Introduction to Visual Basic 6

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

Visual Basic 6.0 is a very powerful programming language. It enables GUI application development, provides access to databases and enables the creation of ActiveX controls.

In addition, Visual Basic 6 is Event-driven because we need to write code in order to perform some tasks in response to certain events. The events usually comprises but not limited to the user's inputs. Some of the events are load, click, double click, drag and drop, pressing the keys and more. We will learn more about events in later lessons. Therefore, a VB6 Program is made up of many subprograms, each has its own program code, and each can be executed independently and at the same time each can be linked together in one way or another.

Start a New Project

Run the Visual Basic software from the list of programs or a desktop shortcut icon. A appear.A window as same as the following picture will appear.

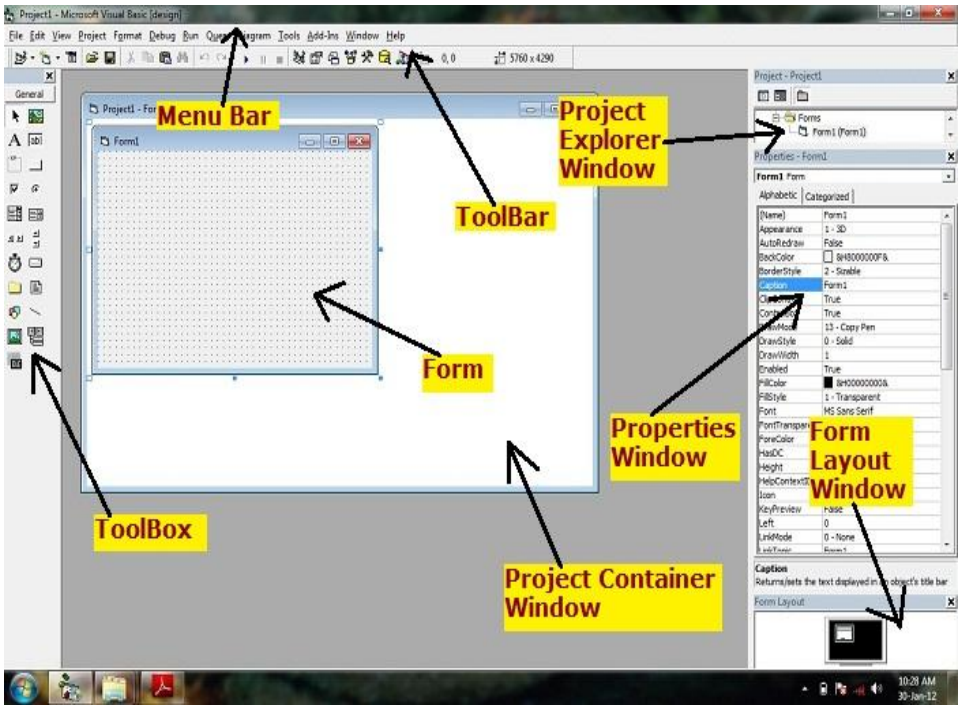
Click "Standard EXE". Start a Standard .exe type of project. In the beginner level, you will only learn about this type of project.

Other project types on this window are for the advanced learners. You can do a lot of things implying VB6 is giving you enough power in your hand through these different project types.

The Integrated Development Environment

After this, the main workspace appears where you will develop your application with the tools in IDE (Integrated Development Environment).

It is very important to know the names of all the elements of this development environment. The tools available here makes it very easy for you to develop an application. The VB6 IDE provides you many tools in one place. You can add a control on the form of your choice, set a property of an object from the Properties Window on the right hand side, set the form layout and many more things that you can use alongside your coding. You can even fill the ToolBox with lots of additional controls.



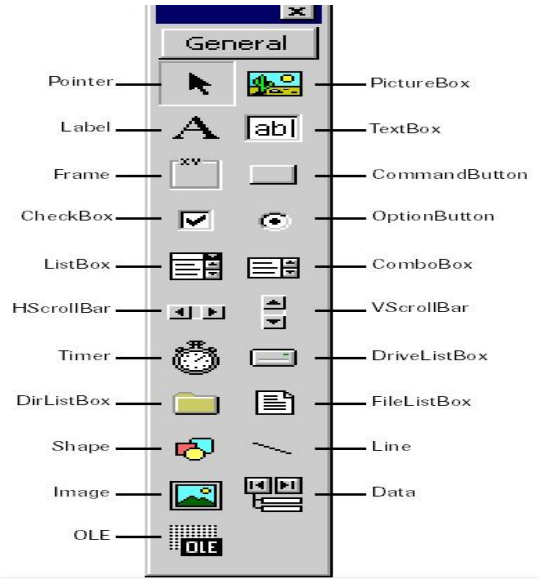
The Menu Bar

The Menu Bar contains all the menus such as File, Edit, View, Tools and so on.

The Tool Bar

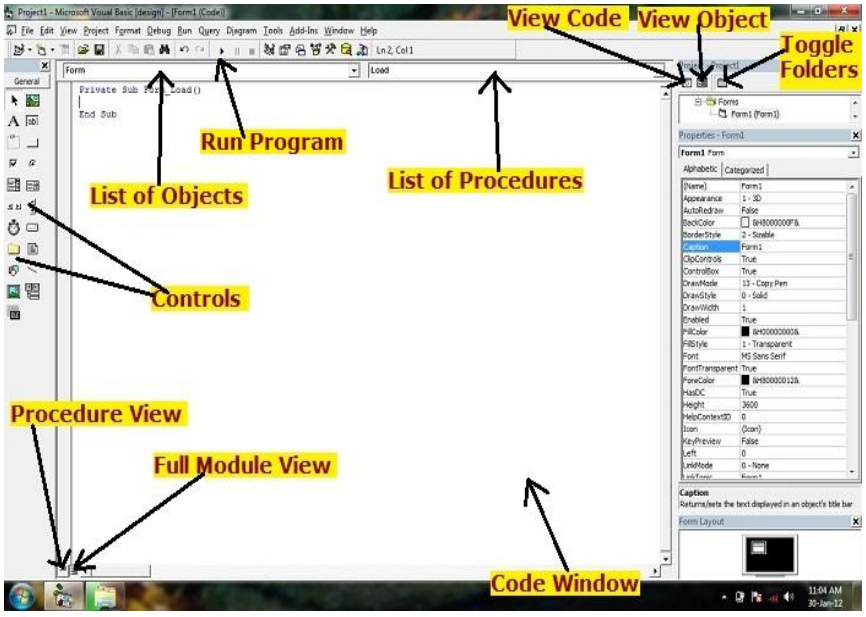
The Tool Bar contains all the tools such as Open, Save, Copy, Cut, Start and so on.

The Tool Box



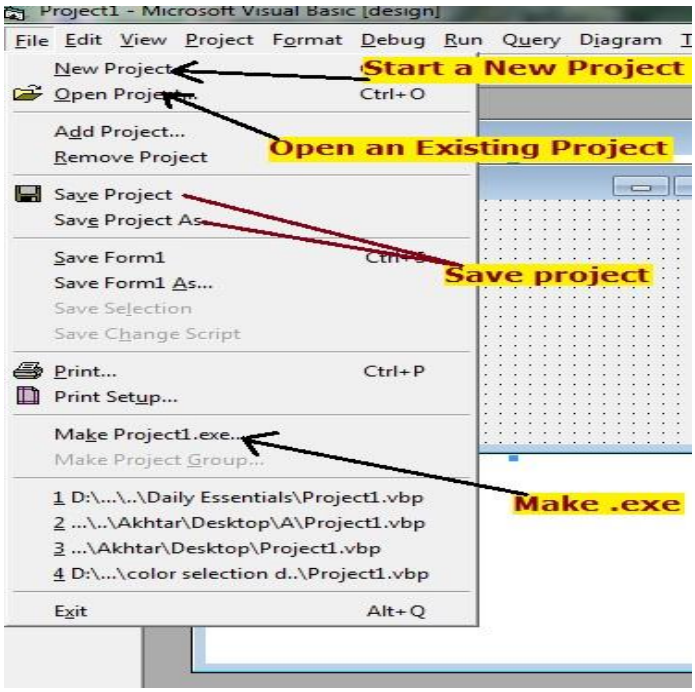
View Code Window

Double-click on form or any control on the form or click "view code" icon in explorer window to view the code window.



Save your project

After developing your application, save your project in order to modify or improve it later, or make an executable file with a few clicks. The project, form and module are saved in .vbp, .frm and .bas extensions respectively.



How to place controls on the form?

1. Select a control from Toolbox, click on form and drag until you have got the shape of the control you want.
2. Alternatively, you may double-click any control to add to form.

After adding a control to the form, you need to set its property and then write code for the control to work how you want.

There are two ways to set property

1. You can set property in Design Time from the Properties Window.
2. Or, you may wish to set property at run time by writing code.

Writing the Code for a Control

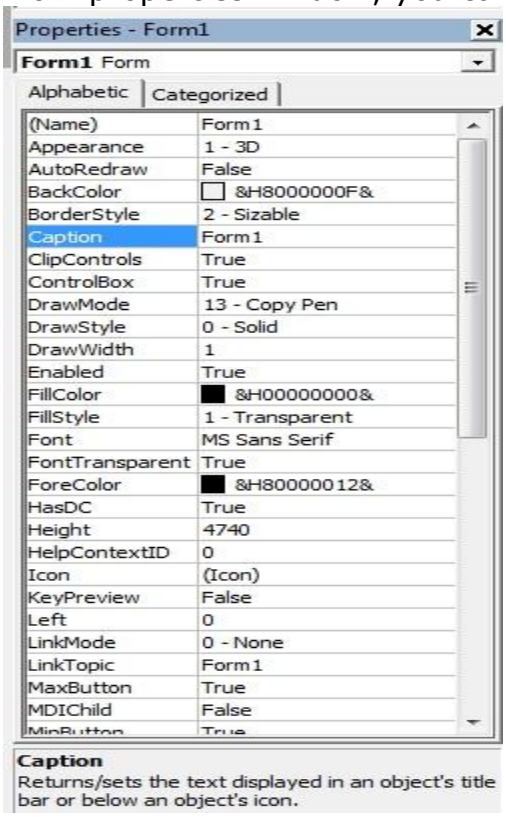
Simply double-click the control (which is on the form) to view the code window and write code to specify how this control will work.

NOTE :

1. Pointer is not a control. Click this icon to select controls already on the form.
2. All controls are object
3. Form is an object, but it is not a control.

The Properties Window

From properties window, you can set properties for controls.



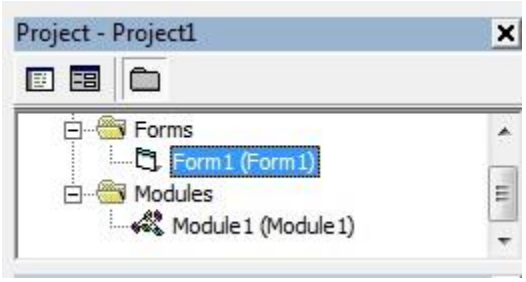
(See in the Picture) 'Caption' is the property of the Form object.
'Form1' is the value of the property.

In the same way, 'Appearance' is the property. '1-3D' is the value.

In the Properties Window, notice the help information about the object. This helps in learning new properties.
(See Picture) Help information for the 'Caption' property is shown.

The Project Explorer Window

Press Ctrl+R if this window is not showing.

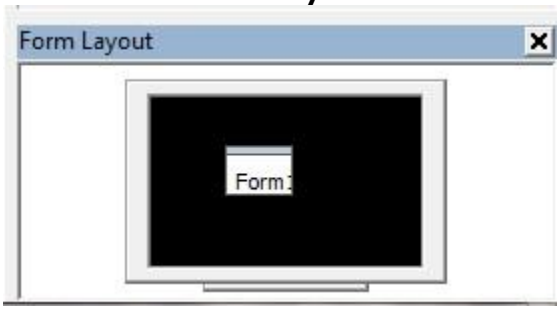


The Project Explorer Window gives you a view of the modules or forms which are contained in your VB application. You can switch from one form to another or from one module to another from the Project Explorer Window. You can view the code window of a particular form or module as well.

The Code Window

You need the Code Window to write code that will specify the behavior of the forms and the objects. Remember that the Form is an object.

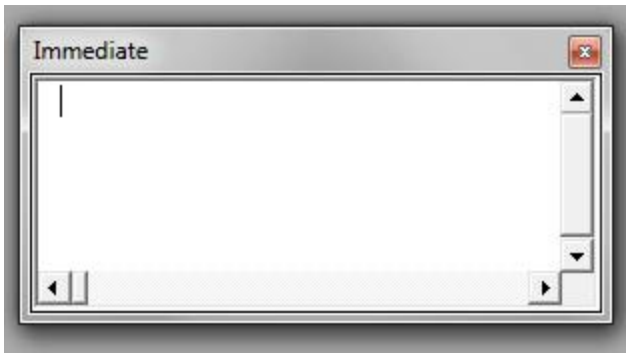
The Form Layout Window



The Form Layout Window shows where on the screen the form will be displayed when the program will be executed. Simply drag on it so that the form appears on the position where you want.

The Immediate Window

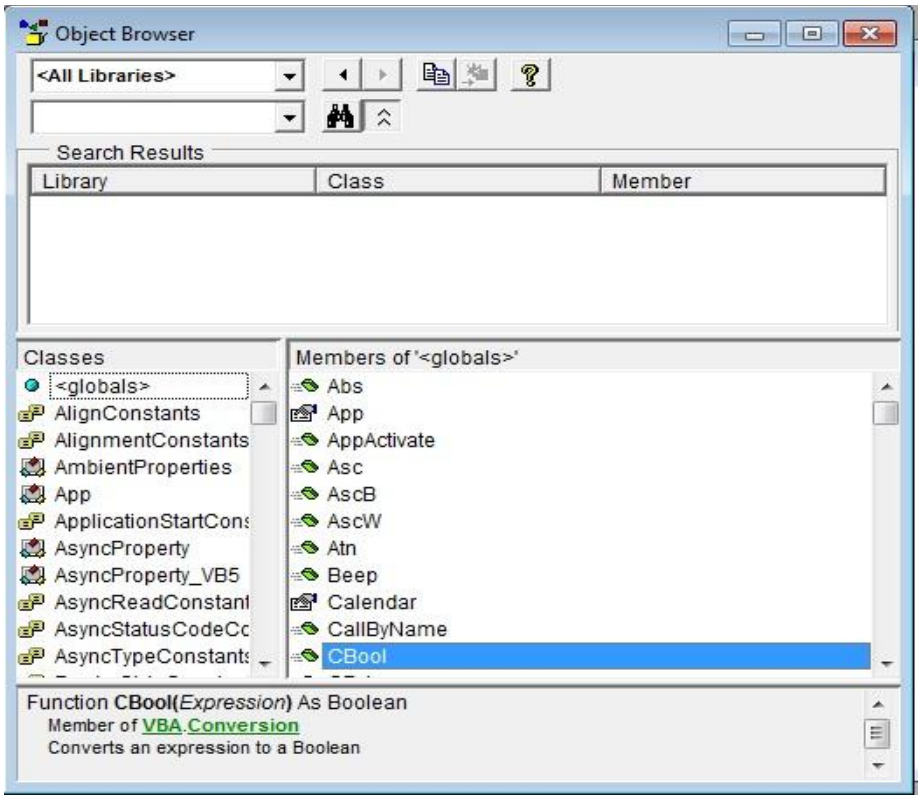
Press Ctrl+G to show the Immediate Window.



The Immediate Window helps in debugging your program by displaying the current values of variables or expressions in a certain line of your code.

The Object Browser

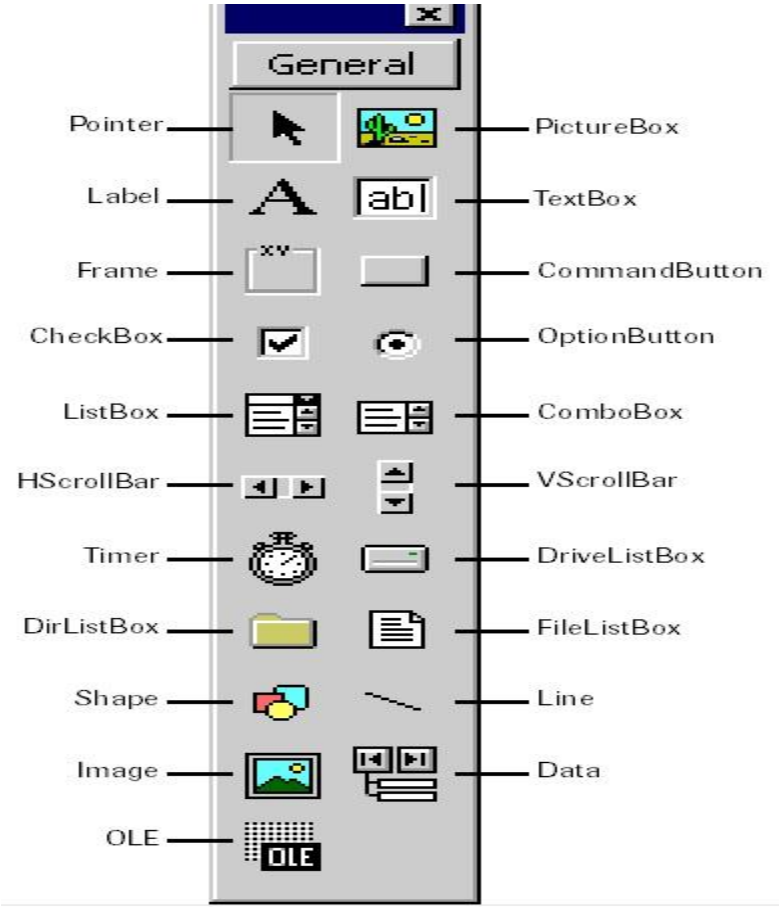
Press F2 to show the Object Browser Window. It is very useful because you can learn about all the methods, functions, properties and events of the objects. If you want to know about any property, method, event, function etc, simply search in the Object Browser.



An Overview of VB6 Controls

Before getting into the actual programming work, you need to have an overview of the VB controls. In this lesson, a very short description of each of the controls has been given.

ToolBox



Label

The label control is used to display text. It is also used to label other controls. The end user cannot edit the label text.

TextBox

The TextBox control contains characters. End-users can edit the characters contained in the TextBox.

CommandButton

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

Option Button

This control enables the end-user to select one among several options. Only one option button among others in a group can be on at the same time. You can name an option using the Caption property.

CheckBox

The CheckBox control is used to make a yes/no or true-false selection. You can check more than one CheckBox at the same time that let you make multiple choices. You can label this control using the Caption property.

VscrollBar & HscrollBar

VscrollBar and HscrollBar controls let you create Vertical scroll bar and Horizontal scroll bar respectively.

Frame

The Frame control is used as a container of other controls. This is also used to group different controls especially in Option Button controls when you wish to select more than one option. The Caption property associated with it is useful to label the frame.

ListBox & ComboBox

The ListBox control contains a number of items. The user can select one or more items from the list.

The comboBox control has the feature of ListBox and TextBox. This control does not support multiple selections.

DriveListBox, DirListBox & FileListBox

These controls are often used together to perform file related tasks like opening or selecting files that are stored in the secondary memory.

Timer

The Timer control is not visible on the form when you run the program. It is used to execute lines of code repeatedly at specific intervals.

The Data Control

The Data control is used for database programming.

Common Properties in VB6

In this lesson, you'll learn about the common properties used in VB6.

BackColor and ForeColor

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

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

Example:

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

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

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

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

Another example:

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

The color can also be expressed in hexadecimal code.

Example:

```
Private Sub cmdChangeColor_Click()
   Label1.BackColor = &HFF&
   Label1.ForeColor = &HC00000
End Sub
```

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

Font

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

Example:

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

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

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

Caption

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

Example:

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

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

Text

It sets the text in a TextBox.

Example:

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

Here the text string is assigned to Text1.Text.

The Left, Top, Width and Height properties

1. The Left Property sets the distance between the internal left edge of an object and the left edge of its container.

2. The Top Property sets the distance between the internal top edge of an object and the top edge of its container.

3. The Width Property sets the width of an object.

4. The Height Property sets the height of an object.

Example:

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

Container

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

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

Example:

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

The command1 control will be moved into frame1.

Visible

Determines whether an object is visible or hidden.

Example:

```
Label1.Visible = False
```

Enabled

Determines whether an object can respond to user generated events.

Example:

```
Text1.enabled=False
```

MousePointer and MouseIcon

The MousePointer property sets the type of mouse pointer over an object. The values of MousePointer property are
0- Default
1- Arrow
2- Cross
3- I-Beam
4- Icon
5- Size etc.
The MouseIcon property sets a custom mouse icon from your files.

TabIndex and TabStop

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

The TabIndex property sets the tab order of an object.

Example:

```
Command3.TabIndex = 0
Command2.TabIndex = 1
Command1.TabIndex = 2
```

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

Control Box

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

ShowInTaskBar

Determines whether the form appears in the windows taskbar.

OLE(Object Linking & Embedding)

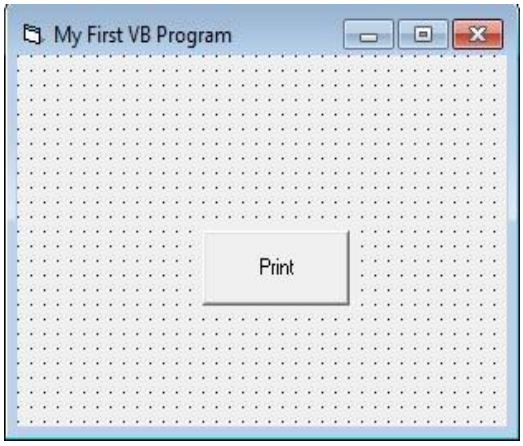
You can connect other programs to your application that you have developed.

Apart from them, there are many other controls provided by the Visual Basic language which will be discussed in the appropriate chapters. You can add external ActiveX controls that will enhance the interface and functionality of your program.

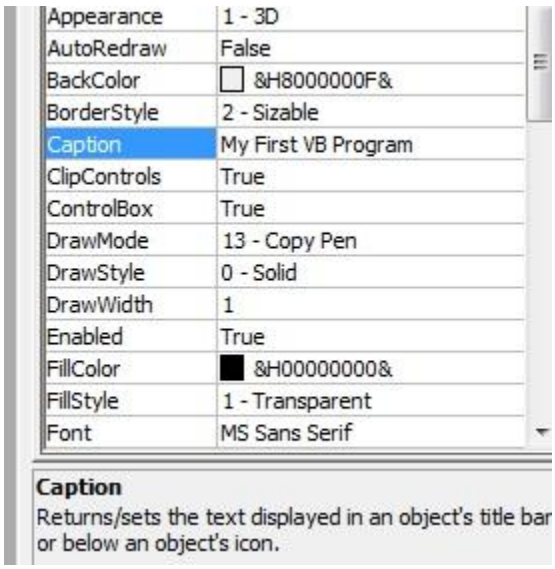
Your First Visual Basic 6 Program

This is your first Visual Basic program, an easy program to introduce you to VB programming.

Step 1: Start a .exe project.
Step 2: Place a CommandButton on the form from ToolBox.



Step 3: Click on Form and edit the caption property of the form. Write "My First VB Program".
Step 4: Edit the caption property of the CommandButton from the Properties Window. Write "Print".



Step 5: Double-Click the CommandButton to open the Code Window. Write the following code.

Write code between the two lines 'Private Sub Command1_Click()' and 'End Sub'. These two lines will be created automatically after double-clicking the CommandButton.

Explanation

'Sub' means Sub-Routine or Sub-Procedure or Function. 'Command1_Click()' is the sub-procedure name where 'Command1' is a control and 'Click' is an event. '( )' this sign indicates that 'Command1_Click()' is a function. 'Private' is the Scope. 'Print' is a Method.

You could also write Form1.print where Form1 is an object and print is its method. ‘End Sub indicates the end of the sub-procedure.

Press F5 to run the program or click the start icon button from the toolbar. The code will be executed and the string "Welcome to www.vbtutes.com" will be displayed on the form as shown in the picture below.

Writing comments

Comments are the lines of text that are not executed but used for the advantage of the programmers. Comments are written so that the other programmers can easily understand your program and you can better understand the code in case that it becomes complex or 16px. When you write comments, it becomes easy to maintain the code of your application. Commenting is a part of Documentation and it is a good practice to write comments.

How to write comments?

Comments are written using the apostrophe ('). That means if you write anything after apostrophe that becomes your comment and text color of the comments becomes green. See the picture below.

Naming Conventions in VB6

The Name property of a control is very important as it helps you identify the control in the code. Every control has the Name property. When you create a control, Visual Basic sets the default Name property like Text1, Command1, Form1 etc. It is suggested to use some specific prefixes for the Name property especially when you'll use the controls in your code. This is a good programming habit to modify the name so that it expresses or identifies a particular control with the meaningful name. See the table below for the three-letter prefixes that you will use for the naming purpose.

Table: Standard three-letter prefixes for controls

Controls	Prefix	Controls	Prefix
CommandButton	cmd	Data	dat
TextBox	txt	HScrollBar	hsb
Label	lbl	VScrollBar	vsb

PictureBox	pic	DriveListBox	drv
OptionButton	opt	DirListBox	dir
CheckBox	chk	FileListBox	fil
ComboBox	cbo	Line	lin
ListBox	lst	Shape	shp
Timer	tmr	OLE	ole
Frame	fra	Form	frm
Image	img	Menu	mnu

Say there is a TextBox control that shows the result, set the Name property to txtResult, a meaningful name. There is no need to edit the Name properties of all the controls in your project because naming all the controls (if there are so many) is time consuming. Name those controls using the prefixes which you use in the code. If it is convenient to use the default Name properties for some controls, there's no need to rename using the prefixes.

In fine, naming a control using a three-letter prefix is a good habit but only then when the particular control is used in the code and this naming convention will certainly increase the readability of your code, which is a great advantage.

Remember, there is no hard and fast naming rules. This is just a suggestion from Microsoft. In fact, this is your personal preference, you may prefix the name of a control in whatever way you want.

Concept of Event Driven Programming

Visual Basic is an event-driven programming language. Before proceeding to the next chapter, it is very important to have a good concept of event-driven programming. The common events are Click, DblClick, Load, MouseMove, MouseDown, MouseUp, KeyPress, KeyUp, KeyDown, GotFocus, LostFocus, etc.

When you click, press a key, move the mouse or fire other events, the

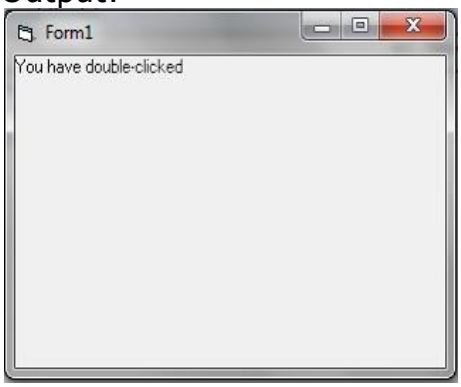
particular block of code of the corresponding event procedure is executed, and then the program behaves in a certain way. This is called event-driven programming.

When you fire an event, the code in the event procedure is executed, and then visual basic performs its operations as per the instructions written in the event procedure code. For example, in the first sample program, when you click the 'Print' button, the click event is fired, and then the code in the click event procedure gets executed. The code tells Visual Basic to print a text on the form. So as a result, you see a text printed on the form.

Example:
Write the following code in the DblClick event procedure of the form.

```
Private Sub Form_DblClick()
        Print "You have double-clicked"
End Sub
```

Output:



When you double-click on the form, the DblClick event procedure of the Form object is invoked, and then the code in the DblClick event procedure is executed. Thus, the code instructs Visual Basic to print a text on the form.

Unit II

Variables and Data Types

This VB6 lesson clarifies the concepts of variables and data types with examples. Some simple definitions and explanations have given here. Hope they will help you understand quickly and easily.

Variable

Variable is used to store value. The value of the variable may vary during the program execution.

Constant

Constant is a fixed value that does not change during the program execution. You can define your own constant to use it in your program.

Naming Rules of variables

1. A variable name must begin with an alphabet.
2. It cannot be more than 255 characters.
3. The variable name must not contain any special character like %,&,!,#,@ or $.
4. And finally, it has to be unique within the same scope.

Data Types

Visual Basic is rich in its data types. Data types are used to declare the variables. At the time of declaration, memory is allocated for the variables. Different data types are used to store different types of values.

Table: Memory storage for data types

Data Type	Storage	Data Type	Storage
Byte	1 byte	String (variable-length)	Length + 10 bytes
Boolean	2 bytes	String (Fixed-Length)	Length of string
Integer	2 bytes	Currency	8 bytes

Long	4 bytes	Decimal	12 bytes
Single	4 bytes	Object	4 bytes
Double	8 bytes	Variant (numeric)	16 bytes
Date	8 bytes	Variant (text)	length +22 bytes

Table: Data types & their value range

Data Type	Value Range
Byte	0 to 255
Boolean	True/False
Integer	-32,768 to 32,767
Long	-2,147,483,648 to 2,147,483,647
Single	-3.402823*10^3 to -1.401298*10^45 for negative values 1.401298*10^-45 to 3.402823*10^38 for positive values
Double	-1.79*10^308 to -4.94*10^-324 for negative values 4.94*10^-324 to 1.79*10^308 for positive values
Date	January 1, 100 to December 31, 9999
String (Variable length)	0 to approximately 2 billion characters
String (Fixed length)	1 to 65,400 characters
Currency	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	+,-79,228,162,514,264,337,593,543,950,335 if no decimal is used +,-7.9228162514264337593543950335 (28 decimal places)
Object	Any object
Variant (numeric)	Any value as 16px as Double
Variant (text)	Same as variable length string

Variable Declaration

Depending on where the variables are declared and how they are declared, there are many ways to declare a variable in visual basic. When you declare a variable, memory space for the variable is reserved. This is called memory allocation. Different amount of memory space is reserved for different data types.

You can declare a variable with the 'dim' keyword.

Syntax:

```
Dim variable As [Type]
```

Example:

```
Private Sub cmdSum_Click()
   Dim m As Integer
   Dim n As Integer
   Dim sum As Integer

   m = 10     'm is a variable, 10 is a constant
   n = 30
   sum = m + n

   Print "The sum is " & sum

End Sub
```

Output: The sum is 40

You can declare many variables in one line as follows and assign multiple variables in one line using ':' operator.

```
Private Sub cmdSum_Click()
   Dim m As Integer, n as Integer, sum as Integer
   m = 10 : n = 30
   sum = m + n

   Print "The sum is " & sum
```

```
End Sub
```

Output: The sum is 40

Implicit declaration : The Variant Data Type

If you use a variable without declaring its type, it is called a variant variable.

Example:

```
Dim num
Or,
Dim num As Variant
```

Or, if you choose not to declare a variable then also it is of the Variant data type. So you can use a variable in Visual Basic without declaring it. The variant data type can store numeric, date/time or string values. This is called implicit declaration. That means, you are declaring the variable implicitly. But it is not recommended to use implicit declaration and a good programmer will declare the variables explicitly. Because, it can lead to errors that may not be detected at run time.

Using the Option Explicit statement

The statement 'Option Explicit' is written in general declaration section that reports the user of any undeclared variable showing an error message. That means, if you forget to declare any variable, an error message will be shown reporting it.

Creating your own constants

You can create your own constant to use it in your program. The value of the constant remains unchanged throughout the program.

Example:

```
Private Sub cmdCalculate_Click()
    Const pi = 3.1415926   'or Const pi As Double = 3.1415926
    Dim area As Double, r As Double
    r = 2
```

```
   area = pi * r * r
   Print area
End Sub
```

Output: 12.5663704

Scope of variables:

Scope of variables determines which part of the code can access the variable. A variable is declared in the general declaration section of a module to make it available to all the procedures in the module.

Concepts of procedures and modules are necessary before jumping on the scope part. So procedures and modules are clarified first.

What is a procedure?

Procedure is a particular block of code. Examine the following example.
Example:
The following block of code is a procedure.

```
Private Sub Command1_Click()
   Dim num As Integer
   num = 45
   Print num
End Sub
```

'Command1_Click()' is the procedure name. Here 'num' is a procedure-level variable. The value of the variable is destroyed when the procedure ends.

'End Sub' indicates the end of the event procedure.

Types of procedures

There are three types of procedures - sub procedure, function procedure and property procedure.

Sub Procedures are of two types

a)General procedure and
b) Event procedure.

Module

Visual Basic uses modules. Three kinds of modules are there - form modules, standard modules, and class modules.

i) Form module: The collection of procedures in a form is referred to as form module.

ii) Standard Module: (.bas extension)
The standard module has the .bas extension. Go to Menu ->Project ->add module to add a module to your application.

ii) Class Module: (.CLS extension)
Class module is used for object oriented programming.

Procedure Level Variable or Local Variable

Local variables are of two types - dynamic local variable and static local variable.

Dynamic local variable: A Dynamic local variable is declared inside a procedure using the 'dim' keyword. Local variables are only available to the procedure in which they are declared. Other parts of the code are unaware of its existence. Values of the variables declared using the 'dim' keyword in a procedure exist only within the procedure.

Example:

```
Private Sub Command1_Click()
   Dim num As Integer  'This is a dynamic local variable
End Sub
```

Static Variable: Static variable is a procedure level variable.

Syntax:

Static variable As [Type]

Example:

```
Static m As Integer
```

Static variables are declared inside a procedure. The static variables retain their values even when a procedure ends but their scope is the procedure itself.

Module level variable

By default, a module level variable is accessible from all the procedures in the module but not from other modules. A module level variable is declared in the declarations section, at the top of the module using the Dim or Private keyword. There is no difference between Dim and Private at the module level.

Example:

```
'In the Declarations Section
Private s As String
```

Public Variable or Global variable

To make a variable accessible to all the modules or throughout the application, declare it with the Public keyword in the declarations section of a module. Then the value of the variable becomes available to all the procedures of your application.

Note: You cannot declare a public variable inside a procedure. Public variables are declared only within the general declarations section of a module.

Example:

```
'Form1
'In the form's declaration section
Public n As Integer
```

Operators and Expressions

More operators are provided by the Visual Basic 6 language.

Table: Operators and their meanings

Operators	Meanings	Operators	Meanings
+	Addition	>=	Greater than or equal to
-	Subtraction	<>	Not equal to
*	Multiplication	=	equal to
/	Division	&	String Concatenation
\	Integer Division	And	Logical And
Mod	Modulo Division	Not	Logical Not
<	Less than	Or	Logical Or
>	Greater than	Xor	Logical Xor
<=	Less than or equal to	^	Power

Assigning values to variables : '=' operator

The Assignment operator (=) is used to assign a value or an expression to a variable. When you assign a value, its not the same as the Equal to operation, but the value is copied to a variable. Besides assigning values and expressions, you can assign a variable to a variable also

Example:
x=a+b
'=' , '+' are the operators.
x,a,b are the operands and 'a+b' is an expression. x=a+b is a statement.

The following program will print 20.

```
Private Sub cmdDisplay_Click()
    Dim num As Integer, r As Integer, n As Integer
    n = 10
    r = n ^ 2      'n to the power 2
    num = n + r / n
    Print num
End Sub
```

Output: 20

The Integer Division Operator ('\')

The Integer Division operator('\') eliminates the decimal part after division.

Example:

```
Private Sub cmdDivision_Click()
    Dim a As Double, b As Double, c As Double
    a = 12: b = 5
    c = a \ b
    Print c
End Sub
```

Output: 2

The 'Mod' operator

The Mod operator is used to calculate remainder.

Example:

```
Private Sub cmdShow_Click()
     Dim remainder As Integer, num As Integer
     num = 21
     remainder = num Mod 10
     Print remainder
End Sub
```

Output: 1

Boolean Operators : And, Or, Not, Xor

Input and Output Operations in VB6

Visual Basic provides some excellent ways for input and output operations. Input can be taken using TextBox, InputBox, and output can be shown with the Print method, TextBox, Label, PictureBox and MsgBox.

Input and output using TextBox

The TextBox Control provides an efficient way for both input and operations. The following sample program shows it.

Example:

```
Private Sub Command1_Click()
   Dim a As Integer
   a = Val(Text1.Text)
   a = a + 1
   Text2.Text = a
End Sub
```

In the above program, Input is taken using Text1 and output is shown in Text2, where output is the incremented value of the input.

Line continuation character(_)

In your vb program you can type maximum of 1023 characters in a line. And sometimes, it doesn't fit in the window when we write so many characters in a line. So Line Continuation Character (underscore preceded by a space) is used to continue the same statement in the next line.

Example:

```
Private Sub cmdShow_Click()
   MsgBox "Hello, welcome to www.vbtutes.com", _
  vbInformation, "Welcome"
End Sub
```

Data Type Conversions in Visual Basic 6

Variable or expression on the right hand side of the assignment operator must first be converted into the target type, otherwise assignment will not be possible.

In some cases, Visual Basic does it for you. That means that you don’t need to write code to convert the data types; conversion is done automatically. But this doesn’t happen all the time, so you need to learn how to convert them.

Implicit data type conversion

In many cases, Visual Basic does the conversion job automatically. For example, when a numeric value is assigned to a String variable, the type casting is done automatically.

In the following example, Visual Basic implicitly converts the value of number to string before assigning to msg.

Example:

```
Dim number As Integer, msg As String
number = 44545
msg = number
Print msg
```

Explicit conversion

In many cases you have to explicitly convert the data values to another data type. Visual Basic has provided so many useful conversion functions.

Example: Works without the CInt function also.

```
Dim n As Integer, d As Double
d = 5.6767
n = CInt(d)
Print n
```

Output: 6

Conversion functions

The functions are CBool, CByte, CCur, CDate, CDbl, CDec, CInt, CLng, CSng, CStr, and CVar. Val and Str functions are also used.

CStr

The Cstr function converts an expression to a string. This is useful when you’re displaying some numeric values in a text box control.

Example:

```
Dim v1 As Double
Text1.Text = CStr(v1)
```

Str

The Str function converts a number to a string. It is similar to the Cstr function.

Example:

```
Dim str1 As String, m As Integer
str1 = Str(m)
```

CDbl

The CDbl function converts an expression into a Double. It also extracts the numeric value from a string.

Example: If you omit the CDbl function, the program will raise an overflow error.

```
Dim m As Integer, v As Double
m = 30887
v = CDbl(m) * 31880
Print v
```

Example:

```
Dim value As Double
value = CDbl(Text1.Text)
Print value + 1
```

Example:

```
Dim value As Double
value = CDbl("55")
Print value + 1
```

Val

Sometimes you can use the Val function instead of the CDbl function. Val function returns a Double value. It only returns the numeric value contained in a string. Unlike the CDbl function, it cannot convert a numeric expression.

Example:

```
Dim l As Double
l = Val(Text1.Text)
```

CInt

The CInt function converts a number to integer.

Example:

```
Dim m As Integer
m = CInt(876.878)   'Returns 877 after roundin the no.
m = CInt(-2.7)   'Returns -3 after rounding the no.
```

The argument must be within the Integer range here.

876.878 and -2.7 are the arguments in the above code.

CLng

The CLng function converts to Long type.

Example:

```
Dim ln As Long
ln = CLng(1147483647.87656) 'Rounds up the value
Debug.Print ln
```

CBool

The CBool function converts an expression to Boolean Data type, i.e either True or False. The value of zero is converted to False and all other numeric values are converted to True. The strings “True” and “False” return True and False Boolean values respectively.

Example:

```
Dim B as Boolean
B=cbool(0)       'Returns False
B=cbool(1)       'Returns true
B=cbool(345)     'Returns True
```

CByte, CSng, CVar,CCur,CDate and CDecimal functions convert to Byte, Sinle, Variant, Currency, Date and Decimal values respectively. You can use these functions in the same way explained above.

Control Structure

Prgrams are not monolithic sets of commands that carry out the sdame calculationa 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 6.0 provides three-control flow, or decision, structuires to take a course of action depending on the outcome pf the test. They are:

- IF … Then
- If … Then … Else
- Select Case

IF Blocks in VB6.0

The If control flow blocks provide a great way for decision making where one or more statements are executed depending upon a condition.

If-Then

In case of If-Then block, the condition is first checked. The condition can either be True or False. If it is True, the statements inside the If-Then block is executed. Otherwise, it does not execute the code in the If-Then block, the If-Then structure is skipped. It starts executing the statements just after the 'End if' phrase.

Syntax:

```
If Condition Then
    statements
End If
```

Example: To check if the number is positive.

```
Dim num1 As Integer
num1 = 30
If num1 > 0 Then
   Print "The number is positive"
End If
```

Output: The number is positive.

Single Line version of If-Then

Single line and single statement

```
If a > 0 Then b = 1
```

Single line and multiple statements

```
If a > 0 Then b = 1: c = 2
```

If-Else

An If-Else block has two parts. If the condition is True then the first part is executed, otherwise the control goes to the second part and starts executing the lines of statements in the second part of the If-Else block.

Syntax:

```
If Condition Then
    statements1
Else
    statements2
End If
```

If Condition is true then statements1 will be executed and if Condition is false, statements2 will be executed.

Example: To print the largest number.

```
Dim num1 As Integer, num2 As Integer
num1 = 30
num2 = 50
If num1 > num2 Then
   Print num1
```

```
Else
   Print num2
End If
```

Output: 50

Single line version of If-Else

```
If a > 0 Then b = 1 Else b = 0
```

Nested If-Else

If you have programmed in other languages, you might probably know about nested if-else control flow block. In VB it is the same concept. See the syntax and the examples to capture the concept.

In nested if-else, the structure is little changed, little advanced better to say, from the simple If-Block. Here if the first condition does not satisfy, it looks for the next condition preceded by the ElseIf term, if that too does not satisfy, it looks for the next, and it goes on until the end of the structure.

Syntax:

```
If Condition Then
        statements
ElseIf Condition then
        statements
ElseIf Condition Then
         statements
.....................
.....................
Else
       statements
End If
```

Example:

```
a = Val(InputBox("Enter a no."))

If a > 0 Then
   Print "Positive"
ElseIf a < 0 Then
   Print "Negative"
```

```
Else
   Print "Zero"
End If
```

In a nested if-else block, one If-Block can reside in another. See the example below.

Example:

```
'If-statement inside another if-statement
Dim a As Integer
a = Val(txtNumber.Text)

If a > 0 Then
   Print "Positive"
   If a > 2 Then
      Print "Greater than 2"
   Else
      Print "Less than 2"
   End If
Else
   If a < 0 Then
      Print "Negative"
   End If
End If
```

Boolean Operators : And, Or, Not, Xor

And indicates that all expressions are true, *Or* indicates that one of the expressions or all are true, *Not* indicates negation, *Xor* indicates that one of the expressions are true.

Example:

```
Dim num1 As Integer, num2 As Integer
num1 = InputBox("Enter 1st number")
num2 = InputBox("Enter 2nd number")

If (num1 > 0) And (num2 > 0) Then
   MsgBox "Both the numbers are positive"
End If
If (num1 > 0) Or (num2 > 0) Then
```

```
   MsgBox "Either 1st number or 2nd number or both are positive"
End If
If Not (num1 = 0) Then
   MsgBox "The first number is non-zero"
End If

If (num1 > 0) Xor (num2 > 0) Then
   MsgBox "Either 1st number or 2nd number is positive"
End If
```

Select Case Blocks

This lesson takes you through the Select Case block, a very useful decision making structure in Visual Basic.

The Select Case block provides an efficient way for decision making which is used to execute one block of statement(s) among a list of statement blocks according to the determined value of an expression. The expression is preceded by the "Select Case" phrase. The choices are made based on the value of this expression

Syntax:

```
Select Case expression
Case value0
       statements
Case value1
       statements
Case value2
      statements
.....
.....
Case else
       statements
End select
```

Example:

```
Dim num As Integer
num = Val(Text1.Text)
Select Case num
Case 0
```

```
   Print "You have entered zero"
Case 1
   Print "You have entered one"
Case 2
   Print "You have entered two"
Case Else
   Print "The number you entered is greater than 2"
End Select
```

One block of statement(s) will be executed depending upon the value of num. If num=0 then, Case 0 will be evaluated and if num =1 then Case 1 will be evaluated.

See the other forms of Select Case Structure in the following programs.

Example:

```
Dim score As Integer
score = Val(Text1.Text)
Select Case score
Case 900 To 1000
   Print "First position"
Case 800 To 899
   Print "Second position"
Case 700 To 799
   Print "Third position"
Case Else
   Print "No position, try again"
End Select
```

Example:

```
Dim num As Integer
num = Val(Text1.Text)
Select Case num
Case Is > 0
   Print "Positive number"
Case Is < 0
   Print "Negative number"
Case 0
   Print "Zero"
End Select
```

Example:

```
Select Case value
   Case 0, 1, 2, 3, 4, 5
      Print "The values are 0,1,2,3,4,5"
   Case Else
      Print "Other values"
End Select
```

Do Loops

Loops are used to execute a block of statements repeatedly based on a condition.

There are different forms of Do Loops in Visual Basic 6. These are:

i) Do While...Loop
ii) Do...Loop While
iii) Do...Loop Until.

i) Do While...Loop: The Do While...Loop structure is used to execute statements repeatedly based on a certain condition.
It first tests the condition then evaluates the loop. If the condition is True, the statements block is executed. If the condition is false, the statements inside the loop are not executed and the control is transferred to the line after the loop statements.

The repeated execution of the statements inside the loop goes on as long as the condition is true. When the condition becomes False, it exits the loop.

Syntax:

```
Do While Condition
    statement(s)
Loop
```

Example: To print from 0 to 9.

```
Dim num As Integer
num = 0
Do While num < 10
   Print num
   num = num + 1
Loop
```

The program first checks the condition. If num is less than 10, the statements inside the Do While...Loop are executed. Again, the condition is checked. If it is True, the statements are executed. In this way, the iterative execution goes on. When the value of num becomes 10, the loop ends.

ii) Do...loop while:

This loop structure first executes the statements and after that tests the condition for the repeated execution.

Syntax:

```
Do
    statement(s)
Loop while Condition
```

Example: To print from 0 to 10.

```
Dim num As Integer
num = 0
Do
   Print num
   num = num + 1
Loop While num <= 10
```

Another example: Though the condition does not satisfy, the program will print 11 as this loop structure first executes the statements and after that tests the condition.

```
Dim num As Integer
num = 11
Do
   Print num
   num = num + 1
Loop While num < 10
```

Output: 11

iv) Do...loop until:

The Do...Loop Untill structure executes the statements repeatedly until the condition is met. It is an infinite loop if the condition does not satisfy. In this case, the program cannot be stopped. Press Ctrl+Break key combination to force it to stop. The loop continues until the condition is met. The repeated execution of the statements stops when the condition is met.

Syntax:

```
Do
    statement(s)
Loop Until Condition
```

Example:

```
'x is incremented until x becomes greater than 10

Dim x As Integer
x = 0
Do
   x = x + 1
Loop Until x > 10

MsgBox x
```

For...Next Loops

For...Next loop structure is useful when a block of statements are to be executed for unknown number of times. But if a block of statements are to be executed for specified number of times then a For … Next loop is better choice. Unlike a Do loop, a For loop uses a variable called a counter that increases or deceases in value during each repetition of the loop.

Syntax

For counter = start **To** end [step increment]

Statements

Next [counter]

The argument counter, start, end, and increment are all numeric. The increment argument can be either positive or negative If increment is positive, start must be less than or equal to end or the statements in the loop will not execute. If increment is negative, start must be greater than or equal to end for the body of the loop to execute. If Step isn't set, then increment defaults to 1.

Example: To print 0 to 10.

```
Dim i As Integer
For i = 0 To 10
   Print i
Next i
```

When the value of i is 0, the Print statement is executed then i is incremented to 1. It checks whether the value of i is from 0 to 10. If it satisfies, the Print statement is executed again. In this way, the loop goes on until the value of i exceeds 10. Every time, the value of i is incremented by 1.

Example: To print 0 to 6 in steps of 2.

```
Dim i As Integer
For i = 0 To 6 Step 2
   Print i
Next i
```

Every time, the value of i is incremented by 2.

Output:
0
2
4
6

Example: To print in descending order from 10 to 0 in step of -3.

```
Dim i As Integer
For i = 10 To 0 Step -3
   Print i
Next i
```

Every time, the value of i is decremented by 3.

Output:
10
7
4
1

For Each … Next

A For Each … Next loop is similar to a For … Next loop, but it repeats a group statements for each element in a collection of objects or in an array instead of repeating the statements a specified number of times This is especially helpful when the number of elements of a collection is not known.

Syntax

For Each element **In** group

Statements

Next element

Exit For and Exit Do statement

A For Next Loop can be terminated by an Exit For statement and a Do loop can be terminated by an Exit Do statement.

Example: Exit For statement :

```
Dim i As Integer
For i = 0 To 10
   If i = 3 Then
   Exit For
   End If
Print i
Next i
```

Output:
0
1
2

Example: Exit Do statement

```
Dim num As Integer
num = 0
Do While num < 10
   Print num
   num = num + 1

   If num = 4 Then
      Exit Do
   End If

Loop
```

Output:
0
1
2
3

The OptionButton Control

This control lets you choose from several items among other in a list. This is one of the mostly frequently used controls in application developement. When you click on an OptionButton, it is switched to selected state or ON state, and all other option buttons in the group become unselected or OFF.

Example:

```
Private Sub Command1_Click()
    If Option1.Value = True Then
        Print "Option1 is ON"
    Else
        Print "Option1 is OFF"
    End If
End Sub
```

Output:

When Option1.value = True then the Option Button is ON and when Option1.value = False then the Option button is OFF.

Example:

```
Private Sub cmdCheck_Click()
    If optWindows.Value = True Then
        Print "Windows is selected"
    ElseIf optLinux.Value = True Then
        Print "Linux is selected"
    ElseIf optMac.Value = True Then
        Print "Mac is selected"
    End If
End Sub
```

Output:

Check Box Control

This control has three states : checked, unchecked and grayed. The value property determines the checkbox state.

Example:

```
Private Sub cmdShow_Click()
   If Check1.Value = 1 Then
      Print "Checked !"
   ElseIf Check1.Value = 0 Then
      Print "Unchecked !"
   End If
End Sub
```

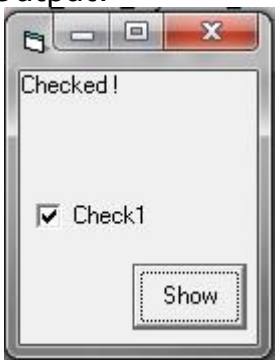
Output:

If you assign Check1.Value =1 then the CheckBox is checked and if Check1.Value = 0 then it is unchecked. If Check1.Value = 2 then the checkbox is grayed.

Example: The previous program can also be written in the following way.

```
Private Sub cmdShow_Click()
   If Check1.Value = vbChecked Then
      Print "Checked !"
   ElseIf Check1.Value = vbUnchecked Then
      Print "Unchecked !"
   End If
End Sub
```

Output:



'vbChecked' and 'vbUnchecked' are vb constants whose values are 1 and 0 respectively.
The grayed state can be set using the 'vbGrayed' vb constant whose value is 2.

Example: Program to make multiple choices.

```
Private Sub cmdShow_Click()
    If chkTea.Value = 1 Then
       Print "Tea"
    End If

    If chkCoffee.Value = 1 Then
       Print "Coffee"
    End If

    If chkPizza.Value = 1 Then
       Print "Pizza"
    End If
```

```
    If chkChocolate.Value = 1 Then
       Print "Chocolate"
    End If
End Sub
```

Output:

Select Case Blocks

The Select Case block provides an efficient way for decision making which is used to execute one block of statement(s) among a list of statement blocks according to the determined value of an expression. The expression is preceded by the "Select Case" phrase. The choices are made based on the value of this expression.

Syntax:

```
Select Case expression
Case value0
        statements
Case value1
        statements
Case value2
       statements
.....
.....
Case else
        statements
End select
```

Example:

```
Dim num As Integer
num = Val(Text1.Text)
Select Case num
Case 0
   Print "You have entered zero"
Case 1
   Print "You have entered one"
Case 2
   Print "You have entered two"
Case Else
   Print "The number you entered is greater than 2"
End Select
```

One block of statement(s) will be executed depending upon the value of num. If num=0 then, Case 0 will be evaluated and if num =1 then Case 1 will be evaluated.

See the other forms of Select Case Structure in the following programs.

Example:

```
Dim score As Integer
score = Val(Text1.Text)
Select Case score
Case 900 To 1000
   Print "First position"
Case 800 To 899
   Print "Second position"
Case 700 To 799
   Print "Third position"
Case Else
   Print "No position, try again"
End Select
```

Example:

```
Dim num As Integer
num = Val(Text1.Text)
Select Case num
Case Is > 0
   Print "Positive number"
Case Is < 0
   Print "Negative number"
Case 0
```

```
    Print "Zero"
End Select
```

Example:

```
Select Case value
    Case 0, 1, 2, 3, 4, 5
        Print "The values are 0,1,2,3,4,5"
    Case Else
        Print "Other values"
End Select
```

Unit III

Arrays

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

Declaring an array

Syntax: Dim Variable_Name(index) As [Type]

Example:

```
Dim month(10) As Integer   '11 elements
'or
Dim month(1 to 12) as Integer   '12 elements
```

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

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

Types of array

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

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

The LBound and UBound functions

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

Example:

```
Private Sub cmdDisplay_Click()
    Dim arr(10) As Integer
    a = LBound(arr)
    b = UBound(arr)

    MsgBox "Lower bound = " & a & "  Upper bound = " & b

End Sub
```

Initializing an array

You can use For Loop to initialize an array.

Example:

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

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

Example: This program inputs the Sale amount of each day and sums the total amount of 5 days.

```
Private Sub cmdStart_Click()
    Dim SaleDay(1 To 5)      'Sale in a particular day
    Dim i As Integer, Sale As Long
    Sale = 0

    For i = 1 To 5
        SaleDay(i) = InputBox("Enter Sale amount of Day " & i)
```

```
        Sale = Sale + SaleDay(i)
    Next i

    MsgBox "Total Sale of 5 days = $" & Sale

End Sub
```

Multi-Dimensional Arrays:

An array can be multi-dimensional that means, it can have more than one dimension. A list of data is represented in a one-dimensional array where a multi-dimensional array represents a table of data. An array can be two dimensional, three dimensional and so on. We generally don't need an array of more than two dimensions, it is enough to use one and two dimensional arrays. You can use a higher dimensional array when the program is too complex. A two dimensional array takes the row-column form.

Declaration:

```
Dim value(5, 5) As Integer       'two dimensional
'Or,
Dim value(1 to 5, 1 to 5) As Double
Dim number(6, 9, 8) As Integer  'three dimensinoal
```

Initialization:

To initialize the array elements, you first need to recognize the array elements. For example, the array 'value(2, 2)' has 9 elements. They are value(0,0), value(0,1),value(0,2), value(1,0), value(1,1), value(1,2), value(2,0), value(2,1), value(2,2). For the initialization, you may wish to use For Loop or initialize each element like variables. Using For Loop is a better choice as it reduces the number of lines of code. But sometimes, separately initializing each element like a variable is much more convenient. And it also depends on the type of program you are writing.

Addition of 2D matrices : Example of a two dimensional array

Example:

```
Private Sub cmdSum_Click()
    Dim matrix1(1, 1) As Integer, matrix2(1, 1) As Integer
    Dim sum(1, 1) As Integer
```

```
    'initializiation of matrix1
    matrix1(0, 0) = Val(Text1.Text)
    matrix1(0, 1) = Val(Text2.Text)
    matrix1(1, 0) = Val(Text3.Text)
    matrix1(1, 1) = Val(Text4.Text)

    'initializiation of matrix2
    matrix2(0, 0) = Val(Text5.Text)
    matrix2(0, 1) = Val(Text6.Text)
    matrix2(1, 0) = Val(Text7.Text)
    matrix2(1, 1) = Val(Text8.Text)

    'Summation of two matrices
    For i = 0 To 1
        For j = 0 To 1
            sum(i, j) = matrix1(i, j) + matrix2(i, j)
        Next j
    Next i

    'Displaying the result
    Print "The resultant matrix"
    For i = 0 To 1
        For j = 0 To 1
            Print sum(i, j);

        Next j
        Print ""
    Next i
End Sub
```

Static array

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

```
' This is a static array.
Dim Names(100) As String
```

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

Most programs don't use static arrays because programmers rarely know at compile time how many items you need and also because static arrays can't be resized during execution. Both these issues are solved by dynamic arrays. You declare and create dynamic arrays in two distinct steps. In general, you declare the array to account for its visibility (for example, at the beginning of a module if you want to make it visible by all the procedures of the module) using a Dim command with an empty pair of brackets. Then you create the array when you actually need it, using a ReDim statement:

```
' An array defined in a BAS module (with Private scope)
Dim Customers() As String
...
Sub Main()
' Here you create the array.
ReDim Customer(1000) As String
End Sub
```

If you're creating an array that's local to a procedure, you can do everything with a single ReDim statement:

```
Sub PrintReport()
' This array is visible only to the procedure.
ReDim Customers(1000) As String
' ...
End Sub
```

If you don't specify the lower index of an array, Visual Basic assumes it to be 0, unless an Option Base 1 statement is placed at the beginning of the module. My suggestion is this: Never use an Option Base statement because it makes code reuse more difficult. (You can't cut and paste routines without worrying about the current Option Base.) If you want to explicitly use a lower index different from 0, use this syntax instead:

```
ReDim Customers(1 To 1000) As String
```

Dynamic Array

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

Declaring dynamic arrays

1. Declare the array with empty dimension list.
   Example : Dim arr() As Integer

2. Resize the array with the ReDim keyword.
Example : ReDim arr(5) As Integer
or, ReDim arr(2 To 5) As Integer

Example:

```
Dim ar() As Integer
ReDim ar(2) As Integer
```

Note: Unlike the Dim and Static statements, the ReDim statements are executable. So a ReDim statement can only be in a procedure and when you execute the ReDim statement, all the values stored in the array are lost. You can use the ReDim statement repeatedly to change the array size.

Preserving the values of Dynamic arrays

The ReDim statement deletes all the values stored in the array. You can preserve the element values using the Preserve keyword. So using Preserve keyword with ReDim statements enables you to change the array size without losing the data in the array.

Example:

```
Dim arr() As Integer
ReDim arr(2) As Integer

For i = 0 To 2
   arr(i) = InputBox("Enter the value")
Next i

ReDim Preserve arr(3) As Integer
arr(3) = 9
Print arr(0), arr(1), arr(2), arr(3)
```

Output: If the input values through InputBox are 5,6,7 then the following will be printed on the form.
5 6 7 9

Functions and Procedures

A function procedure in Visual Basic 6 has a scope, a unique name, parameter list and return value. You can pass any datatype to a procedure e.g Integer, Boolean, Long, Byte, Single, Double, Currency, Date, String and Variant. Object data types and arrays are also supported. This is same for the return type values.

Difference between argument and parameter

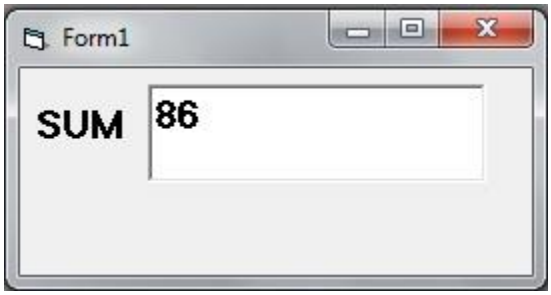
The value you're passing, while calling the function, is called argument and the variable, in the function definition, that will receive the value is called parameter. Both the terms are used for the same value.
A function procedure may not return a value.

Example: In this example, 32 and 54 are passed to the function 'sum' from Form_Load procedure.

```
'Function Definition
Private Function sum(n1 As Integer, n2 As Integer) 'n1, n2 are 'parameters
    Text1.Text = n1 + n2
End Function
_____________________________________________________________________
__
Private Sub Form_Load()
    Text1.Text = ""

   'Function callgin
    Call sum(32, 54)   '32 and 54 are arguments
End Sub
```

Output:



Function procedure that returns value

Example:

```
'Function Definition
Private Function sum(n1 As Integer, n2 As Integer) As Integer
    'Returns a value
    sum = n1 + n2
End Function
_____________________________________________________________________
Private Sub Form_Load()
    Text1.Text = ""
    'Function calling and assigning the returned value
    Text1.Text = sum(60, 40)
End Sub
```

Passing arguments: By Value or By Reference

You can pass an argument either by value or by reference. Arguments are passed by value using the ByVal keyword and by reference using the ByRef keyword or by omitting any specifier.

While passing the arguments by reference, references of the variables are passed. So if the argument is passed by reference, it can be modified by the called procedure and the original value of the argument in the calling procedure will be changed. But the argument value will be unchanged if you call the procedure using constants or expressions as parameters.

Example:

```
'Calling procedure
Private Sub Command1_Click()
    Dim a As Integer      'The value of a is 0 after declaration
    Call num(a)

    Print a    'Value of a is 1, modified
End Sub
_____________________________________________________________________
__
'Called procedure
Public Function num(ByRef x As Integer) 'You may omit ByRef
```

```
   x = x + 1
End Function
```

On the other hand, when the arguments are passed by value, the actual values are passed. So the called procedure cannot change their original values in any way.

Example:

```
'Calling procedure
Private Sub Command1_Click()
   Dim a As Integer    'The value of a is 0 after declaration
   Call num(a)

   Print a  'The value of a is 0, its unchanged
End Sub
_____________________________________________________________________
___
'Called procedure
Public Function num(ByVal x As Integer)
   x = x + 1
End Function
```

Note: Both Sub and Function procedures can accept parameters.

Control Array

Till now we have discussed array of variables. Similarly you can also have an array of controls by grouping a set of controls together. The controls must be of the same type like all TextBoxes or all CommandButtons.

Creating control arrays

1. Place some same type of controls say CommandButtons on the form. Make their name properties same and then a warning (see picture below) dialog box will come asking whether you want to create a control array, click Yes.

Or, after placing a control on the form, copy that and paste on the form. It will create control array for you.

2. Set the Index property of each control or you may not change as it is automatically set.

3. Now its done. You are ready to use the control array in your code.

Using control array

Syntax to refer to a member of the control array :
Control_Name(Index).Property

Example: Set the Style property of Command1(1) to 1 to work with the BackColor property.

```
Private Sub Command1_Click(Index As Integer)
   Command1(1).BackColor = vbGreen
End Sub
```

Example: Create a control array of 5 CommandButton controls and then set their Style property to 1.

```
Private Sub Command1_Click(Index As Integer)
   Dim i As Integer

   For i = 0 To 4
      Command1(i).BackColor = vbBlue
```

```
   Next i
End Sub
```

You can also pass a value to the Index parameter from other procedures.

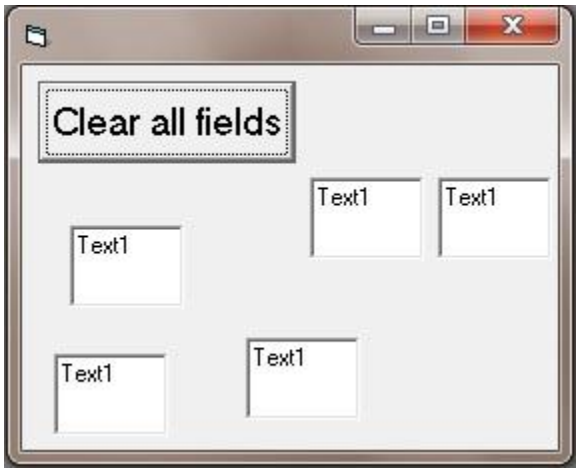
Control array is useful when you want to clear a set of TextBox fields. Create a control array of 5 TextBox controls and write the following code in the Click event procedure of a CommandButton control.

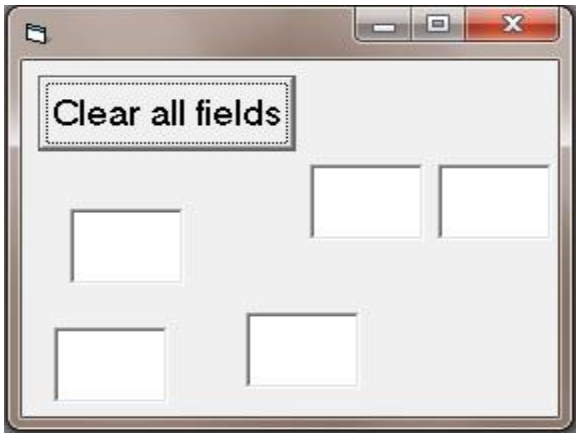
Example:

```
Private Sub cmdClearAllFields_Click()
   Dim i As Integer

   For i = 0 To 4  'Or, For i=Text1.LBound To Text1.UBound
      Text1(i).Text = ""
   Next i
End Sub
```

Output:





Sharing Event procedures

Create a control array of some command buttons with the name "Command1". Note that Visual Basic automatically passes the Index parameter value. So the following code will work for all controls in the control array. You don't need to write code for all the CommandButton controls. Click on any CommandButton, the following single block of code will work for all.

Example:

```
Private Sub Command1_Click(Index As Integer)
   Command1(Index).BackColor = vbBlack
End Sub
```

Creating controls at run-time

Once you have created a control array, you can create controls at run-time using the Load command.

Example: First of all, create Text1(0) and Text1(1) at design time.

```
Private Sub Command1_Click()
   Load Text1(2)

   'Move the control where you want
   Text1(2).Move 0, 100
```

```
    Text1(2).Visible = True
End Sub
```

On clicking the Command1 button, a new TextBox control will be created.
You can remove any control from the control array using the Unload command.

```
Unload Text1(2).
```

Unit IV

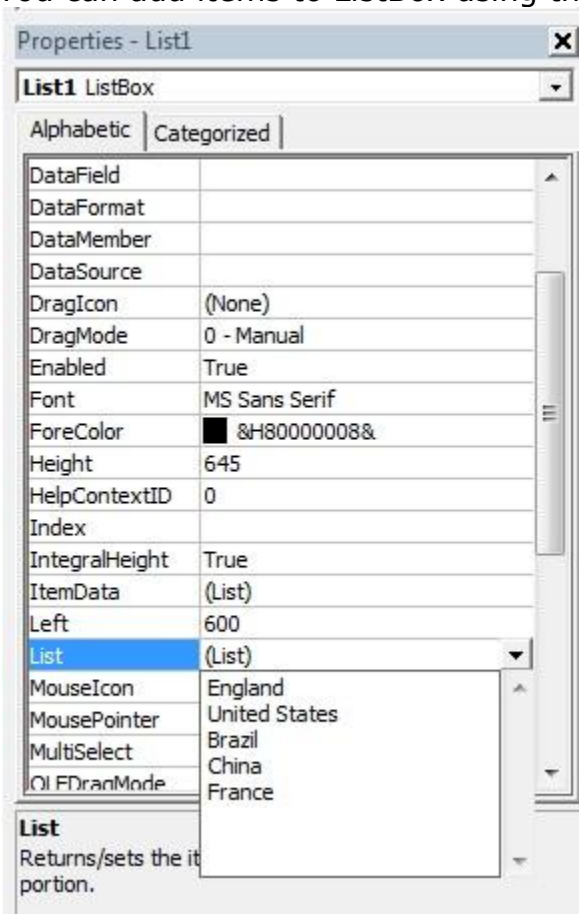
Advanced Controls: ListBox Control

This lesson shows you how to work with the ListBox control in Visual Basic 6.

The first thing that you may want to do with the ListBox control is to add items or elements to it. You have two options for that. You can either add items in design-time or in run-time.

Adding items in design time

You can add items to ListBox using the List property.



Adding items in run-time

You can add items to ListBox using the AddItem method.

Example:

```
Private Sub Form_Load()
   List1.AddItem "England"
End Sub
```

Here List1 is the ListBox name.

Output:

Deleting an item from ListBox

You can remove an item using the RemoveItem method.

Syntax:

ListBox.RemoveItem n

n is the index of an item. The Index starts from 0, so the index of the second item is 1.

Example: This program will delete the 1st item from ListBox.

```
Private Sub Form_Load()
   List1.RemoveItem 0
End Sub
```

Deleting all items

The Clear method removes all items from the Listbox.

Example:

```
Private Sub Form_Load()
   List1.Clear
End Sub
```

Number of items in the ListBox

The ListCount property counts items in a ListBox.

Example:

```
Private Sub Form_Load()
    Form1.Show
    Print lstCountry.ListCount
End Sub
```

Output:

Index number of the recently added item

The NewIndex property gives the index number which is most recently added using the AddItem method.

Example:

```
Private Sub Form_Load()
    Form1.Show
    lstCountry.AddItem "England"
    lstCountry.AddItem "USA"
    lstCountry.AddItem "Germany"
    Print lstCountry.NewIndex
End Sub
```

Output: 2

Index number of the currently highlighted item

Index number of the currently highlighted item can be obtained using the ListIndex property. The value of ListIndex is -1 if no item is highlighted.

Example:

```
Private Sub cmdShow_Click()
    Print lstCountry.ListIndex
```

```
End Sub
_____________________________________________________________________
Private Sub Form_Load()
lstCountry.AddItem "England"
lstCountry.AddItem "USA"
lstCountry.AddItem "Germany"
End Sub
```

Output:

The List property

The particular item of the ListBox having index n can be obtained using the List(n) property.

Example:

```
Private Sub Form_Load()
Form1.Show
lstCountry.AddItem "England"
lstCountry.AddItem "USA"
lstCountry.AddItem "Germany"
    Print lstCountry.List(0)
End Sub
```

Output: England

Currently highlighted item

The text property of ListBox gives the currently highlighted item/string.

Example:

```
Private Sub Command1_Click()
    Print lstCountry.Text
End Sub
```

```
Private Sub Form_Load()
    lstCountry.AddItem "England"
    lstCountry.AddItem "USA"
    lstCountry.AddItem "Germany"
End Sub
```

Output:

ComboBox

ComboBox is the combination of a TextBox and a ListBox. The user can type in the text field to select an item or select an item manually from the list. All the properties, events and methods of a ComboBox control are as same as the ListBox control. So the discussion of ListBox control in previous lessons also applies for ComboBox control.

Styles of ComboBox

There are three styles of ComboBox controls-- Dropdown Combo, Simple Combo and Dropdown List. You can change/select the style from the Style property of ComboBox.

Example:


```
Private Sub cmdSelectBirthYear_Click()
    Print "Your year of birth is" & cboBirthYear.Text
End Sub
```

```
_
Private Sub Form_Load()
    For i = 1980 To 2012
        cboBirthYear.AddItem Str(i)   'Str function returns the
                                      'string representation of a number
        i = Val(i)
    Next i
End Sub
```

Timer Control

Timer control is used to execute lines of code at specific intervals. Using this control, you can create animations without any knowledge of graphics or

animation. Though you cannot create some groundbreaking animations using timer control, but it is sufficient for your application. You can create an animated menu for your software. This control is not visible at runtime.

Some properties of the Timer control

- **Enabled:** To activate or deactivate the timer.
- **Interval:** Number of milliseconds between calls to a timer control's timer event.

For example, set the Interval to 100 , to execute the code in the timer event procedure every 100 milliseconds.

1000 milliseconds = 1 second.

Example: Now place a Timer control and a CommandButton on the form. Set its Enabled property to False and set the Interval property to 1.

```
Private Sub cmdStart_Click()
   Timer1.Enabled = True   'Enables the Timer
End Sub
_____________________________________________________
Private Sub Timer1_Timer()
   Form1.Width = Form1.Width + 1
End Sub
```

Now run the program and click on Start button to start the Timer. Once the Timer is enabled, the code in the Timer procedure is executed every 1 millisecond and it increases the width of the current form in the above program example.

InputBox

InputBox is a function that prompts for user-input. InputBox shows a dialog box that inputs value from the user.

Syntax:

a=InputBox(promt, [Title], [Default], [xpos], [ypos])

where 'a' is a variable to which the value will be assigned. The texts inside the InputBox are optional except the "prompt" text. "prompt" text is the prompt message. "title" is the title of the message box window. "Default" is the default value given by the programmer. 'xpos' and 'ypos' are the geometric positions with respect to x and y axis respectively.

Note: Parameters in brackets are always optional. Do not write the brackets in your program.

Example:

```
Private Sub cmdTakeInput_Click()
    Dim n As Integer
    n = InputBox("Enter the value of n : ")
    Print n
End Sub
```

The above program prints the value of n taken from the InputBox function.

Example: InputBox with the title text and default value.

```
Private Sub cmdTakeInput_Click()
    Dim n As Integer
    n = InputBox("Enter the value of n : ", "Input", 5)
    Print n
End Sub
```

The InputBox dialog appears with the title "Input" and the highlighted default value in the provided text field is 5.

MsgBox

The MsgBox function shows a dialog box displaying the output value or a message.

Syntax:

MsgBox Prompt, [MsgBox style], [Title]

where Promt text is the prompt message, [MsgBox Style] is the msgbox style constants and [Title] is the text displayed in the title bar of the MsgBox dialog.

Example:

```
Private Sub cmdShowMessage_Click()
    Dim n As Integer
    n = 10
    MsgBox "The value of n is " & n
End Sub
```

Menu:

Windows applications provide groups of related commands in Menus. These commands depend on the application, but some-such as Open and Save are frequently found in applications.

Visual Basic provides an easy way to create menus with the modal Menu Editor dialog. The below dialog is displayed when the Menu Editor is selected in the Tool Menu. The Menu Editor command is grayed unless the form is visible. And also you can display the Menu Editor window by right clicking on the Form and selecting Menu Editor.

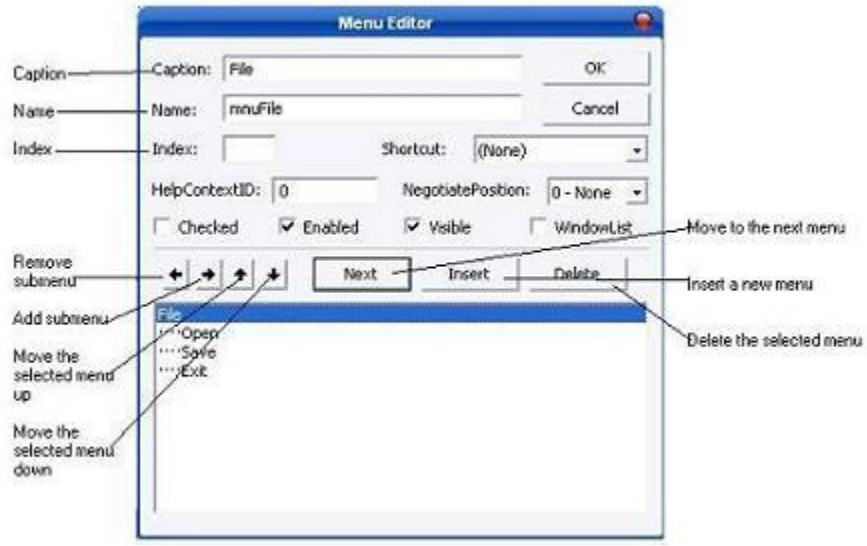
Basically, each menu item has a **Caption** property (possibly with an embedded & character to create an access key) and a **Name**. Each item also exposes three Boolean properties, Enabled, Visible, and Checked, which you can set both at design time and at run time.

Building a menu is a simple. You enter the item's Caption and Name, set other properties (or accept the default values for those properties), and press Enter to move to the next item. When you want to create a submenu, you press the Right Arrow button (or the Alt+R hot key). When you want to return to work on top-level menus—those items that appear in the menu bar when the application runs—you click the Left Arrow button (or press Alt+L). You can move items up and down in the hierarchy by clicking the corresponding buttons or the hot keys Alt+U and Alt+B, respectively.

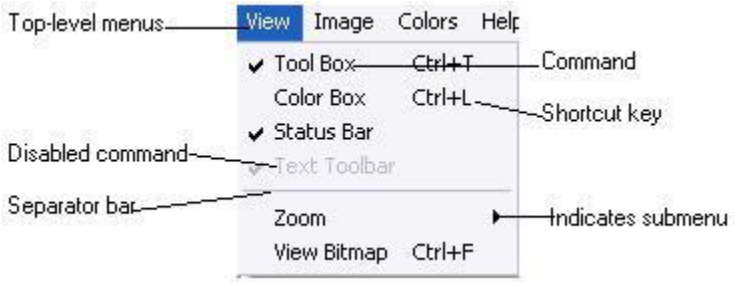
You can create up to five levels of submenus (six including the menu bar), which are too many even for the most patient user. If you find yourself working with more than three menu levels, think about trashing your specifications and redesigning your application from the ground up.

You can insert a separator bar using the hypen (-) character for the Caption property. But even these separator items must be assigned a unique value for the Name property, which is a real nuisance. If you forget to enter a menu item's Name, the Menu Editor complains when you decide to close it. The convention used in this book is that all menu names begin with the three letters mnu.

An expanded Menu Editor window.



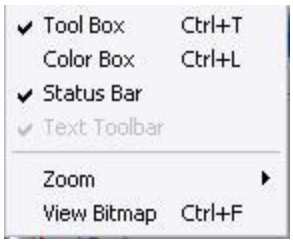
An expanded menu



The programmer can create menu control arrays. The Index TextBox specifies the menu's index in the control array.

The Menu Editor dialog also provides several CheckBoxes to control the appearance of the Menu.

Checked : This is unchecked by default and allows the programmer the option of creating a checked menu item(a menu item that act as a toggle and displays a check mark when selected. The following is a Check Menu items.



Enabled : specifies whether a menu is disabled or not. If you see a disabled command in a menu that means that feature is not available. The Visible checkbox specifies whether the menu is visible or not.

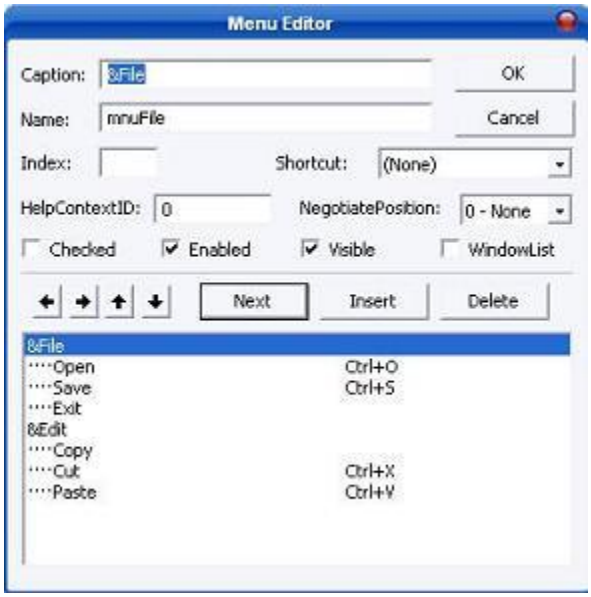
To add commands to the Form's menu bar, enter a caption and a name for each command. As soon as you start typing the command's caption, it also appears in a new line in the list at the bottom of the Menu Editor window. To add more commands click Enter and type the Caption and the Name.

Creating Menus

Open a new Project and save the form as menu.frm and save the project as menu.vbp.

Choose Tools ››› Menu Editor and type the menu items as shown below.

Caption	Name
File	mnuFile
Open	mnuOpen
Save	mnuSave
Exit	mnuExit
Edit	mnuEdit
Copy	mnuCopy
Cut	mnuCut
Paste	mnuPaste



Run the application by pressing F5.

Unit V

Form Events

This section is discussed about the form events in Visual Basic 6.

Initialize

The Initialize event is the first event of a form when the program runs. This event is raised even before the actual form window is created. You can use this event to initialize form’s properties.

Example:

```
Private Sub Form_Initialize()
   Text1.Text = ""
   Text2.Text = ""
End Sub
```

Load

After the Initialize event, Load event fires when the Form is loaded in the memory. This Form event is invoked for assigning values to a control's property and for initializing variables.

Note that the form is not visible yet. For this reason, you cannot invoke a graphic function like Cls, PSet, Point, Circle, Line etc and you cannot give the focus to any control with the SetFocus method in the form's Load event procedure. The Print command will not even work.

On the other hand, this won't be a problem setting a control's property in the form's load event.

Example:

```
Private Sub Form_Load()
   Text1.Text = ""
   Text2.Text = ""
   var1 = 0
End Sub
```

Resize

After the Load event, the form receives the Resize event. The form's Resize event is also raised when you resize the form either manually or programmatically.

Activate and Deactivate

After the Resize event, Activate event fires. While working with multiple forms, the Activate event fires when the form becomes an active form in the current application, and the Deactivate event fires when the other form becomes the active Form.

Another important form event is the Paint event which I'll not discuss here. Paint event will be covered later in another lesson.

Note that when you run your program, form events such as Initialize, Load, Resize, Activate and Paint events are raised automatically one by one. The Paint event is not raised when the form's AutoRedraw property is set to True.

QueryUnload

When you close or unload the form, the form first receives the QueryUnload event and then the Unload event.

The QueryUnload event is invoked when the form is about to be closed. When the form is closed, it may be unloaded by the user, the task manager, the code, owner form, MDI form or it may be closed when the current windows session is ending.

```
Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode _
 As Integer)
    'code here
End Sub
```

This event procedure takes two parameters, Cancel and UnloadMode. The Cancel parameter cancels the unload operation, and depending on the symbolic values of UnloadMode, a particular task can be performed.

Example:

```
Private Sub Form_QueryUnload(Cancel  _
As Integer, UnloadMode As Integer)
   If UnloadMode = vbFormControlMenu Then   'vbFormControlMenu=0
      MsgBox "the form is being closed by the user."
   End If
End Sub
```

The other constants are vbFormCode, vbAppWindows, vbAppTaskManager, vbFormMDIForm, vbFormOwner.

Symbolic constant values for the UnloadMode parameter are explained below:-

- **vbFormControlMenu:** when the form is closed by the user.
- **vbFormCode:** Use this constant when you're closing a form by code.
- **vbAppWindows:** when the current windows session is ending.
- **vbAppTaskManager:** when the task manager is closing the application.
- **vbFormMDIForm:** when the MDI parent is closing the form.
- **vbFormOwner:** when the owner form is closing.

Unload

The Unload event fires when the Form unloads. You can use this event to warn the user that data needs to be saved before closing the application.

Example: In this program, you cannot close the Form keeping the text field blank

```
Private Sub Form_Unload(Cancel As Integer)
   If Text1.Text = "" Then
      MsgBox "You cannot exit keeping the text field blank"
      Cancel = True
   End If
End Sub
```

When Cancel = True, you cannot close the Form. The Cancel parameter is used to cancel the form's unload operation.

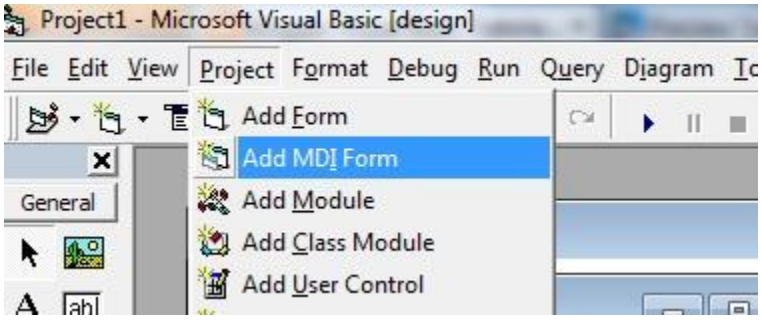
MDI Forms in VB6.0

This lesson discusses how to work with Visual Basic 6.0.

What is an MDI form?

MDI stands for Multiple Document Interface. When you want to handle multiple documents, MDI forms are useful in a Windows program.

How to add an MDI form to the current project?



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

Restrictions of the MDI form

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

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

How does the MDI form work?

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

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

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

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

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

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

Note that '(ActiveForm Is Nothing)' represents that there is no active form. The 'Not' keyword before it negates the value.

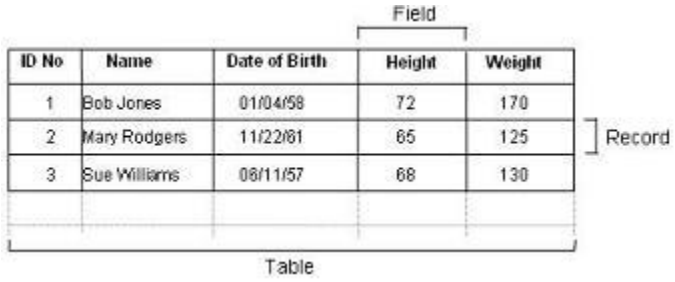
Data Access for Visual Basic 6.0

In simplest terms, a database is a collection of information. This collection is stored in well-defined tables, or matrices.

• The rows in a database table are used to describe similar items. The rows are referred to as database records. In general, no two rows in a database table will be alike.

• The columns in a database table provide characteristics of the records. These characteristics are called database fields. Each field contains one specific piece of information. In defining a database field, you specify the data type, assign a length, and describe other attributes.

• Here is a simple database example:



In this database table, each record represents a single individual. The fields (descriptors of the individuals) include an identification number (ID No), Name, Date of Birth, Height, and Weight.

Most databases use indexes to allow faster access to the information in the database. Indexes are sorted lists that point to a particular row in a table. In the example just seen, the ID No field could be used as an index.

A database using a single table is called a flat database. Most databases are made up of many tables. When using multiple tables within a database, these tables must have some common fields to allow cross-referencing of the tables. The referral of one table to another via a common field is called a relation. Such groupings of tables are called relational databases.

In our first example, we will use a sample database that comes with Visual Basic. This database (BIBLIO.MDB) is found in the main Visual Basic directory (try c:\Program Files\Microsoft Visual Studio\VB98). It is a database of books about computers.

In Visual Basic 6.0, data access is accomplished using ActiveX Data Objects (ADO). In Visual Basic 2008, data access is accomplished using ADO.NET, which is a part of the .NET Framework. There are a number of differences, both conceptually and in terms of tasks, between the two technologies.

In Visual Basic 6.0, there are two common methods of implementing data access in an application: at design time, by binding to an ADODC (ADO data control) or by using a Data Environment, or at run time by creating and interacting with Recordset objects programmatically.

In Visual Basic 6.0, data binding is accomplished by setting the binding-related properties of a control: DataChanged, DataField, DataFormat, DataMember, and DataSource. In most cases, the display property of a control (for example, the Text property of aTextBox control) is bound to a field in a data source.
The intrinsic Data control is geared toward MS-Access 97 and earlier databases, although a later VB service pack added connectivity for Access 2000 databases. We use the two samples Access databases provided with Visual Basic (BIBLIO.MDB and NWIND.MDB). These databases are provided in Access 97 format. On a default installation of VB6, these databases can be found in the folder: C:\Program Files\Microsoft Visual Studio\VB98.

The ADO (ActiveX Data Object) data control is the primary interface between a Visual Basic application and a database. It can be used without writing any code at all! Or, 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 ADO Data Control, then click OK. The control will be added to your toolbox.

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:

1. Connect to a database.
2. Open a specified database table.
3. Create a virtual table based on a database query.
4. Pass database fields to other Visual Basic tools, for display or editing. Such tools are bound tools (controls), or data aware.
5. Add new records or update a database.
6. Trap any errors that may occur while accessing data.

7. Close the database.

• Data Control Properties:
Align Determines where data control is displayed.

Caption Phrase displayed on the data control.

ConnectionString Contains the information used to establish a connection to a database.

LockType indicates the type of locks placed on records during editing (default setting makes databases read-only).

Recordset A set of records defined by a data control's ConnectionString and RecordSource properties. Run-time only.

RecordSource Determines the table (or virtual table) the data control is attached to.

• 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.
• 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.

After placing a data control on a form, you set the ConnectionStringproperty. The ADO data control can connect to a variety of database types. There are three ways to connect to a database: using a data link, using an ODBC data source, or using a connection string. In this lesson, we will look only at connection to a Microsoft Access database using a data link. A data link is a file with a UDL extension that contains information on database type.

• If your database does not have a data link, you need to create one. This process is best illustrated by example. We will be using

the BIBLIO.MDBdatabase in our first example, so these steps show you how to create its data link:

1. Open Windows Explorer.
2. Open the folder where you will store your data link file.
3. Right-click the right side of Explorer and choose New. From the list of files, select Microsoft Data Link.
4. Rename the newly created file BIBLIO.UDL
5. Right-click this new UDL file and click Properties.
6. Choose the Provider tab and select Microsoft Jet 3.51 OLE DB Provider (an Access database).
7. Click the Next button to go to the Connection tab.
8. Click the ellipsis and use the Select Access Database dialog box to choose the BIBLIO.MDB file which is in the Visual Basic main folder. Click Open.
9. Click Test Connection. Then, click OK (assuming it passed). The UDL file is now created and can be assigned toConnectionString, using the steps below.

• If a data link has been created and exists for your database, click the ellipsis that appears next to the ConnectionString property. Choose Use Data Link File. Then, click Browse and find the file. Click Open. The data link is now assigned to the property. Click OK.

• Once the ADO data control is connected to a database, we need to assign a table to that control. Recall each data control is attached to a single table, whether it is a table inherent to the database or the virtual table we discussed. Assigning a table is done via the RecordSource property.

Most of the Visual Basic tools we've studied can be used as bound, or data-aware, tools (or controls). That means, certain tool properties can be tied to a particular database field. To use a bound control, one or more data controls must be on the form.

• Some bound data tools are:

Label - Can be used to provide display-only access to a specified text data field.

Text Box - Can be used to provide read/write access to a specified text data field. Probably, the most widely used data bound tool.

Check Box - Used to provide read/write access to a Boolean field.

Combo Box - Can be used to provide read/write access to a text data field.

List Box - Can be used to provide read/write access to a text data field.

Picture Box - Used to display a graphical image from a bitmap, icon, or metafile on your form. Provides read/write access to a image/binary data field.

Image Box - Used to display a graphical image from a bitmap, icon, or metafile on your form (uses fewer resources than a picture box). Provides read/write access to a image/binary data field.

• There are also three 'custom' data aware tools, the DataCombo (better than using the bound combo box), DataList (better than the bound list box), and DataGrid tools, we will look at later.

• Bound Tool Properties:

DataChanged - Indicates whether a value displayed in a bound control has changed.

DataField - Specifies the name of a field in the table pointed to by the respective data control.

DataSource - Specifies which data control the control is bound to.

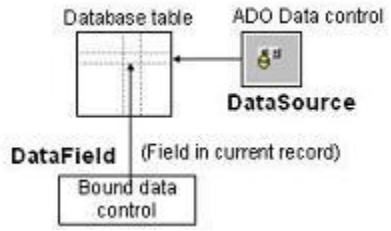
If the data in a data-aware control is changed and then the user changes focus to another control or tool, the database will automatically be updated with the new data (assuming LockType is set to allow an update).

• To make using bound controls easy, follow these steps (in order listed) in placing the controls on a form:

1. Draw the bound control on the same form as the data control to which it will be bound.
2. Set the DataSource property. Click on the drop-down arrow to list the data controls on your form. Choose one.
3. Set the DataField property. Click on the drop-down arrow to list the fields associated with the selected data control records. Make your choice.
4. Set all other properties, as required.

By following these steps in order, we avoid potential data access errors.

• The relationships between the bound data control and the data control are:



Connecting to an Access Database Using the VB Data Control

EXERCISE 1

STEPS:
1. Open a new Visual Basic project.
2. Put a data control (an intrinsic control, located in the VB toolbox) on the form and set the properties as follows:

Property	Value
(Name)	datAuthors
Caption	Use the arrows to view the data
Connect	Access (default)
DatabaseName	..\biblio.mdb
DefaultType	UseJet (default)
RecordSource	Authors (choose from list)

Notes: When you use the Data Control in a project, the properties that must be set are Database Name and Record Source, in that order. DatabaseName is the name of the database you want to use, and the RecordSource is the name of the table in that database that you want to use.

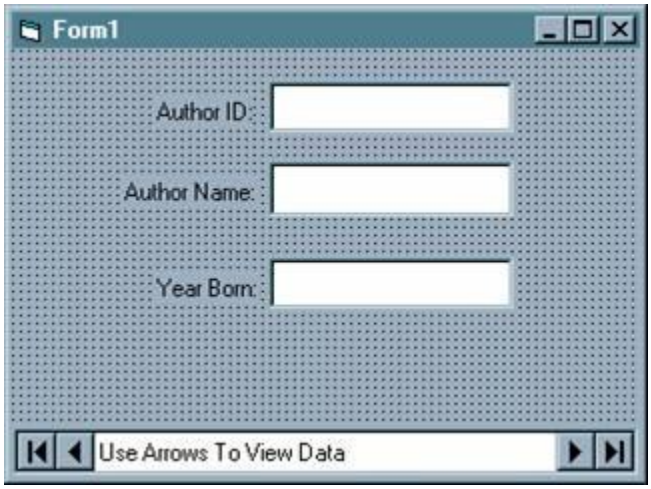
3. In your form, create a text box for each field in the Authors table, with labels. (If you were to open the database in Access, you would see that the three fields of the Authors table are Au_ID, Author, and Year Born.) Set the properties of the three textboxes as follows:

Name	DataSource	DataField
txtAuthID	datAuthors	Au_ID
txtAuthor	datAuthors	Author
txtYearBorn	datAuthors	Year Born

In addition, set the Enabled property of txtAuthID to False.

When you want a control (such as a text box) to display data from a database, the properties that must be set are DataSource and Datafield. The DataSource is the name of the data control on the form (it should already be configured), and the DataField is the name of the particular field in the database that should be displayed in the control (this field will be in the table that was chosen for the RecordSource of the data control).

At this point, your form should resemble the screen-shot below:



4. Save and run the project. Use the arrows on the data control to scroll through the data.

5. On any record, change the data in the author name or year born field. Move ahead, then move back to the record you changed. Note that your changes remain in effect. The data control automatically updates a record when you move off of the record.

EXERCISE 2

Using Navigation Buttons with a Data Control

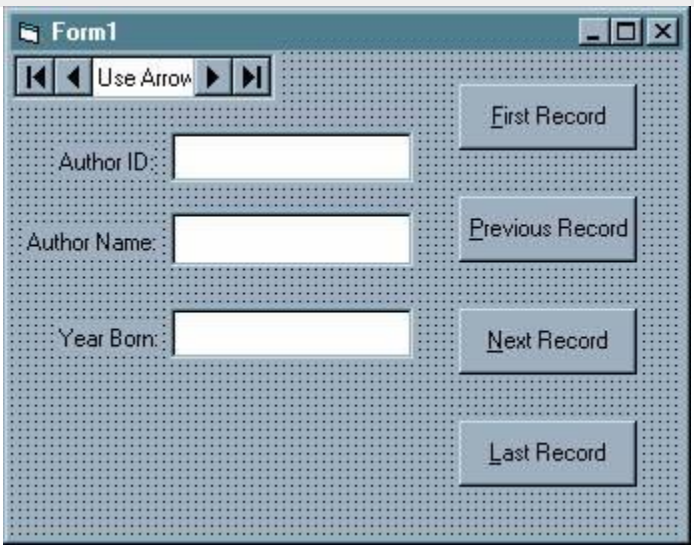
In the previous exercise, you saw that by clicking specific buttons of the data control, you could move to the first, previous, next, or last record. What is happening is that the data control is automatically invoking specific methods of the recordset object: namely the MoveFirst,MovePrevious, MoveNext, and MoveLast methods. You can also invoke these methods through code, which is what this exercise demonstrates.
STEPS:

1. Copy the files from Exercise 1 into a new folder and open the VBP file in the new folder.
2. Set the Visible property of the data control (datAuthors) to False.

3. Make four command buttons with the following properties:

Name	Caption
cmdMoveNext	Next Record
cmdMoveLast	Last Record
cmdMovePrevious	Previous Record
cmdMoveFirst	First Record

At this point, your form should resemble the screen-shot below:



4. Put the following four lines of code in the appropriate Click events for the buttons:

Event	Code
cmdMoveNext_Click	datAuthors.Recordset.MoveNext
cmdMoveLast_Click	datAuthors.Recordset.MoveLast
cmdMovePrevious_Click	datAuthors.Recordset.MovePrevious
cmdMoveFirst_Click	datAuthors.Recordset.MoveFirst

5. Save and run your program.

6. Move to the last record and then click the Move Next button twice.

EXERCISE 3

Using the EOF and BOF Properties with Navigation Buttons

STEPS:

1. Copy the files from Exercise #2 into a new folder and open the VBP file in the new folder.

2. When the user clicks on the MoveNext button, and there is no next record, your code should stay on the same record (the last one).

Put the following code in the cmdMoveNext_Click() event:

```
datAuthors.Recordset.MoveNext
If datAuthors.Recordset.EOF = True Then
datAuthors.Recordset.MoveLast
End If
```

FYI: Instead of Recordset.MoveLast, you could use MoveFirst to let the user loop around to the first record.

3. Put similar code in the cmdMovePrevious_Click() event. In this case, you will be checking forRecordset.BOF = True.

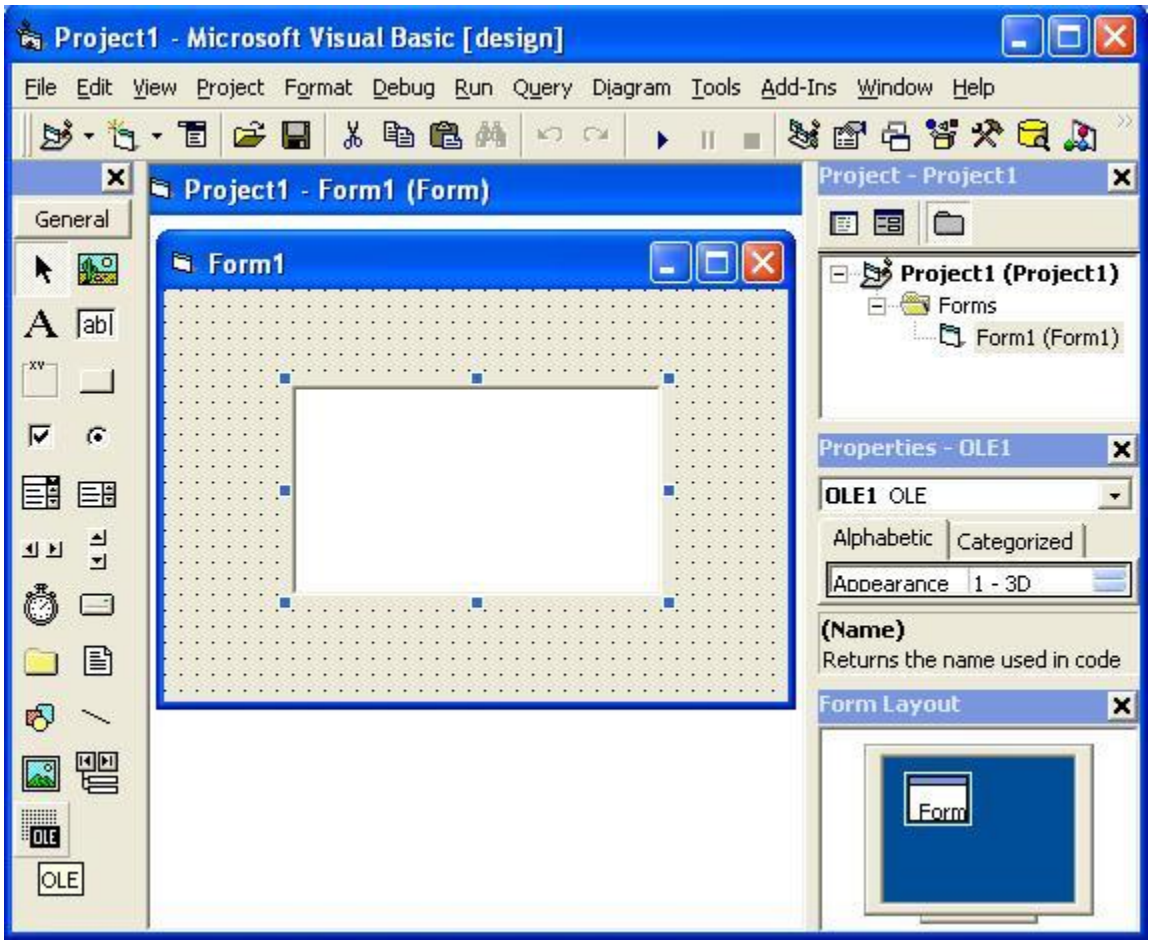
3. Save and run the project and test it thoroughly.

OLE (Object Linking and Embedding)

OLE (Object Linking and Embedding) is a means to interchange data between applications. One of the primary uses of the**OLE Container** control was to embed Word documents or Excel spreadsheets in a form. This is the control to use when you want to link or embed an object into your Visual Basic application.

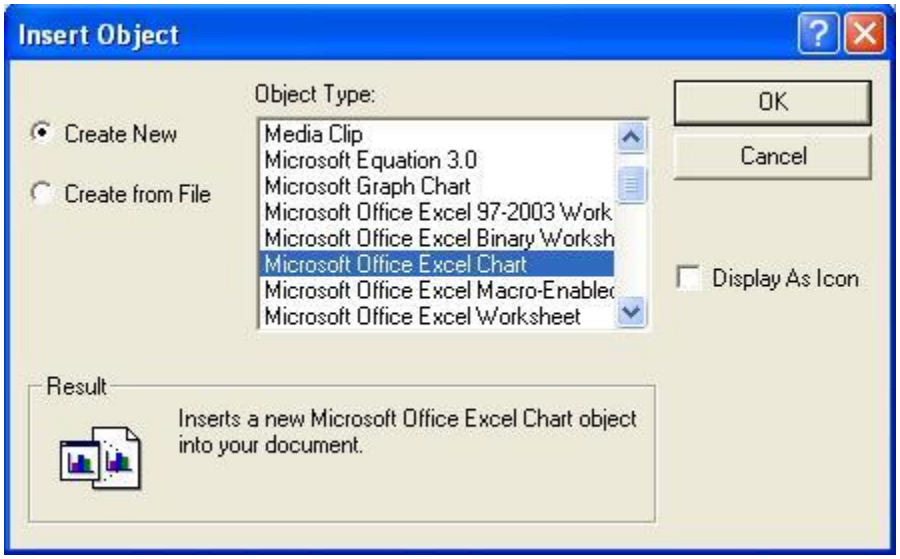
1. From the **Windows Start** menu, choose **Programs**, **Microsoft Visual Studio 6.0**, and then **Microsoft Visual Basic 6.0**.

2.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 dialogbox 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.



6. Creating a Excel chart will allow you to access the Excel application to define the chart.

7. Select the Excel chart.

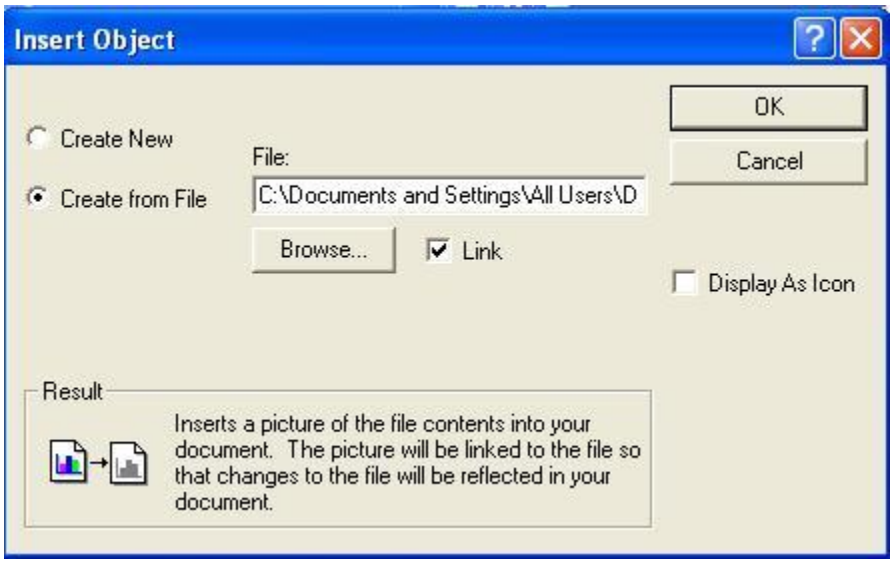
8.Run your application. Double click on the OLE control and use it in your project as you want.

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.



4. This Word document page has been sized so that it will fit onto a form. Change the SizeMode 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.