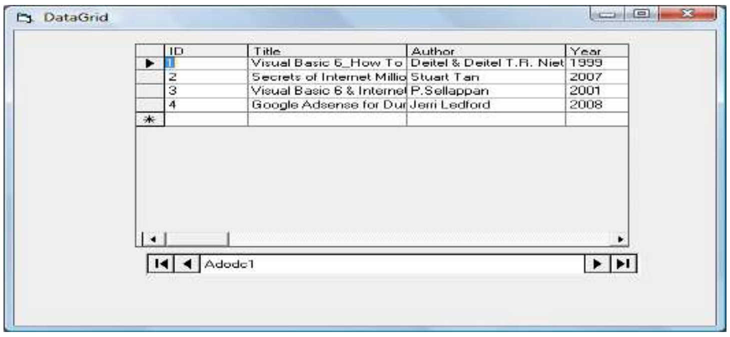

Project1 - Microsoft Visual Basic [design]
0, 0
7920 x 5625
Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help
Project1 - Form1 (Form)
DataGrid
Adodc1

Before proceeding. A database file using Microsoft Access. Here must be created a database file to store the information of books and named the table as **book**. Having created the table, enter a few records, as shown in Figure below:

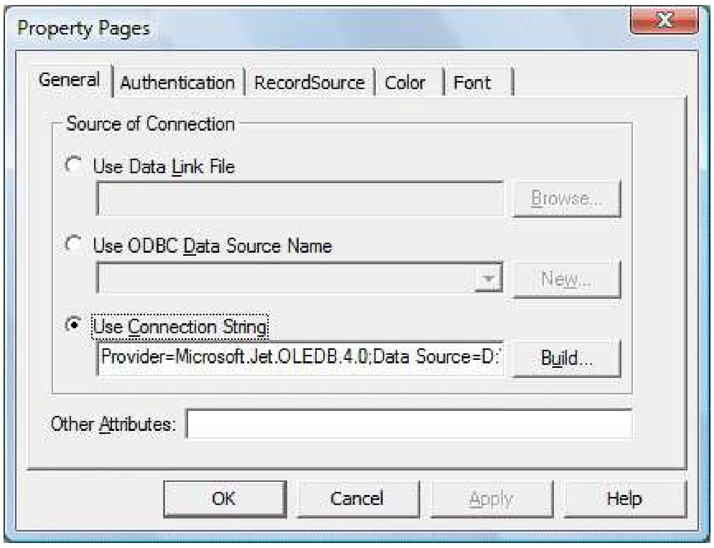

Microsoft Access
File Edit View Insert Format Records Tools Window Help
books : Database (Access 2000 file format)
book : Table

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

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



	ID	Title	Author	Year
▶	1	Visual Basic 6_How To	Deitel & Deitel T.R. Niet	1999
	2	Secrets of Internet Millio	Stuart Tan	2007
	3	Visual Basic 6 & Internet	P.Sellappan	2001
	4	Google Adsense for Dur	Jerri Ledford	2008
*				



( ) Use Data Link File
( ) Use ODBC Data Source Name
[✓ Use Connection String] Provider=Microsoft.Jet.OLEDB.4.0;Data Source=D:
Other Attributes:

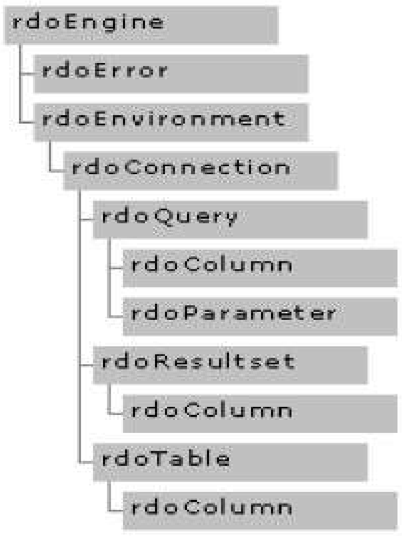
Next click on the Build button and the Data Link Properties dialog box will appear. In this dialog box, select the database file **books.mdb**. Press test connection to see whether the connection is successful.

Remote Data Objects

Remote Data Objects (RDO) is a thin layer over the ODBC application-programming interface (API). RDO depends on the ODBC driver and the database engine for much of its functionality. Data access using RDO is intended for only ODBC relational databases.

RDO Hierarchy

The following Figure shows the RDO hierarchy



The RDO objects and their important method(s) are briefed below: rdo Engine object. The base RDO object. This object is created automatically

Method	**Description**
Rdo Create Environment	Creates a new rdo Environment object.
	Open Connection Opens a connection to an ODBC data source and returns a reference to the rdo Connection object that represents a specific database.
Open Result Set	Creates a new rdo Result set object.
	Execute Runs an action query or executes an SQL statement that does not return rows.

Rdo Environment:

Object Defines a logical set of connections and transaction scope for a particular user name. This object contains both open and allocated (but unopened) connections. This object is created automatically.

Rdo Connection:

Object Represents either an open connection to a remote data source, or an allocated, but as yet unopened, connection.

Rdo Query:

Object an SQL query statement with zero or more parameters.

Rdo Column:

Object represents a column of data, including data type and common properties.

Rdo Parameter:

Object represents a parameter associated with an rdo Query object. Query parameters can be input, output, or both.

Rdo Result set:
Object A set of rows returned from a query.

An RDO-based application uses the following operations to access a data source.

1. **Set the environment handle** Identifies the memory location for global data and status information for the defined connections.
2. **Open the Connection** Specifies the connection string with information such as data source name, user identification, password, default database, network name of the data source server, and name of the data source driver.
3. **Open the result set** This runs a query and creates a result set.
4. **Use the result set** The result set is now available to your application. Depending on the cursor type, you can browse and change the item data at either the server or client side.
5. **Close the connection** Drops the connection to the data source.
6. **Free the environment handle** Drops the global data and frees all associated memory.

Establishing a connection

To establish a connection to a database in oracle, Open Connection is method of an existing rdo Environment object. An rdo Environment object defines a logical set of connections and transaction scope for a particular user name. Rdo Environments(0) is a member of the rdo Environments collection and is created automatically when you include RDO in your program.

Syntax

Set connection = environment. **Open Connection (**ds Name [,prompt [,read only [,connect [,options]]]]**).**
The ds Name argument, as this was the Data Source Name given during the ODBC setup.

The prompt argument controls whether or not the user is prompted for their User ID and password via the "ODBC Data Sources" dialog box. Provided that enough information is given within the other arguments of the Open Connection method, the constant **rd Driver No-Prompt** will suppress the prompt.

The third argument, read only is a Boolean indicating whether or not the connection should be open for read/write access. In program omits this argument, thus defaulting to **False**, meaning the connection is open for read/write access.

The connect argument is a string expression used to pass arguments to the ODBC driver manager for opening the database. In the case of the sample application, we are passing a user id of "admin" and a blank password in the connection string, which are the defaults for an unsecured MS Access database.

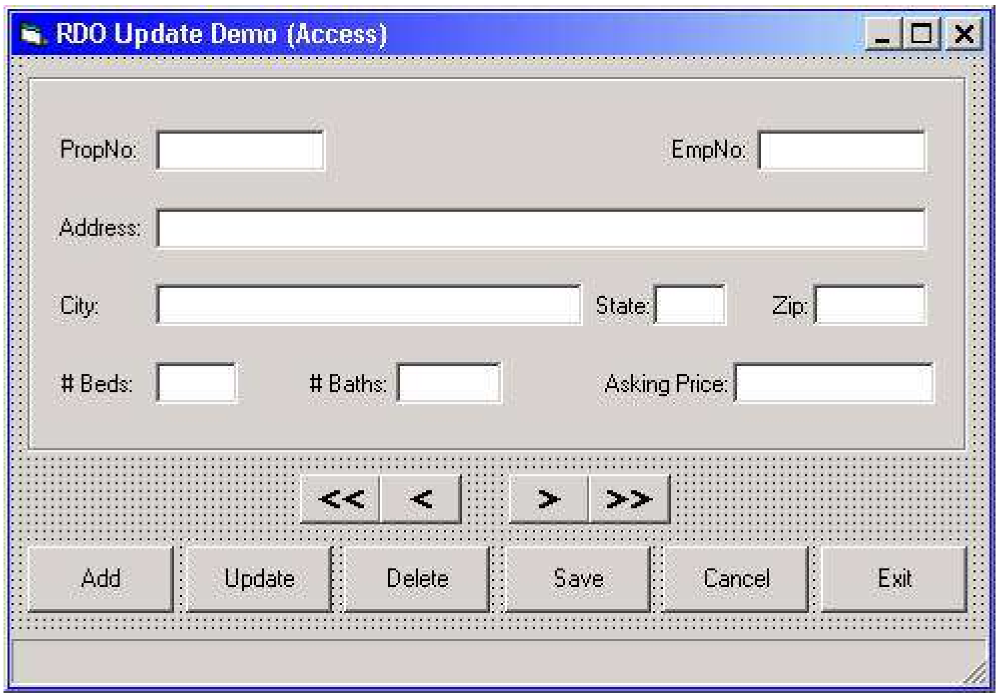
Excuting SQL Statement
After establishing a connection, the user can excute quaries on the database.
The **Open Result set** method causes an rdo Result set object to be created. The syntax is:
Set variable = connection. **Open Result set**(name [,type [,lock type [,option]]])

The name argument is a string that specifies the source of the rows for the new **rdo Result set**. This argument can specify the name of an **rdo Table** object, the name of an **rdo Query**, or an SQL statement that might return rows. In the case of the sample application, it is the SQL statement "select * from property".

The type argument specifies which type of cursor to create (using **rd Open Key set** allows the use of the "Move" methods (Move First, Move Next, etc.), so that option was coded here).

The lock type argument specifies the concurrency option that controls how other users are granted or refused access to the data being updated. A value of **rd Concur Row Ver** specifies optimistic concurrency based on row ID.

Using RDO to insert, Update and Delete Records



Records can be inserted, existing records can be modified and unwanted records deleted using RDO objects.

Create the form shown above. The settings for the various controls are given below.

The navigation buttons have the following properties:

Name	**Caption**
Cmd Move First	<<
Cmd Move Previous	<
Cmd Move Next	>
Cmd Move Last	>>

Example Code:-
```
DIM ENV AS RDOENVIRONMENT
DIM CN AS RDOCONNECTION
DIM RS AS RDORESULTSET
Enter the following code in the form_load event procedure
Set env = rdoenvironments(0)
env.Cursordriver = rdUseOdbc
Set cn = env.openconnection ( dsname:="",prompt:=rdodriverprompt,
Readonly:=false,connect:="")
Dim s as string
s = "select * from emp1"
set rs = cn.OpenResultset (Name:=s, Type:=rdOpenDynamic,_LockType:=
rdConcrRowVer)
If rs.EOF <> True Then
Call Displayrecord
Else
MsgBox " No record found"
End If
End sub
```

When the user click the **Add** commandbutton, it clears the form and allows the entry of new information by the user.

```
Private Sub Add_Click()
Call.ClearRecord
rs.AddNew
End sub
```

The **Update** routine is used to save the changes that are made to the currentrecord.

```
Private Sub Update_Click ()
rs.edit
rs("dept_no") = txtdeptno.Text
Remote Data Objects
Centre for Information Technology and Engineering, Manonmaniam Sundaranar University 221
rs("dept_name") = txtdeptname.Text
rs.update
end sub
```

When the user click the **Delete** command Button, the current record must bedeleted from the table.

```
Private sub Del_click()
rs.delete
Call Displayrecord
End sub
Private sub Displayrecord()
Txtdeptno.Text = rs( dept_no)
Txtdeptname.text = rs(dept_name)
End sub
Private sub ClearRecord()
Txtdeptno.Text = ""
Txtdeptname.text = ""
End sub
```

Data Environment

The Data Environment designer provides an interactive, design-time environment for creating ADO objects. These can be used as a data source for data-aware objects on a form or report.

The following actions can be performed using Data Environment Designer

1. Deine the database connection
2. Create commands for accessing the data
3. Build complex queries
4. Define aggregate functions
5. Specify sort order of the result set of a query

Accessing Data using Data Environment

Data Report

It allows using drag and dropping to quickly create reports from any record set, including hierarchical record set.

OBJECT LINKING AND EMBEDDING (OLE)

Introduction

OLE (Object Linking and Embedding) is a means to interchange data between applications. For developers, it brought OLE Control Extension (OCX), a way to develop and use custom user interface elements. On a technical level, an OLE object is any object that implements the IOle Object interface, possibly along with a wide range of other interfaces, depending on the object's needs. OLE allows an editing application to export part of a document to another editing application and then import it with additional content. OLE is also used for transferring data between different applications using Drag and Drop operations OLE 1.0, released in 1990, was an evolution of the original Dynamic Data Exchange (DDE) concept that Microsoft developed for earlier versions of Windows. While DDE was limited to transferring limited amounts of data between two running applications, OLE was capable of maintaining active links between two documents or even embedding one type of document within another.

Objects and classes

An object is a combination of code and data that can be treated as a unit. An object can be a piece of an application, like a control or a form. An entire application can also be an object.

When you create an application in Visual Basic, you constantly work with objects. You can use objects provided by Visual Basic, such as controls, forms, and data access objects. You can also use objects from other applications within your Visual Basic application. You can even create your own objects and define additional properties and methods for them. Objects act like prefabricated building blocks for programs - they let you write a piece of code once and reuse it over and over.

Each object in Visual Basic is defined by a class. A class describes the variables, properties, procedures, and events of an object. Objects are instances of classes; you can create as many objects you need once you have defined a class.

To understand the relationship between an object and its class, think of cookie cutters and cookies. The cookie cutter is the class. It defines the characteristics of each cookie, for example size and shape. The class is used to create objects. The objects are the cookies.

Using the OLE Container Control:

The OLE container control is used to create a document-cantered application. In such an application, the user combines data from different applications to create a single document. This type of application may be a word processor that allows the user to enter text and then embed a spreadsheet or chart.

One of the primary uses of the OLE Container control was to embed Word documents or Excel spreadsheets in a form.

The OLE container control allows you to add objects from other applications to Visual Basic applications. By using this control, the following actions can be performed.

1. Create a linked object in an application.
2. Bind the OLE container control to a database.

3. Perform an action if the user moves, sizes, or updates the object in the OLE container control.
4. Create objects from data that was copied onto the Clipboard.
5. Display objects as icons.
6. Provide backward compatibility with an application that includes many OLE container controls (called OLE client controls in previous versions of Visual Basic).

To create an object at Design Time

Embed objects at design time. Should be known the object to be embedded and to limit the users of the application to a specific object or type of object (a Word document, for example).

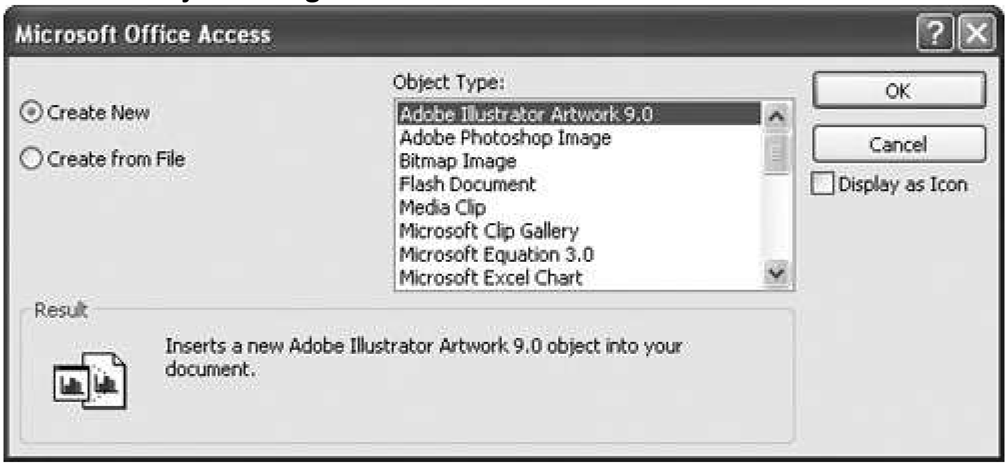
It's also possible to link objects with the following techniques, but at design time, rather than link the document should be embedded.

You can embed objects in two ways at design time:

1. By pasting existing object (or dragging-and-dropping the object on the control).
2. By selecting the object through the Insert Object dialog box. This is the control to use when you want to link or embed an object into your Visual Basic application.

1. From the Windows Start menu, choose Programs, Microsoft Visual Studio 6.0, and then Microsoft Visual Basic 6.0.
2. 2.Click the OLE tool form toolbox
3. Double clicking the OLE control on your form will activate the object's application. Visual Basic also allows you to activate the object from your code. A Insert dialog box is open. There are two option create new or create from file. you select it as you want .
4. First we create new Object.
5. Select Create New and Microsoft office Excel Chart from the Object Type list.

The Insert Object dialog box



[● Create New] ○ Create from File

Object Type:
Adobe Illustrator Artwork 9.0
Adobe Photoshop Image
Bitmap Image
Flash Document
Media Clip
Microsoft Clip Gallery
Microsoft Equation 3.0
Microsoft Excel Chart

Inserts a new Adobe Illustrator Artwork 9.0 object into your document.

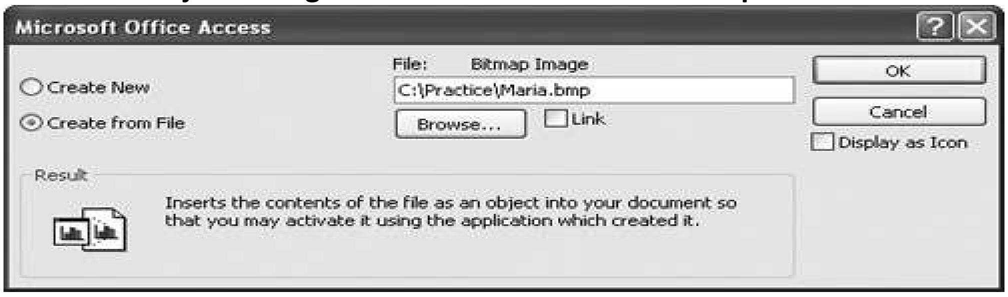
6. Creating a Excel chart will allow you to access the Excel application to define the chart.
7. Select the Excel chart.
8. Run your application. Double click on the OLE control and use it in your project as you want.

Second is linking to an existing object for this:

1. Add a OLE to the form.

2. Select Create from File and use the Browse button to find the Word document
3. Select Link and click on OK.

The Insert Object dialog box with the Create from File option selected.


Microsoft Office Access

Create New
[✓ Create from File]

File: Bitmap Image
C:\Practice\Maria.bmp
Browse...
Link

OK
Cancel
Display as Icon

Result
Inserts the contents of the file as an object into your document so that you may activate it using the application which created it.

4. This Word document page has been sized so that it will fit onto a form. Change the Size Mode property of the OLE control to Stretch.
5. Run your application. Double click the OLE control to see that Word is activated with the selected document opened.
6. Close Word and stop your application.

Using Paste Special Dialog box

This method displays the Paste Special dialog box. The user's selections are reported to the application via the OLE Container control's properties.

To find out the intrinsic constants you can use with the various methods and properties of the OLE Container control, use the Object Browser.

1. Choose View > Object Browser,
2. Select the Visual Basic object library, and then choose the Constants object to display the names of the constants under Methods/Properties.

Creating an Embedded Object at Run Time

We can create a linked object at run time using the Source Doc Property and create Link Method.

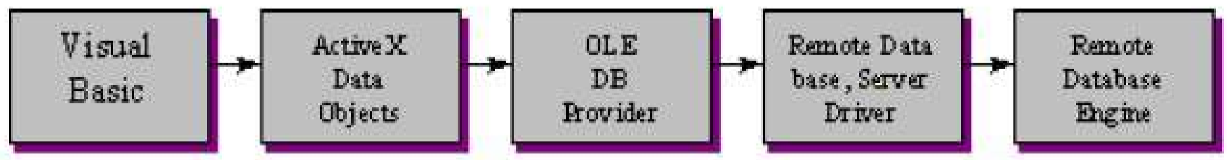
To manipulate embedded or linked objects, it provides the following methods.

Constant	**Description**
acOLECreateEmbed (0)	Creates an embedded object. To use this setting, you must first set the control's **OLETypeAllowed** property to **acOLEEmbedded** or **acOLEEither**. Set the **Class** property to the type of OLE object that you want to create. You can use the **SourceDoc** property to use an existing file as a template.
acOLECreateLink (1)	Creates a linked OLE object from the contents of a file. To use this setting, you must first set the control's **OLETypeAllowed** and **SourceDoc** properties. Set the **OLETypeAllowed** property to **acOLELinked** or **acOLEEither**. The **SourceDoc** property specifies the file used to create the OLE object.

Constant	Description
	You can also set the control's **SourceItem** property (for example, to specify a row-and-column range if the object you are creating is a Microsoft Excel worksheet). When you create an OLE object by using this setting, the control displays a metafile graphic image of the file specified by the control's **SourceDoc** property.

ACTIVE X DATA OBJECTS (ADO)

ACTIVE X DATA OBJECTS (ADO)



An ADO connection to a Remote Database

Goal of Universal Data Access

Universal Data Access provides high-performance access to a variety of information sources, including relational and non-relational sources, and an easy-to-use programming interface that is tool - and language - independent. These technologies enable corporations to integrate diverse data sources, create easy-to-maintain solutions, and use their choice of the best tools, applications, and platform services.

These are three major components that are interacting within the Universal Data platform:

Data consumers are applications that need access to a broad range of data. These include development tools, languages, and personal productivity tools. An application becomes ODBC-enabled by using the ODBC API to talk to data. Similarly, an application becomes OLE DB-enabled by using the OLE DB API to talk to data.

Data providers make their data available for consuming. They may do this by natively supporting OLE DB or they may rely on additional OLE DB data providers. OLE DB Providers for Microsoft SQL Server, Oracle, Microsoft Jet and more on the way.

Service components provide additional functionality such as query processing or cursor engines. A query processor allows SQL queries to be constructed and run against the data source. A cursor engine provides scrolling capabilities for data sources that don't support scrolling.

OLE DB

OLE DB is Microsoft's strategic low-level interface to all kinds of data throughout the enterprise.

OLE DB is an open specification designed to build on the success of ODBC by providing an open standard for accessing all kinds of data.

Whereas ODBC was created to access relational databases, OLE DB is designed for relational and non-relational information sources, including mainframe ISAM/VSAM and hierarchical databases; email and file system stores; text, graphical, and geographical data; custom business objects; and more.

OLE DB components consist of data providers, which expose their data; data consumers, which use data; and service components, which process and transport data.

ActiveX Data Objects

ADO is Microsoft’s strategic, high-level interface to all kinds of data.

ADO provides consistent, high-performance access to data, whether you’re creating a front-end database client or middle-tier business object using an application, tool, language, or even an Internet browser.

ADO is the single data interface you need to know for 1- to n-tier client/server and Web-based data-driven solution development.

ADO is designed as an easy-to-use application level interface to Microsoft’s newest and most powerful data access paradigm, OLE DB.

Remote Data Service

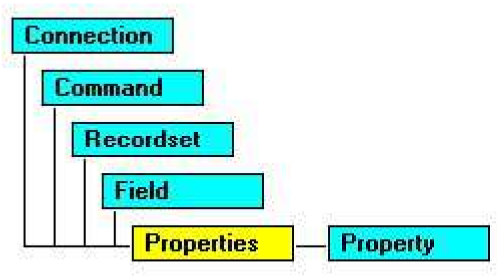
RDS is a feature of ADO and makes full-featured, data-centric Web applications a reality by combining data manipulation of retrieved data, efficient client-side caching, and support for data-aware ActiveX controls with an elegant and powerful programming model.

RDS goes beyond the current generation of Web data access tools by allowing clients to update the data they see. Using drop-in ActiveX data controls, such as grids, lists, and combo boxes, developers can deploy sophisticated user interfaces that allow end users to view and change data with a minimum of programming. End users are no longer restricted to staring at a static HTML results table.

With RDS, they can now alter, add, and delete data they have queried and retrieved. In addition, all changes are buffered locally, and can be submitted to the server for inspection, processing, and storage in the database.

ADO OBJECT MODEL

1. ADO provides better overall performance than earlier object models.
2. ADO is more resource-efficient than earlier object models.
3. ADO provides more universal data access, due to the universal nature of its underlying standard, OLE DB.



ADO Object Model

Connection

The connection object is ADO's connection to a data store via OLE DB. The connection object stores information about the session and provides methods of connecting to the data store. As some data stores have different methods of establishing a connection, some methods may not be supported in the connection object for particular OLE DB provider. A connection object connects to the data store using its 'Open' method with a connection string which specifies the connection as a list of key value pairs (for example: "Provider='SQLOLEDB';Data

Source='TheSqlServer'; Initial Catalog='Northwind';Integrated Security='SSPI';"). The start of this connection string must identify the type of data store connection that the connection object requires:

1. an OLE DB provider (for example SQLOLEDB), using the syntax "provider=";
2. a file name, using the syntax "file name=";
3. a remote provider and server (see RDS), using the syntax "Remote provider=" and "Remote server="; or
4. an absolute URL, using the syntax "URL="

Command

After the connection object establishes a session to the data source, instructions are sent to the data provider via the command object. The command object can send SQL queries directly to the provider through the use of the CommandText property, send a parameterised query or stored procedure through the use of a Parameter object or Parameters collection or run a query and return the results to a dataset object via the Execute method.

Record set

A record set is a group of records, and can either come from a base table or as the result of a query to the table. The Record Set object contains a Fields collection and a Properties collection. The Fields collection is a set of Field objects, which are the corresponding columns in the table.

Immediate

The record set is locked using the ad Lock Optimistic or ad Lock Pessimistic lock. The data are updated at the data source after the record is changed and the Update method is called.

Parameter

A parameter is a means of altering the behaviour of a common piece of functionality, for instance a stored procedure might have different parameters passed to it depending on what needs to be done; these are called parameterised commands.

Field

Each Record object contains many fields, and a Record Set object has a corresponding Field object also. The Record Set object's Field object corresponds to a column in the database table that it references.

Property

This object is specific to the OLE DB provider and defines an ability that the provider has implemented.
ADO objects have two types of properties: **built-in** and **dynamic**.

Built-in properties are those properties implemented in ADO and immediately available to any new object, using the Some Object. Property syntax.

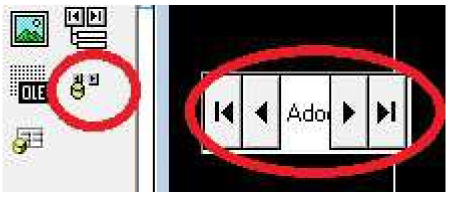
Dynamic properties are defined by the underlying data provider, and appear in the **Properties** collection for the appropriate ADO object.

Error

When an OLE DB provider error occurs during the use of ADO, an Error object will be created in the Errors collection. Other errors do not go into an Error object, however. For instance, any errors that occur when manipulating data in a RecordSet or Field object are stored in a Status property.

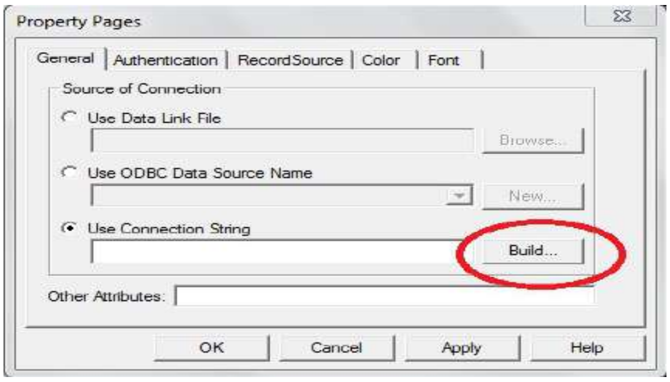
Working with ADO Control

1. Add the ADODB components in the VB Toolbox.
2. Add an **Adodb component**, [Right Click] on the **Control Object** > Click **Properties**.



ADO DB Component

3. Under the **Adodc properties** windows > Click **Build**.

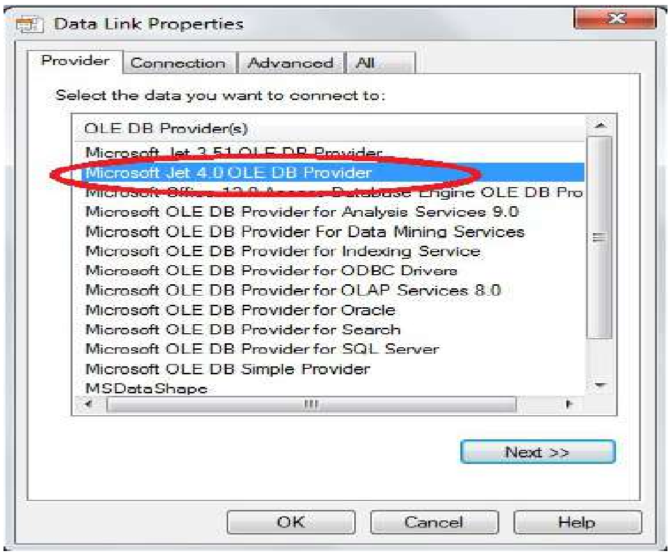


Source of Connection

- Use Data Link File
- Use ODBC Data Source Name
- Use Connection String [✓]

Other Attributes:

ADODC Properties



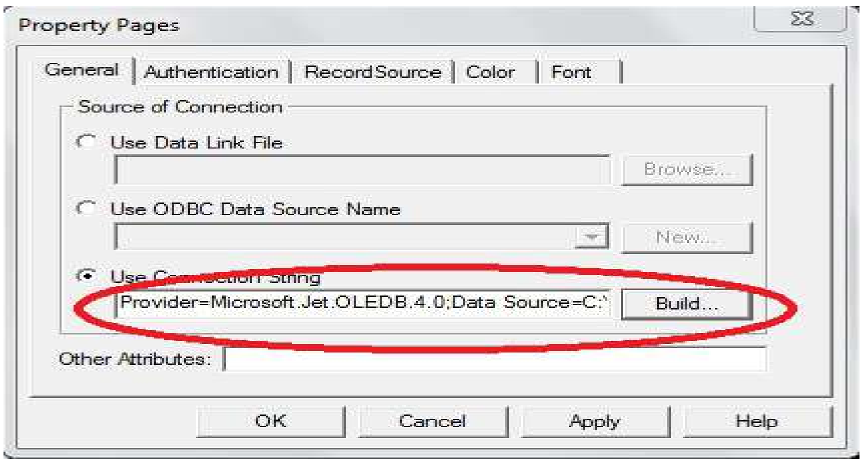
Select the data you want to connect to:

OLE DB Provider(s)
Microsoft Jet 3.51 OLE DB Provider
Microsoft Jet 4.0 OLE DB Provider
Microsoft Office 12.0 Access Database Engine OLE DB Pro
Microsoft OLE DB Provider for Analysis Services 9.0
Microsoft OLE DB Provider For Data Mining Services
Microsoft OLE DB Provider for Indexing Service
Microsoft OLE DB Provider for ODBC Drivers
Microsoft OLE DB Provider for OLAP Services 8.0
Microsoft OLE DB Provider for Oracle
Microsoft OLE DB Provider for Search
Microsoft OLE DB Provider for SQL Server
Microsoft OLE DB Simple Provider
MSDataShape

Select the Provider

1. Under the **Connection** tab > Select the **Database** you created previously.

2. Now Click **"Test Connection"** to check if the database connection is successful.
3. Click on the "**Provider**" tab again and we now have our connection string.


Property Pages

General | Authentication | RecordSource | Color | Font

Source of Connection

Use Data Link File

Use ODBC Data Source Name

[✓ Use Connection String]

Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\

Other Attributes:

Connection String

This new Connection String will be used in Module. Bas file for accessing the database at run-time.

1. In the Property page click the **Authentication Tab** provide the user name along with the password
2. Next double click the **record Source** property to specify the name of the table.
3. Place a **Textbox** and set its data source property to **Adodc1**. The Data field Property set to **Emp_id.**
4. Similarly include all other objects for creating Employee details Form.

Using the ADO connection and Record set objects
Now let us use ADO to connect it to the Employee details tables.

Connecting to the Database

The first step in every database program is connecting the database.

```
Dim con As New adodb.Connection
Dimrs As New adodb.Recordset
Dim constr As String
constr = "Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Users\Peter\Documents\emp.mdb;Persist Security Info=False"
con.Openconstr
MsgBox ("connected")
```

Closing the Database Connection

The database can be closed using Close method.

```
Con.close
Set con = nothing
```

Where con is the connection variable.

Retrieving a Record set

ADO provides features similar to DAO but uses the command object to do the actual work.The code as follows,

```
Dim rs as ADODB.Recordset
Set rs = new ADODB.Recordset
Rs.cursortype =adopenkeyset
Rs.locktype= adlockoptimistic
Rs.open “empdetails”, con,adcmdtable
```

Using Action Quaries

Action quaries more commonly known as stored procedures,perform an action on the database without returning a redcordset.

These are used to delete, insert & update records.

Adding Records

The **Add new()** method is used to add a new record to a table. The constants used to open the record set are fairly important.

Modifying Records

To Modify a record satisfying a particular condition, the record is first referred. And then use **Edit()** method to change the records.

Files And File System Controls

Introduction

VB provides three native toolbox controls for working with the file system: the **Drive List Box**, **Dir List Box**, and **File List Box**. You can use these controls independently, or in concert with one another to navigate the file system.

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

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

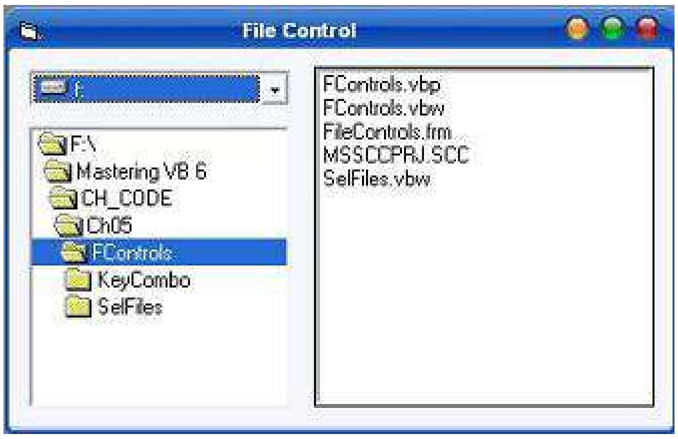
```
Dir1.Path = Drive1.Drive
```

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

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

The **FileListBox** control lists files in the directory specified by its Path property.All the files in the current directory can be displayed , or to show only certain types of files by using the puttern properly.

Following figure shows three file controls are used in the design of Forms.



The three File controls are not tied to one another. All three of them on a Form can be placed the names of all the folders under the current folder,should the limited.

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.

When the user selects a new drive in the DriveListBox control, it fires a Change event and returns the drive letter (and volume label) in its Drive property.

```
Private Sub Drive1_Change()
' The Drive property also returns the volume label, so trim it.
Dir1.Path = Left$(Drive1.Drive, 1) & ":\"
End Sub
```

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

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

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

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

Accessing Files

1. A File consists of series of related bytes located on a Disk.
2. Depending on the kind of data the file contains we can use the appropriate file access type.

There are basically three ways of accessing files:

A. Sequential file:

This is a file where all the information is written in order from the beginning to the end. Before to access a given record ,the system should to read all the records. It is in fact like listening to a tape. In fact, in the old days, magnetic tape was the most commonly used medium to store data and all files were organized this way. Now, it is still useful when there is a small amount of data to store, a file of application settings. It can even be of use when there is a large amount of data to be stored, provided it all has to be processed at one time, eg: a file of invoices to produce a statement at month-end.

B. Random file:

A file where all records are accessible individually. It is like a CD where you can jump to any track. This is useful when there is a large quantity of data to store and it has to be available quickly: you have to know if a part is in stock for a customer who is on the phone; the program doesn't have time to search through 10,000 records individually to locate the correct one. This method of storage became popular when hard-disk drives were developed.

C. Binary file:

This is a special, compacted form of the random file. Data is stored at the byte level and you can read and write operation can be performed at individual bytes to the file. This makes the file access very fast and efficient.

Opening and closing files

These two commands that are common to both Sequential and Random files.

The first command to include in a program that needs to work with files is the **Open** command.

Open assigns the file to a numbered **file handle**, also called a **channel**, or sometimes a **buffer**. The format of the command is:

Open "Filename" [For Mode] [AccessRestriction] [LockType] As #FileNumber
For example:
Open "MyFile.txt" For Random Read Lock Read As #1

1. **MyFile.txt** is the name of the file in the disk directory.
2. **For Random** means that access to the records can be random; if access is not specified, For random is the default value.
3. **Read** restricts access to Read-only - the user cannot write or change the records.
4. **Lock Read** means that only the person reading the record can have access to it at any given time; it is not shared among users.
5. **As #1** means the file is assigned file handle #1; for all processing in the program, it will always be referred to as #1, not its Filename.

Once processing is finished, you need to **Close** all the files that have been opened. The format for the **Close statement** is:
Close #FileNumber1 [, #FileNumber2] ...
You can close any number of files with one Close statement. Eg:
Close #1, #2, #3
The following statement closes all open files:
Close

Sequential Files

There are two commands that allow you to write data to a sequential file: **Print** and **Write**. They work in almost the same way but, the Print command does not separate the fields in the file in quite the same way which makes the data harder to read afterwards. Sequential accesses files are opened in one of the three ways – output, append or input.

Example:

```
Private sun cmd1_click ()
File num= Free File
Open “c:\OUTPUT.TXT” For Output as filenum
Print #filenum, text1.text
Close filenum
End sub
```

The print statement is used to write text into the file.
Two parameter are passed to this statement.
The first parameter is the filenumber and the second is the string to be written into the file.

Binary Access File

Binary access files are accessed byte by byte.
Once a file is opened for binary access we can read from and write to any byte location in the file.
Before accesing a file in binary mode,the file should be opened first for binary access.

Example:

```
Private sun cmd1_click()
```

```
Str = “This is a Binary Access File”
filenum = FreeFile
Open “c:\BINARY.TXT” For Binary as filenum
put #filenum, 100,str
Close filenum
End sub
```

The application is executed and the click event of the cmd creates the file BINARY.TXT.