
ABOUT VISUAL BASIC

Visual Basic (VB) was developed from the BASIC programming language. In the 1970s, Microsoft started developing ROM-based interpreted BASIC for the early microprocessor-based computers. In 1982, Microsoft QuickBasic revolutionized Basic and was legitimized as a serious development language for MS-DOS environment. Later on, Microsoft Corporation created the enhanced version of BASIC called Visual Basic for Windows.

ABOUT VISUAL BASIC

Visual Basic (VB) is an event-driven programming language.

This is called because programming is done in a graphical environment unlike the previous version BASIC where programming is done in a text only environment and executed sequentially in order to control the user interface. Visual Basic enables the user to design the user interface quickly by drawing and arranging the user elements. Due to this spent time is saved for the repetitive task.

Important Features of Visual Basic (VB)

- Full set of objects - you 'draw' the application
 - Lots of icons and pictures for your use
 - Response to mouse and keyboard actions
 - Clipboard and printer access
 - Full array of mathematical, string handling, and graphics functions
 - Can handle fixed and dynamic variable and control arrays
 - Sequential and random access file support
 - Useful debugger and error-handling facilities
 - Powerful database access tools
 - ActiveX support
 - Package & Deployment Wizard makes distributing your applications simple
-

The Integrated Development Environment

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

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

- Menu Bar
 - Tool Bar
 - Project Explorer
 - Properties window
 - Form Layout Window
 - Toolbox
 - Form Designer
 - Object Browser
-

New Project

Microsoft Visual Basic

New | Existing | Recent

 Standard EXE	 ActiveX EXE	 ActiveX DLL	 ActiveX Control	 VB Application Wizard
 VB Wizard Manager	 ActiveX Document DLL	 ActiveX Document EXE	 Addin	 Data Project

VB Enterprise
Edition
Controls

Open

Cancel

Help

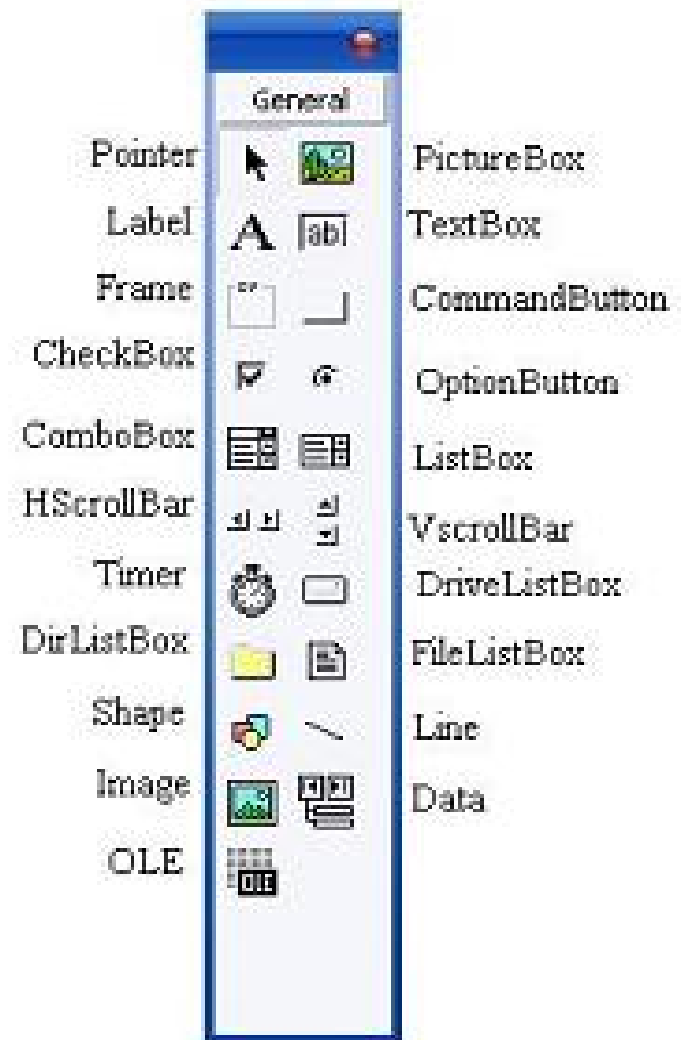
Don't show this dialog in the future

Menu Bar

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

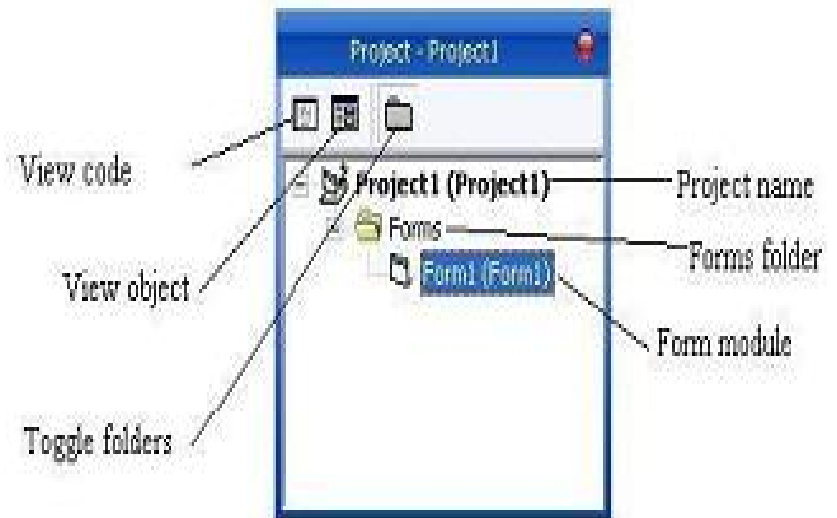
Toolbox

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



Project Explorer

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



Properties Window

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

Object Browser

- The Object Browser allows us to browse through the various properties, events and methods that are made available to us. It is accessed by selecting Object Browser from the View menu or pressing the key F2. The left column of the Object Browser lists the objects and classes that are available in the projects that are opened and the controls that have been referenced in them. It is possible for us to scroll through the list and select the object or class that we wish to inspect. After an object is picked up from the Classes list, we can see its members (properties, methods and events) in the right column.
 - A property is represented by a small icon that has a hand holding a piece of paper. Methods are denoted by little green blocks, while events are denoted by yellow lightning bolt icon.
-

Visual Basic 6 (VB6) Data Types, Modules and Operators

Visual Basic uses building blocks such as Variables, Data Types, Procedures, Functions and Control Structures in its programming environment. This section concentrates on the programming fundamentals of Visual Basic with the blocks specified.

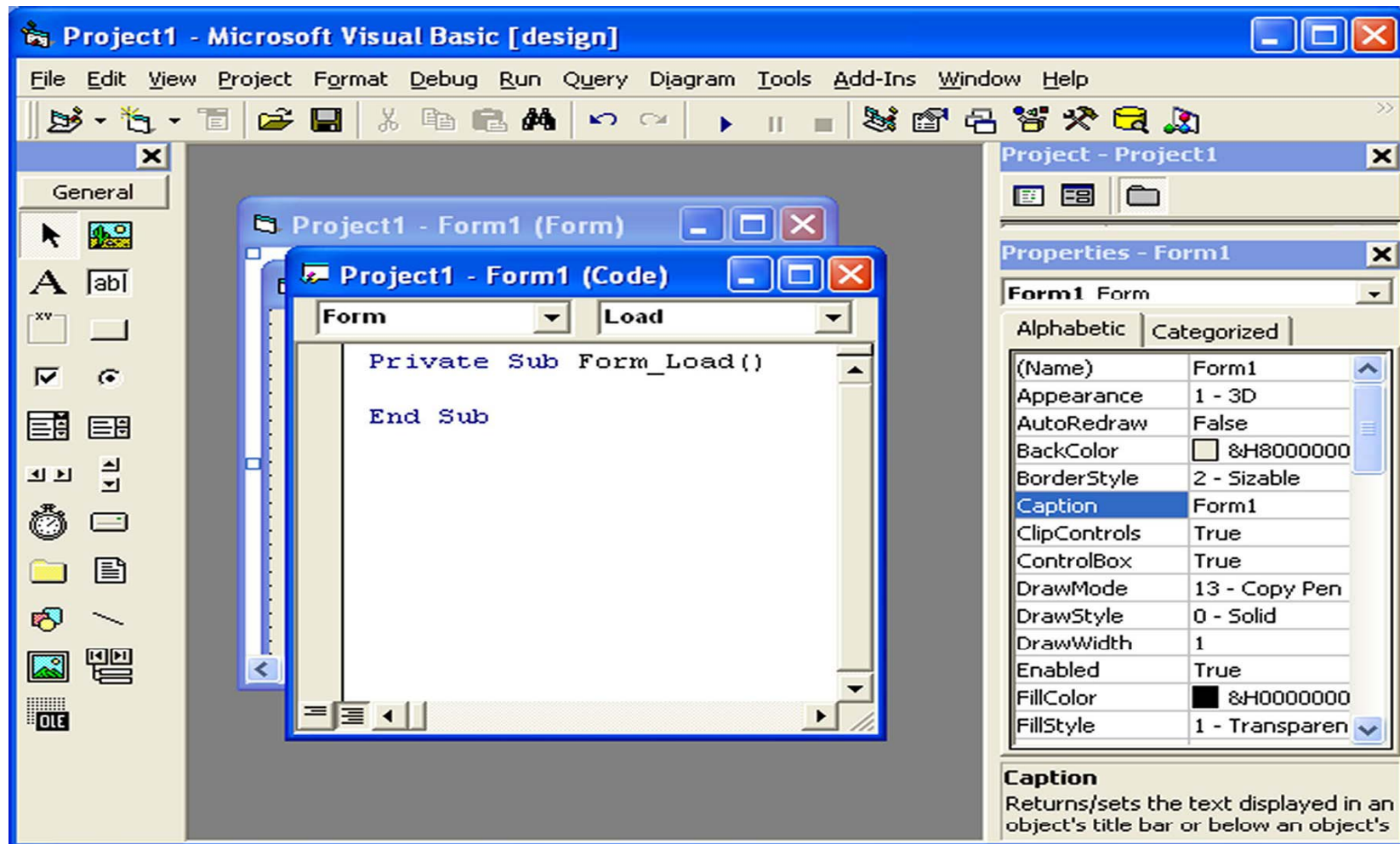
Modules

- Code in Visual Basic is stored in the form of modules. The three kind of modules are Form Modules, Standard Modules and Class Modules. A simple application may contain a single Form, and the code resides in that Form module itself. As the application grows, additional Forms are added and there may be a common code to be executed in several Forms. To avoid the duplication of code, a separate module containing a procedure is created that implements the common code. This is a standard Module.
 - Class module (.CLS filename extension) are the foundation of the object oriented programming in Visual Basic. New objects can be created by writing code in class modules. Each module can contain:
-

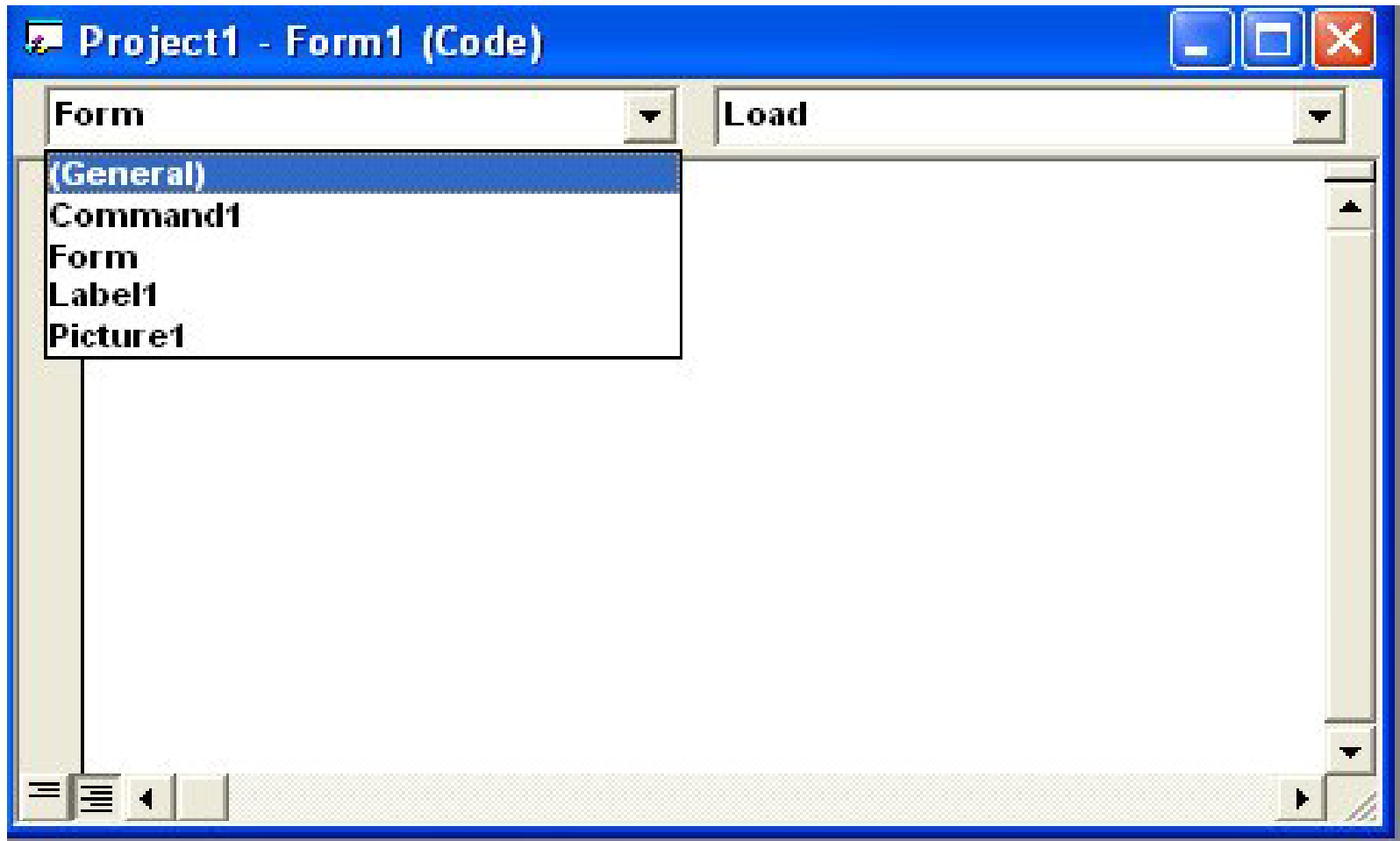
Declarations : May include constant, type, variable and DLL procedure declarations.

- **Procedures** : A sub function, or property procedure that contain pieces of code that can be executed as a unit.
 - These are the rules to follow when naming elements in VB - variables, constants, controls, procedures, and so on:
 - A name must begin with a letter.
 - May be as much as 255 characters long (but don't forget that somebody has to type the stuff!).
 - Must not contain a space or an embedded period or type-declaration characters used to specify a data type; these are ! # % \$ & @
 - Must not be a reserved word (that is part of the code, like Option, for example)
 - The dash, although legal, should be avoided because it may be confused with the minus sign. Instead of First-name use First_name or FirstName.
-

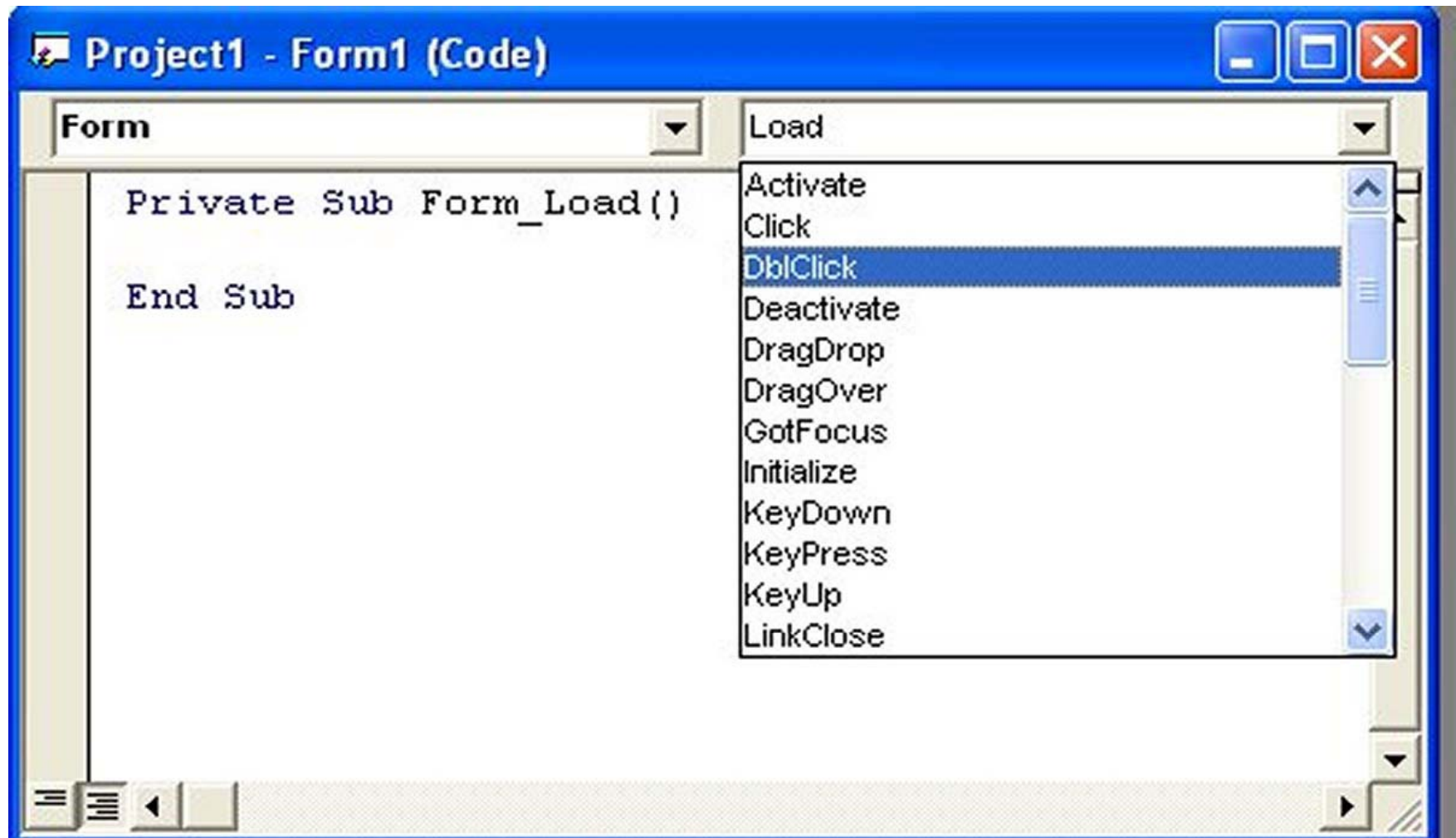
2.1 Creating Your First Application



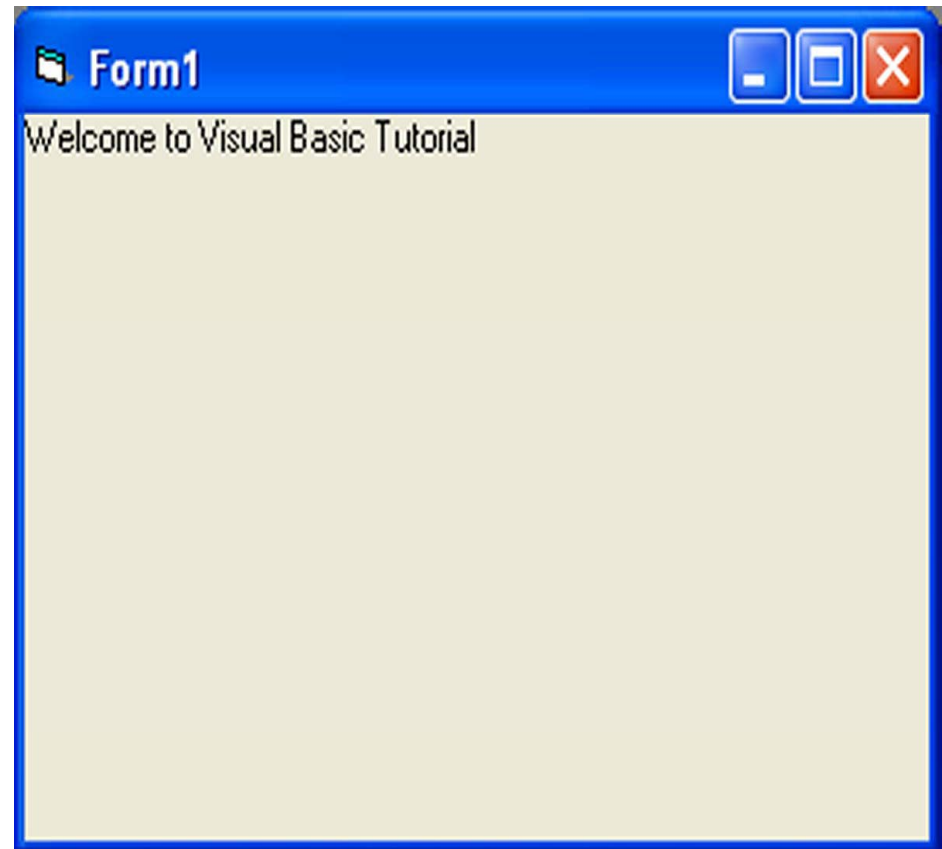
2.2: List of Objects



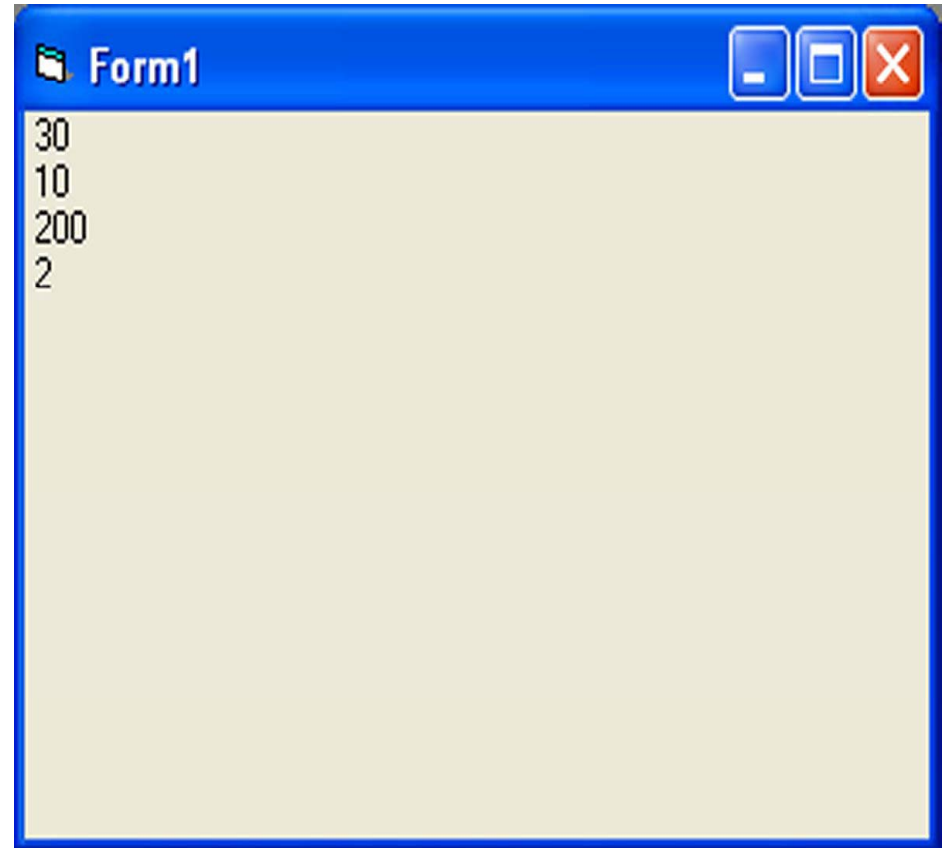
List of Procedures



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



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



You can also use the + or the & operator to join two or more texts (string) together like in example 2.1.4 (a) and (b)

Example 2.1.4(a)

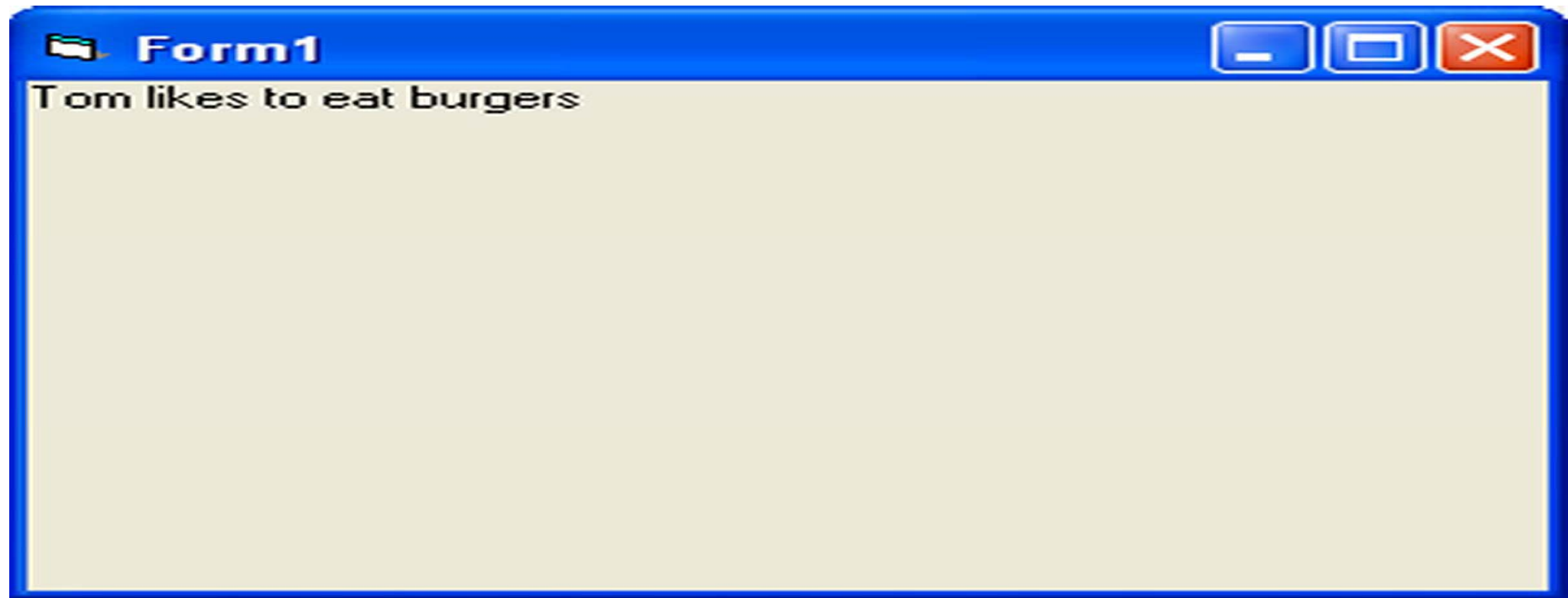
```
Private Sub  
A = Tom  
B = "likes"  
C = "to"  
D = "eat"  
E = "burger"  
Print A + B + C + D + E  
End Sub
```

Example

```
Private Sub  
A = Tom  
B = "likes"  
C = "to"  
D = "eat"  
E = "burger"  
Print A & B & C & D & E  
End Sub
```

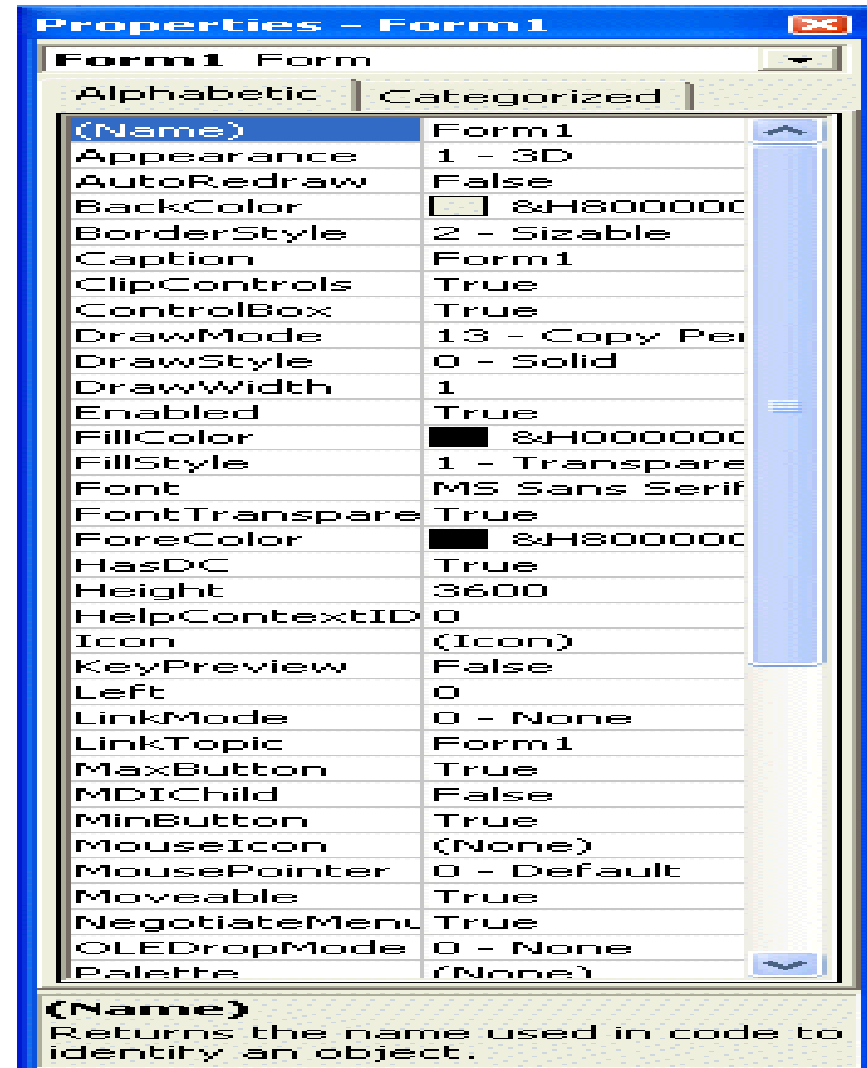
2.1.4(b)

The Output of Example



Working With Controls

The Control Properties



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

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

Another example is to change the control Shape to a particular shape at runtime by writing the following code. This code will change the shape to a circle at runtime.

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

Handling some of the common controls

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

```
Private Sub Command1_Click()  
    'To add the values in text box 1 and text box 2  
    Sum = Val(Text1.Text) + Val(Text2.Text)  
    'To display the answer on label 1  
    Label1.Caption = Sum  
End Sub
```

The screenshot shows a Windows form titled "Form1" with a blue title bar and standard window controls (minimize, maximize, close). The form has a light beige background. It contains three text boxes and one button. The first text box is labeled "Number 1" and contains the value "100". The second text box is labeled "Number 2" and contains the value "500". The third text box is labeled "Sum" and contains the value "600". To the right of the "Sum" text box is a button labeled "Calculate".

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

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

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

```
Private Sub Form_Load ( )
```

```
List1.AddItem "Lesson1"
```

```
List1.AddItem "Lesson2"
```

```
List1.AddItem "Lesson3"
```

```
List1.AddItem "Lesson4"
```

```
End Sub
```

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

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

```
Private Sub Form_Load ( )
```

```
Combo1.AddItem "Item1"
```

```
Combo1.AddItem "Item2"
```

```
Combo1.AddItem "Item3"
```

```
Combo1.AddItem "Item4"
```

```
End Sub
```

The Check Box The Check Box control lets the user select or unselect an option. When the Check Box is checked, its value is set to 1 and when it is unchecked, the value is set to 0. You can include the statements `Check1.Value=1` to mark the Check Box and `Check1.Value=0` to unmark the Check Box, as well as use them to initiate certain actions. For example, the program will change the background color of the form to red when the check box is unchecked and it will change to blue when the check box is checked. You will learn about the conditional statement `If....Then....Elesif` in later lesson. `VbRed` and `vbBlue` are color constants and `BackColor` is the background color property of the form.

The Option Box

The Option Box control also lets the user select one of the choices. However, two or more Option Boxes must work together because as one of the Option Boxes is selected, the other Option Boxes will be unselected. In fact, only one Option Box can be selected at one time. When an option box is selected, its value is set to “True” and when it is unselected; its value is set to “False”. In the following example, the shape control is placed in the form together with six Option Boxes. When the user clicks on different option boxes, different shapes will appear. The values of the shape control are 0, 1, and 2,3,4,5 which will make it appear as a rectangle, a square, an oval shape, a rounded rectangle and a rounded square respectively.

```
Private Sub Option1_Click ( )  
Shape1.Shape = 0  
End Sub
```

```
Private Sub Option2_Click()  
Shape1.Shape = 1  
End Sub
```

```
Private Sub Option3_Click()  
Shape1.Shape = 2  
End Sub
```

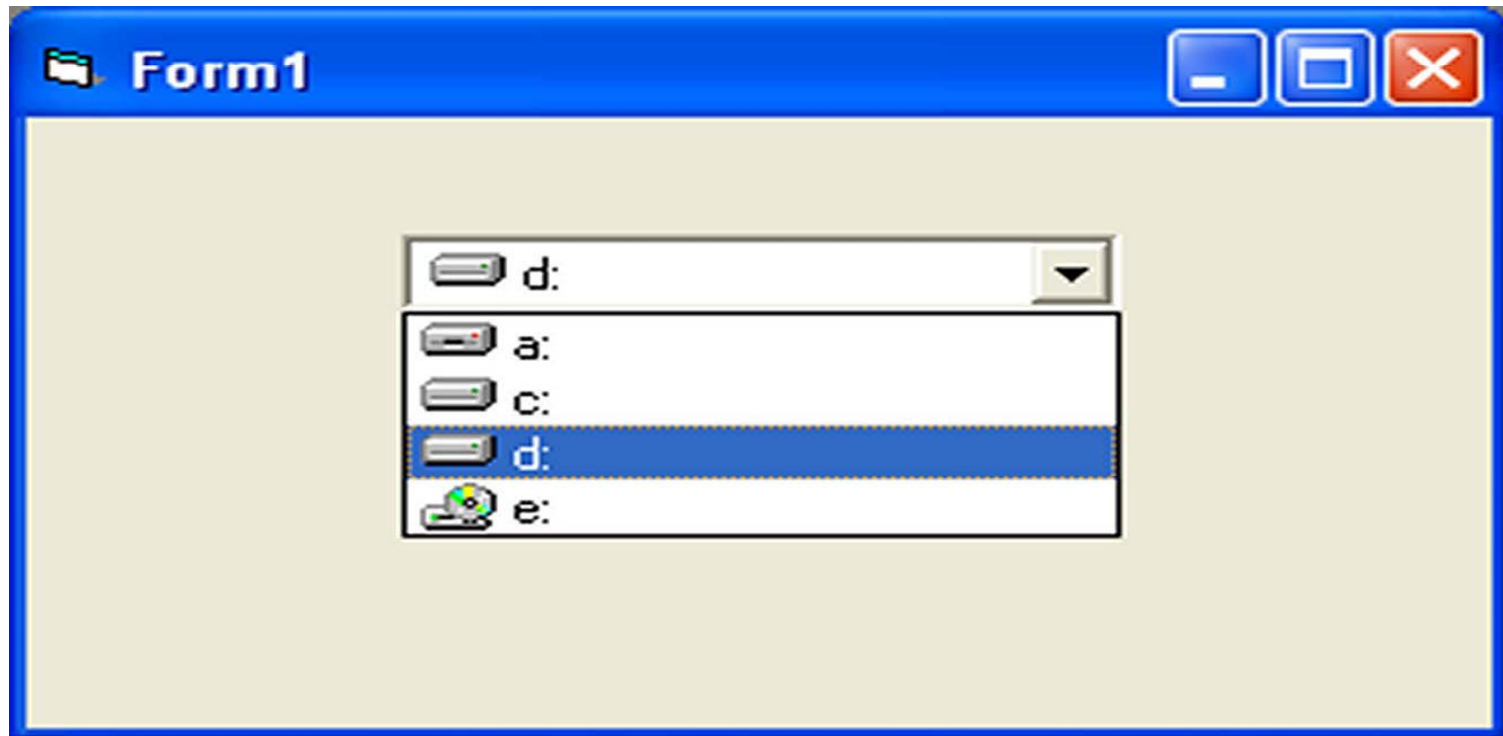
```
Private Sub Option4_Click()  
Shape1.Shape = 3  
End Sub
```

```
Private Sub Option5_Click()  
Shape1.Shape = 4  
End Sub
```

```
Private Sub Option6_Click()  
Shape1.Shape = 5  
End Sub
```

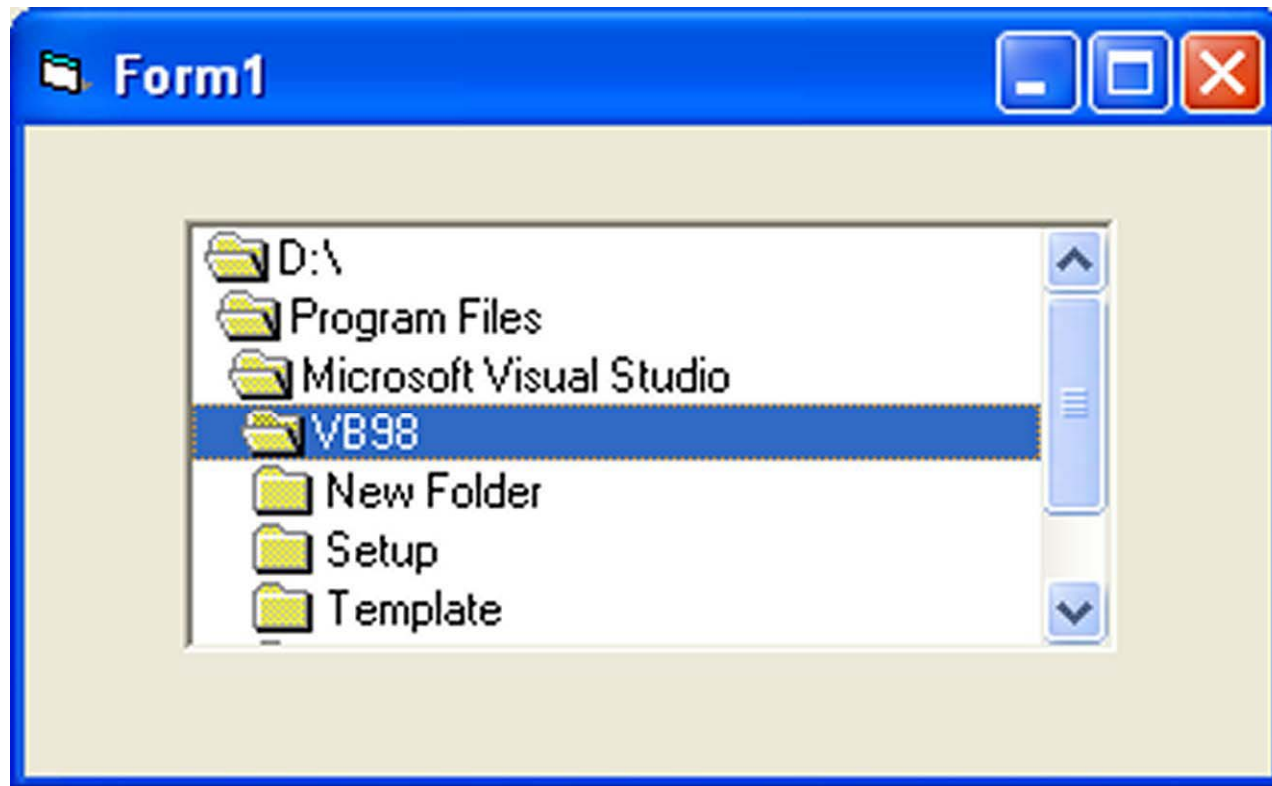
The Drive List Box

The Drive ListBox is for displaying a list of drives available in your computer. When you place this control into the form and run the program



The Directory List Box

The Directory List Box is for displaying the list of directories or folders in a selected drive. When you place this control into the form and run the program



Data types in Visual Basic 6

By default Visual Basic variables are of variant data types. The variant data type can store numeric, date/time or string data. When a variable is declared, a data type is supplied for it that determines the kind of data they can store. The fundamental data types in Visual Basic including variant are integer, long, single, double, string, currency, byte and boolean. Visual Basic supports a vast array of data types. Each data type has limits to the kind of information and the minimum and maximum values it can hold. In addition, some types can interchange with some other types.

1. Numeric

Byte	Store integer values in the range of 0 - 255
Integer	Store integer values in the range of (-32,768) - (+ 32,767)
Long	Store integer values in the range of (-2,147,483,468) - (+ 2,147,483,468)
Single	Store floating point value in the range of (-3.4x10 ⁻³⁸) - (+ 3.4x10 ³⁸)
Double	Store large floating value which exceeding the single data type value
Currency	store monetary values. It supports 4 digits to the right of decimal point and 15 digits to the left

2. String

Use to store alphanumeric values. A variable length string can store approximately 4 billion characters

3. Date

Use to store date and time values. A variable declared as date type can store both date and time values and it can store date values 01/01/0100 up to 12/31/9999

4. Boolean

Boolean data types hold either a true or false value. These are not stored as numeric values and cannot be used as such. Values are internally stored as -1 (True) and 0 (False) and any non-zero value is considered as true.

5. Variant

Stores any type of data and is the default Visual Basic data type. In Visual Basic if we declare a variable without any data type by default the data type is assigned as default.

Operators in Visual Basic

Arithmetical Operators

Operators	Description	Example	Result
+	Add	5+5	10
-	Substract	10-5	5
/	Divide	25/5	5
\	Integer Division	20\3	6
*	Multiply	5*4	20
^	Exponent (power of)	3^3	27
Mod	Remainder of division	20 Mod 6	2
&	String concatenation	"George"&" "&"Bush"	"George Bush"

Declaring Variables

In Visual Basic, one needs to declare the variables before using them by assigning names and data types. They are normally declared in the general section of the codes' windows using the **Dim** statement.

The format is as follows:

Dim Variable Name As Data Type

Dim password As String

Dim yourName As String

Dim firstnum As Integer

Dim secondnum As Integer

Dim total As Integer

Dim doDate As Date

Working with Variables

After declaring various variables using the Dim statements, we can assign values to those variables. The general format of an assignment is

Variable=Expression

The variable can be a declared variable or a control property value. The expression could be a mathematical expression, a number, a string, a Boolean value (true or false) and more. The following are some examples:

```
firstNumber=100
secondNumber=firstNumber-99
userName="John Lyan"
userpass.Text = password
Label1.Visible = True
Command1.Visible = false
Label4.Caption = textbox1.Text
ThirdNumber = Val(usernum1.Text)
total = firstNumber + secondNumber+ThirdNumber
```

```
Dim firstName As String
Dim secondName As String
Dim yourName As String
```

```
Private Sub Command1_Click()
firstName = Text1.Text
secondName = Text2.Text
yourName = secondName + " " + firstName
    Label1.Caption = yourName
End Sub
```

In this example, three variables are declared as string. For variables `firstName` and `secondName` will receive their data from the user's input into `textbox1` and `textbox2`, and the variable `yourName` will be assigned the data by combining the first two variables. Finally, `yourName` is displayed on `Label1`.

```
Dim number1, number2, number3 as Integer
Dim total, average as variant
Private sub Form_Click
number1=val(Text1.Text)
number2=val(Text2.Text)
number3= val(Text3.Text)
Total=number1+number2+number3
Average=Total/5
Label1.Caption=Total
Label2.Caption=Average
End Sub
```

In the example above, three variables are declared as integer and two variables are declared as variant. Variant means the variable can hold any data type. The program computes the total and average of the three numbers that are entered into three text boxes.

Controlling Program Flow

In this chapter, you will learn how to create VB code that can make decision when it process input from the user, and control the program flow in the process. Decision making process is an important part of programming because it can help to solve practical problems intelligently so that it can provide useful output or feedback to the user. For example, we can write a program that can ask the computer to perform certain task until a certain condition is met.

Conditional Operators

To control the VB program flow, we can use various conditional operators. Basically, they resemble mathematical operators. Conditional operators are very powerful tools, they let the VB program compare data values and then decide what action to take, whether to execute a program or terminate the program and more.

Operators	Description	Example	Result
>	Greater than	10>8	True
<	Less than	10<8	False
>=	Greater than or equal to	20>=10	True
<=	Less than or equal to	10<=20	True
<>	Not Equal to	5<>4	True
=	Equal to	5=7	False

Logical Operators

In addition to conditional operators, there are a few logical operators which offer added power to the VB programs.

Operator	Meaning
And	Both sides must be true
or	One side or other must be true
Xor	One side or other must be true but not both
Not	Negates truth

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

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

The general format for the **if...then...else** statement is

If conditions **Then**

VB expressions

Else

VB expressions

End If

* any If..Then..Else statement must end with End If.

Sometime it is not necessary to use Else.

```
Private Sub OK_Click()  
firstnum = Val(usernum1.Text)  
secondnum = Val(usernum2.Text)  
total = Val(sum.Text)  
If total = firstnum + secondnum And Val(sum.Text)  
<> 0 Then  
correct.Visible = True  
wrong.Visible = False  
Else  
correct.Visible = False  
wrong.Visible = True  
End If  
End Sub
```

Select Case...End select Control Structure

In the previous lesson, we have learned how to control the program flow using the **If...Elseif** control structure. In this chapter, you will learn another way to control the program flow, that is, the **Select Case** control structure. However, the Select Case control structure is slightly different from the If...Elseif control structure. The difference is that the Select Case control structure basically only make decision on one expression or dimension (for example the examination grade) while the If ...Elseif statement control structure may evaluate only one expression, each If....Elseif statement may also compute entirely different dimensions. Select Case is preferred when there exist many different conditions because using If...Then..Elseif statements might become too messy.

The format of the Select Case control structure is show below:

Select Case expression

Case value1

Block of one or more VB statements

Case value2

Block of one or more VB Statements

Case value3

Block of one or more VB statements

Case value4

.

.

.

Case Else

Block of one or more VB Statements

End Select

* The data type specified in expression must match that of Case values.

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

Dim mark As Single

Private Sub Compute_Click()
'Examination Marks
mark = mrk.Text

Select Case mark
Case Is >= 85
 comment.Caption = "Excellence"
Case Is >= 70
 comment.Caption = "Good"
Case Is >= 60
 comment.Caption = "Above
Average"
Case Is >= 50
 comment.Caption = "Average"
Case Else
 comment.Caption = "Need to work
harder"
End Select
End Sub

Looping

Visual Basic allows a procedure to be repeated as many times as long as the processor could support. This is generally called looping .

9.1 Do Loop

The formats are

- a) Do While condition
 Block of one or more VB statements
Loop
 - b) Do
 Block of one or more VB statements
Loop While condition
 - c) Do Until condition
 Block of one or more VB statements
Loop
 - d) Do
 Block of one or more VB statements
Loop Until condition
-

Do while counter <=1000

num.Text=counter

counter =counter+1

Loop

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

The above example can be rewritten as

Do

num.Text=counter

counter=counter+1

Loop until counter>1000

Exiting the Loop

Sometime we need exit to exit a loop prematurely because of a certain condition is fulfilled. The syntax to use is known as Exit Do.

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

Explanation

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

For....Next Loop

The format is:

For counter=startNumber to endNumber (Step increment)

One or more VB statements

Next

```
For counter=1 to 10
display.Text=counter
Next
```

```
For counter=1 to 1000 step 10
counter=counter+1
Next
```

```
For counter=1000 to 5 step -5
counter=counter-10
Next
```

*Notice that increment can be negative

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

Introduction to VB Built-in Functions

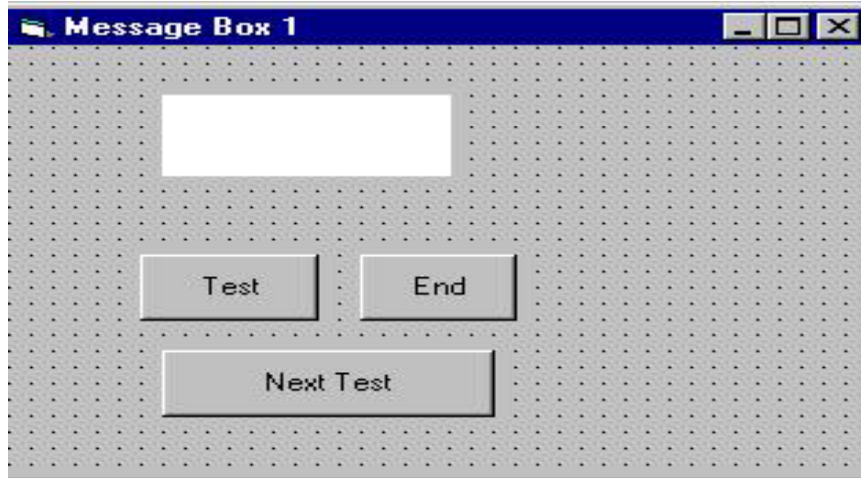
MsgBox () Function

The objective of MsgBox is to produce a pop-up message box and prompt the user to click on a command button before he /she can continues. This format is as follows:

yourMsg=MsgBox(Prompt, Style Value, Title)

The first argument, Prompt, will display the message in the message box. The Style Value will determine what type of command buttons appear on the message box, please refer Table 10.1 for types of command button displayed. The Title argument will display the title of the message board.

Style Value	Named Constant	Buttons Displayed
0	vbOkOnly	Ok button
1	vbOkCancel	Ok and Cancel buttons
2	vbAbortRetryIgnore	Abort, Retry and Ignore buttons.
3	vbYesNoCancel	Yes, No and Cancel buttons
4	vbYesNo	Yes and No buttons
5	vbRetryCancel	Retry and Cancel buttons



The procedure for the test button:

```
Private Sub Test_Click()  
    Dim testmsg As Integer  
    testmsg = MsgBox("Click to test", 1,  
        "Test message")  
    If testmsg = 1 Then  
        Display.Caption = "Testing  
            Successful"  
    Else  
        Display.Caption = "Testing fail"  
    End If  
End Sub
```



The InputBox() Function

An InputBox() function will display a message box where the user can enter a value or a message in the form of text. The format is

myMessage=InputBox(Prompt, Title, default_text, x-position, y-position)

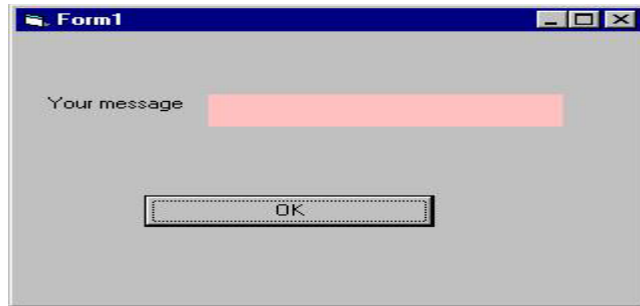
myMessage is a variant data type but typically it is declared as string, which accept the message input by the users. The arguments are explained as follows:

Prompt - The message displayed normally as a question asked.

Title - The title of the Input Box.

default-text - The default text that appears in the input field where users can use it as his intended input or he may change to the message he wish to key in.

x-position and y-position - the position or the coordinate of the input box.



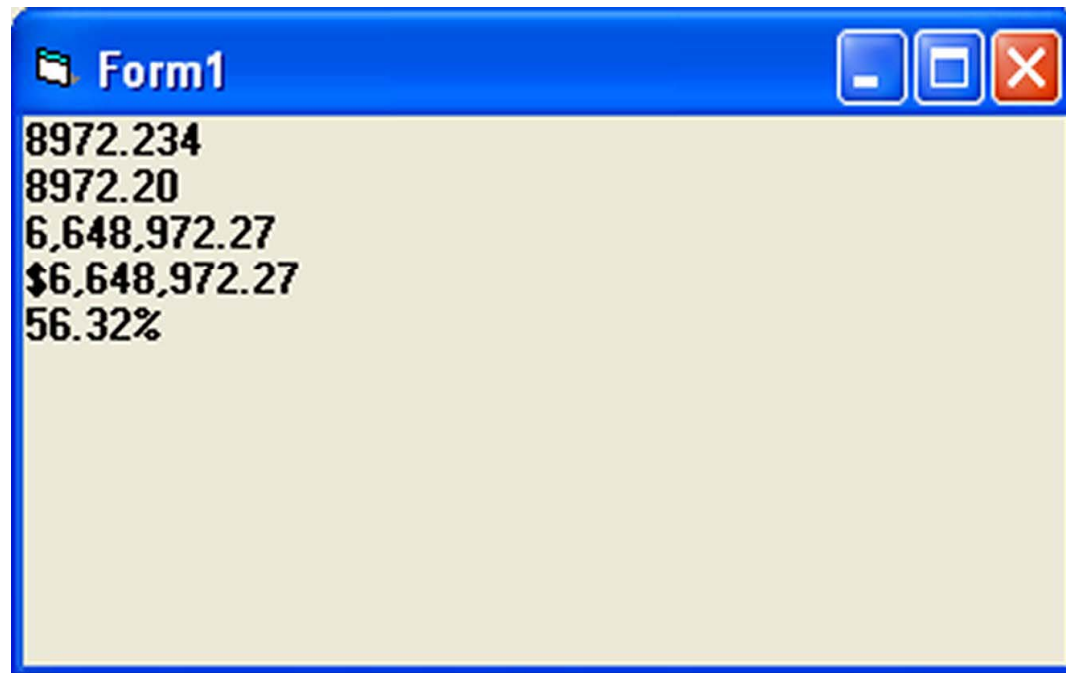
```
Private Sub OK_Click()  
    Dim userMsg As String  
    userMsg = InputBox("What is your  
        message?", "Message Entry  
        Form", "Enter your message here",  
        500, 700)  
    If userMsg <> "" Then  
        message.Caption = userMsg  
    Else  
        message.Caption = "No  
        Message"  
    End If  
End Sub
```



Formatting Functions

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

```
Private Sub Form_Activate()  
Print Format (8972.234, "General Number")  
Print Format (8972.2, "Fixed")  
Print Format (6648972.265, "Standard")  
Print Format (6648972.265, "Currency")  
Print Format (0.56324, "Percent")  
End Sub
```



Creating User-Defined Functions

Creating Your Own Function

The general format of a function is as follows:

Public Function functionName (Arg As dataType,.....) As
dataType

or

Private Function functionName (Arg As dataType,.....) As
dataType

- * Public indicates that the function is applicable to the whole project and
Private indicates that the function is only applicable to a certain
module or procedure.
-

The following program will automatically compute examination grades based on the marks that a student obtained. The code is shown on the right.

The image shows a screenshot of a Windows application window titled "Form1". The window has a blue title bar with standard minimize, maximize, and close buttons. The main area of the window has a light gray background with a grid of small dots. The text "Mark" is positioned to the left of a white rectangular text box. Below the text box, the text "Grade" is displayed. At the bottom of the window, there are two gray buttons with black outlines. The left button is labeled "Compute" and the right button is labeled "End".

```
Public Function grade(mark As Variant) As String
```

```
    Select Case mark
```

```
        Case Is >= 80
```

```
            grade = "A"
```

```
        Case Is >= 70
```

```
            grade = "B"
```

```
        Case Is >= 60
```

```
            grade = "C"
```

```
        Case Is >= 50
```

```
            grade = "D"
```

```
        Case Is >= 40
```

```
            grade = "E"
```

```
        Case Else
```

```
            grade = "F"
```

```
    End Select
```

```
End Function
```

```
Private Sub compute_Click()
```

```
    grading.Caption = grade(mark)
```

```
End Sub
```

Arrays

Introduction to Arrays

By definition, an array is a list of variables, all with the same data type and name. When we work with a single item, we only need to use one variable. However, if we have a list of items which are of similar type to deal with, we need to declare an array of variables instead of using a variable for each item. For example, if we need to enter one hundred names, we might have difficulty in declaring 100 different names, this is a waste of time and efforts. So, instead of declaring one hundred different variables, we need to declare only one array. We differentiate each item in the array by using subscript, the index value of each item, for example name(1), name(2),name(3)etc. , which will make declaring variables streamline and much systematic.

Dimension of an Array

An array can be one dimensional or multidimensional.

One dimensional array is like a list of items or a table that consists of one row of items or one column of items. A twodimensional array will be a table of items that make up of rows and columns. While the format for a one dimensional array is `ArrayName(x)`, the format for a two dimensional array is `ArrayName(x,y)` while a three dimensional array is `ArrayName(x,y,z)` . Normally it is sufficient to use one dimensional and two dimensional array ,you only need to use higher dimensional arrays if you need with engineering problems or even some accounting problems.Let me illustrates the the arrays with tables.

One dimensional Array

Student Name	Name(1)	Name(2)	Name(3)	Name(4)	Name(5)	Name(6)
--------------	---------	---------	---------	---------	---------	---------

Two Dimensional Array

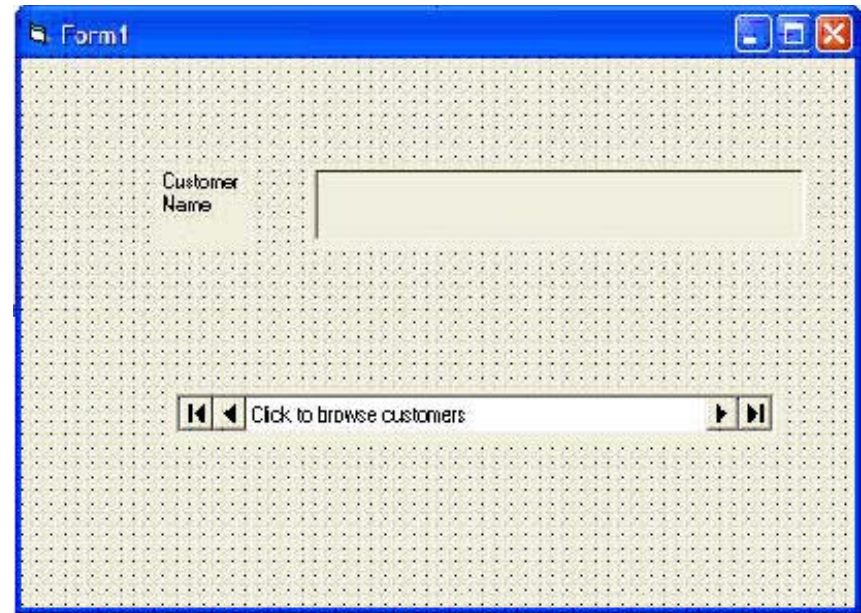
Name(1,1)	Name(1,2)	Name(1,3)	Name(1,4)
Name(2,1)	Name(2,2)	Name(2,3)	Name(2,4)
Name(3,1)	Name(3,2)	Name(3,3)	Name(3,4)

The code

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

Creating database applications in VB-Part I

Visual basic allows us to manage databases created with different database programs such as MS Access, Dbase, Paradox and etc. In this lesson, we are not dealing with how to create database files but we will see how we can access database files in the VB environment.



Form1



Customer
Name

Maria Anders

Click to browse customers



Customer Name

Maria Anders

Address

Obere Str. 57

City

Berlin

Tel. No

030-0074321

⏪ ⏩ Click to browse customers ⏪ ⏩

Creating database applications in VB-Part II

The following are some of the commands that you can use to move the pointer around

data_navigator.RecordSet.MoveFirst record	' Move to the first
data_navigator.RecordSet.MoveLast record	' Move to the last
data_navigator.RecordSet.MoveNext record	' Move to the next
data_navigator.RecordSet.Previous record	' Move to the first

You can also add, save and delete records using the following commands:

data_navigator.RecordSet.AddNew	' Adds a new record
data_navigator.RecordSet.Update the new record	' Updates and saves
data_navigator.RecordSet.Delete record	' Deletes a current

*note: data_navigator is the name of data control

```
Private Sub Command2_Click()  
    dtaBooks.Recordset.MoveFirst  
End Sub
```

```
Private Sub Command1_Click()  
    dtaBooks.Recordset.MoveNext  
End Sub
```

```
Private Sub Command3_Click()  
    dtaBooks.Recordset.MovePrevious  
End Sub
```

```
Private Sub Command4_Click()  
    dtaBooks.Recordset.MoveLast  
End Sub
```

Customer Name: Maria Anders

Address: Obere Str. 57

City: Berlin

Tel. No: 030-0074321

Navigation buttons: First Record, Next Record, Previous Record, Last Record

Close Database button

Creating VB database applications using ADO control

To build VB database applications using data control.

However, data control is not a very flexible tool as it could only work with limited kinds of data and must work strictly in the Visual Basic environment. To overcome these limitations, we can use a much more powerful data control in Visual Basic, known as ADO control. ADO stands for ActiveX data objects. As ADO is ActiveX-based, it can work in different platforms (different computer systems) and different programming languages. Besides, it can access many different kinds of data such as data displayed in the Internet browsers, email text and even graphics other than the usual relational and non relational database information.

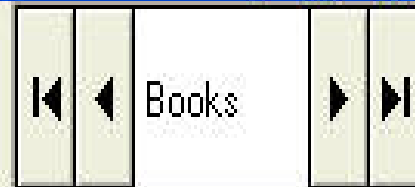
To be able to use ADO data control, you need to insert it into the toolbox. To do this, simply press Ctrl+T to open the components dialog box and select Microsoft ActiveX Data Control 6. After this, you can proceed to build your ADO-based VB database applications.

The following example will illustrate how to build a relatively powerful database application using ADO data control. First of all, name the new form as **frmBookTitle** and change its caption to **Book Titles-ADO Application**. Secondly, insert the ADO data control and name it as **adoBooks** and change its caption to **book**. Next, insert the necessary labels, text boxes and command buttons. The runtime interface of this program is shown in the diagram below, it allows adding and deletion as well as updating and browsing of data.

Book Titles - ADO Application



Book Titles



Title :

Year Published

ISBN:

Publisher's ID

Subject:

Property Pages



General

Source of Connection

Use Data Link File

Browse...

Use ODBC Data Source Name

New...

Use Connection String

Provider=Microsoft.Jet.OLEDB.3.51;Persist Security

Build...

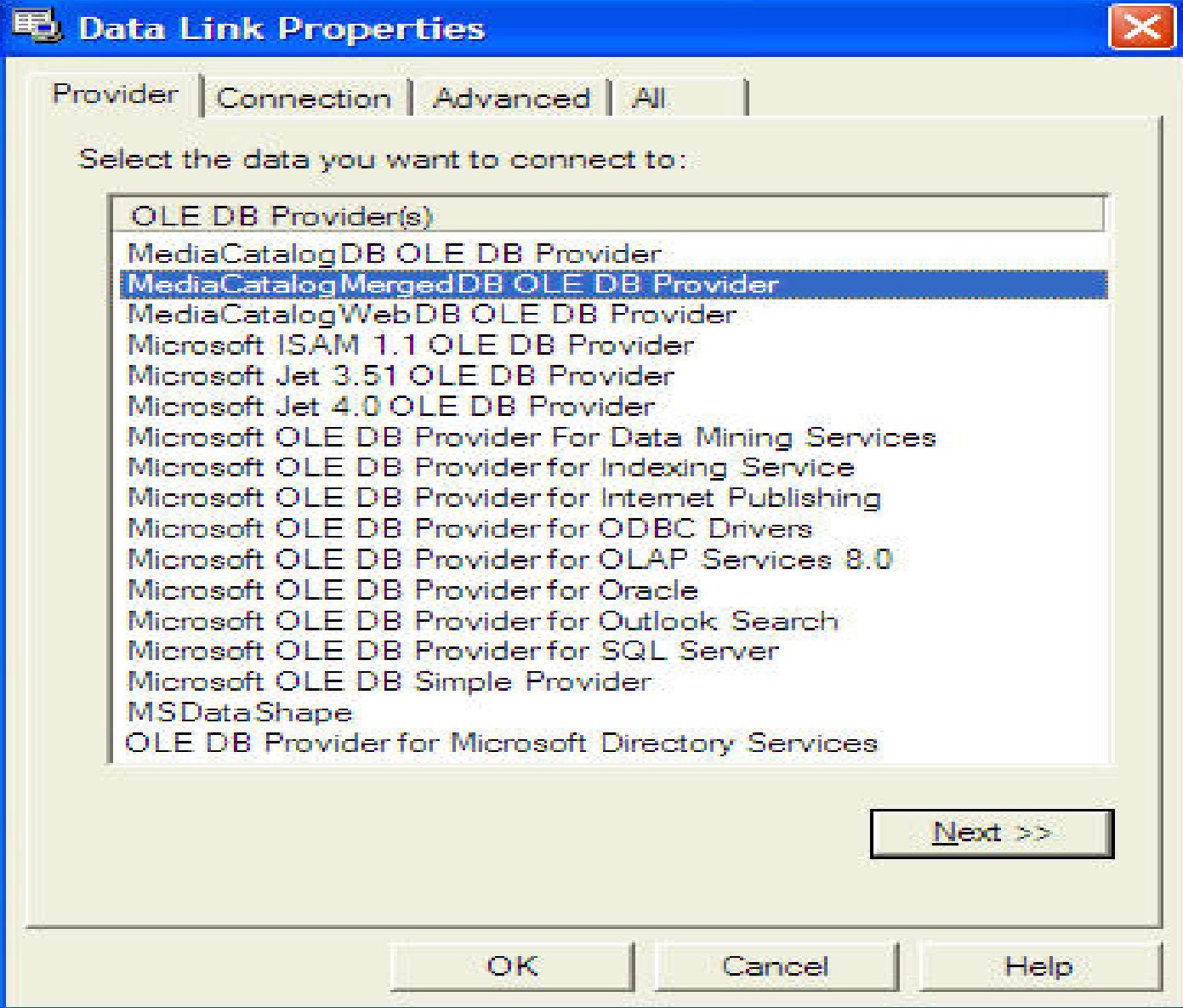
Other Atributes:

OK

Cancel

Apply

Help



Property Pages



RecordSource

RecordSource

Command Type

2 - adCmdTable

Table or Stored Procedure Name

Titles

Command Text (SQL)

OK

Cancel

Apply

Help

For the **Save** button, the program codes are as follow:

```
Private Sub cmdSave_Click()  
adoBooks.Recordset.Fields("Title") = txtTitle.Text  
adoBooks.Recordset.Fields("Year Published")  
= txtPub.Text  
adoBooks.Recordset.Fields("ISBN") =  
txtISBN.Text  
adoBooks.Recordset.Fields("PubID") =  
txtPubID.Text  
adoBooks.Recordset.Fields("Subject") =  
txtSubject.Text  
adoBooks.Recordset.Update  
End Sub
```

For the **Add** button, the program codes are as follow:

```
Private Sub cmdAdd_Click()  
adoBooks.Recordset.AddNew  
End Sub
```

For the **Delete** button, the program codes are as follow:

```
Private Sub cmdDelete_Click()  
Confirm = MsgBox("Are you sure you want to delete this  
record?", vbYesNo, "Deletion Confirmation")  
If Confirm – vbYes Then  
adoBooks.Recordset.Delete  
MsgBox "Record Deleted!", , "Message"  
Else  
MsgBox "Record Not Deleted!", , "Message"  
End If  
End Sub
```

For the **Cancel** button, the program codes are as follow:

```
Private Sub cmdCancel_Click()
```

```
txtTitle.Text = ""
```

```
txtPub.Text = ""
```

```
txtPubID.Text = ""
```

```
txtISBN.Text = ""
```

```
txtSubject.Text = ""
```

```
End Sub
```

For the Previous (<) button, the program codes are
Private Sub cmdPrev_Click()

```
If Not adoBooks.Recordset.EOF Then  
adoBooks.Recordset.MovePrevious  
If adoBooks.Recordset.EOF Then  
adoBooks.Recordset.MoveNext  
End If  
End If  
End Sub
```

For the Next(>) button, the program codes are
Private Sub cmdNext_Click()

```
If Not adoBooks.Recordset.EOF Then  
adoBooks.Recordset.MoveNext  
If adoBooks.Recordset.EOF Then  
adoBooks.Recordset.MovePrevious  
End If  
End If  
End Sub
```

THANK YOU
