1.1 The concept of computer programming

Before we begin Visual Basic 6 programming, let us understand some basic concepts of programming. According to Webopedia, a computer program is an organized list of instructions that, when executed, causes the computer to behave in a predetermined manner. Without programs, computers are useless. Therefore, programming means designing or creating a set of instructions to ask the computer to carry out certain jobs which normally are very much faster than human beings can do.

A lot of people think that computer CPU is a very intelligent thing, which in actual fact it is a dumb and inanimate object that can do nothing without human assistant. The microchips of a CPU can only understand two distinct electrical states, namely, the on and off states, or 0 and 1 codes in the binary system. So, the CPU only understands a combinations of 0 and 1 codes, a language which we called machine language. Machine language is extremely difficult to learn and it is not for us laymen to master it easily. Fortunately , we have many smart programmers who wrote interpreters and compilers that can translate human language-like programs such as BASIC into machine language so that the computer can carry out the instructions entered by the users. Machine language is known as the primitive language while Interpreters and compilers like Visual Basic are called high-level language. Some of the high level computer languages beside Visual Basic are Fortran, Cobol, Java, C, C++, Turbo Pascal, and etc .

1.2 What is Visual Basic?

VISUAL BASIC is a high level programming language which was evolved from the earlier DOS version called BASIC. BASIC means Beginners' All-purpose Symbolic Instruction Code. It is a very easy programming language to learn. The codes look a lot like English Language. Different software companies produced different version of BASIC, such as Microsoft QBASIC, QUICKBASIC, GWBASIC ,IBM BASICA and so on. However, it seems people only use Microsoft Visual Basic today, as it is a well developed programming language and supporting resources are available everywhere. Now, there are many versions of VB exist in the market, the most popular one and still widely used by many VB programmers is none other than Visual Basic 6. We also have VB.net, VB2005 and the latest VB2008, which is a fully object oriented programming (OOP) language. It is more powerful than VB6 but looks more complicated to master. If you wish to learn VB2008, click on the VB2008 Tutorial.

VISUAL BASIC is a VISUAL and events driven Programming Language. These are the main divergence from the old BASIC. In BASIC, programming is done in a text-only environment and the program is executed sequentially. In VB, programming is done in a graphical environment. In the old BASIC, you have to write program codes for each graphical object you wish to display it on screen, including its position and its color. However, In VB , you just need to drag and drop any graphical object anywhere on the form, and you can change its color any time using the properties windows.

On the other hand, because users may click on a certain object randomly, so each object has to be programmed independently to be able to response to those

actions (events). Therefore, a VB Program is made up of many subprograms, each has its own program codes, and each can be executed independently and at the same time each can be linked together in one way or another.

1.3 What programs can you create with Visual Basic 6?

With VB 6, you can create any program depending on your objective. For example, if you are a college or university lecturer, you can create educational programs to teach business, economics, engineering, computer science, accountancy , financial management, information system and more to make teaching more effective and interesting. If you are in business, you can also create business programs such as inventory management system , point-of-sale system, payroll system, financial program as well as accounting program to help manage your business and increase productivity. For those of you who like games and working as games programmer, you can create those programs as well. Indeed, there is no limit to what program you can create ! There are many such programs in this tutorial, so you must spend more time on the tutorial in order to learn how to create those programs. If you wish to see some of the sample programs, you can take a look at the link below:

VB Sample Programs

1.4 The Visual Basic 6 Integrated Development Environment

Before you can program in VB 6, you need to install Visual Basic 6 in your computer. If you do not own a copy of Visual Basic 6 software yet , you can purchase it from Amazon.com by clicking the link below:

Microsoft Visual Basic 6.0 Professional

Basically any present computer system should be able to run the program, be it a Intel Pentium II, Intel Pentium III, Intel Pentium IV or even AMD machines, Visual Basic 6 can run without any problem. It might not be true for VB2008, older machines might not be able to run VB2008 as it takes up much more resources, therefore I still prefer VB 6 as it is light and easy to program. It is still very useful and powerful, and I am happy to know that Microsoft Windows Vista can support VB 6. However, if you prefer to learn VB 2008, you can refer VB 2008 Tutorial.

On start up, Visual Basic 6.0 will display the following dialog box as shown in figure 1.1. You can choose to either start a new project, open an existing project or select a list of recently opened programs. A project is a collection of files that make up your application. There are various types of applications that we could create, however, we shall concentrate on creating Standard EXE programs (EXE means executable program). Now, click on the Standard EXE icon to go into the actual Visual Basic 6 programming

environment.

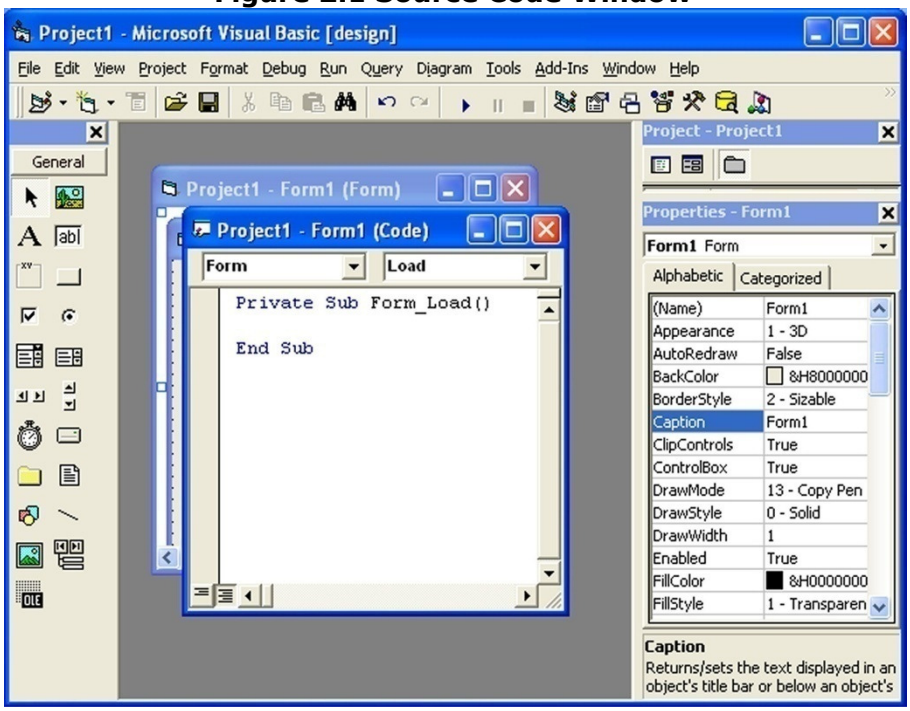
Building Visual Basic Applications

2.1 Creating Your First Application

In this section, we will not go into the technical aspects of Visual Basic programming yet, what you need to do is just try out the examples below to see how does in VB program look like:

Example 2.1.1 is a simple program. First of all, you have to launch Microsoft Visual Basic 6. Normally, a default form with the name Form1 will be available for you to start your new project. Now, double click on Form1, the source code window for Form1 as shown in figure 2.1 will appear. The top of the source code window consists of a list of objects and their associated events or procedures. In figure 2.1, the object displayed is Form and the associated procedure is Load.

Figure 2.1 Source Code Window



When you click on the object box, the drop-down list will display a list of objects you have inserted into your form as shown in figure 2.2. Here, you can see a form with the name Form1, a command button with the name Command1, a Label with the name Label1 and a Picture Box with the name Picture1. Similarly, when you click on the procedure box, a list of procedures associated with the object will be displayed as shown in figure 2.3. Some of the procedures associated with the object Form1 are Activate, Click, DblClick (which means Double-Click) , DragDrop, keyPress and more. Each object has its own set of procedures. You can always

select an object and write codes for any of its procedure in order to perform certain tasks.

You do not have to worry about the beginning and the end statements (i.e. Private Sub Form_Load…….End Sub.); Just key in the lines in between the above two statements exactly as are shown here. When you press F5 to run the program, you will be surprise that nothing shown up .In order to display the output of the program, you have to add the Form1.show statement like in Example 2.1.1 or you can just use Form_Activate ( ) event procedure as shown in example 2.1.2. The command Print does not mean printing using a printer but it means displaying the output on the computer screen. Now, press F5 or click on the run button to run the program and you will get the output as shown in figure 2.4.

You can also perform arithmetic calculations as shown in example 2.1.2. VB uses * to denote the multiplication operator and / to denote the division operator. The output is shown in figure 2.3, where the results are arranged vertically.

Figure 2.2: List of Objects

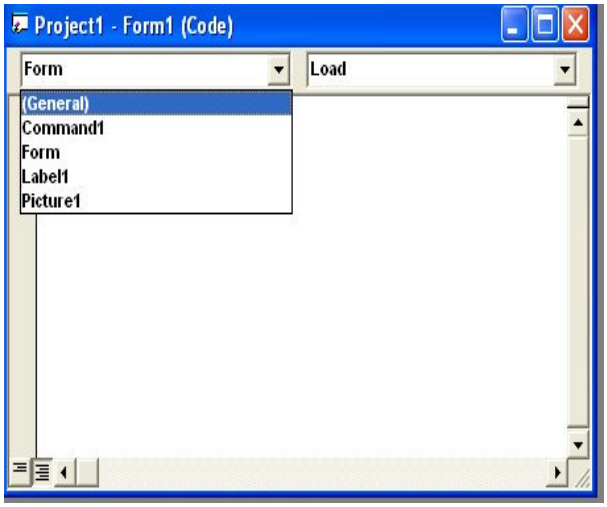
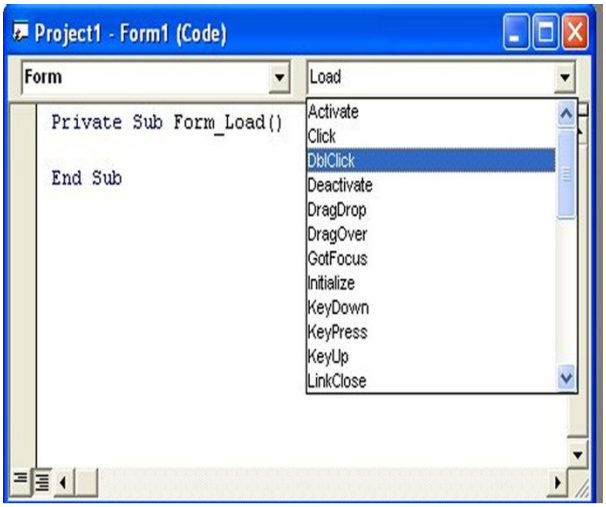


Figure 2.3: List of Procedures

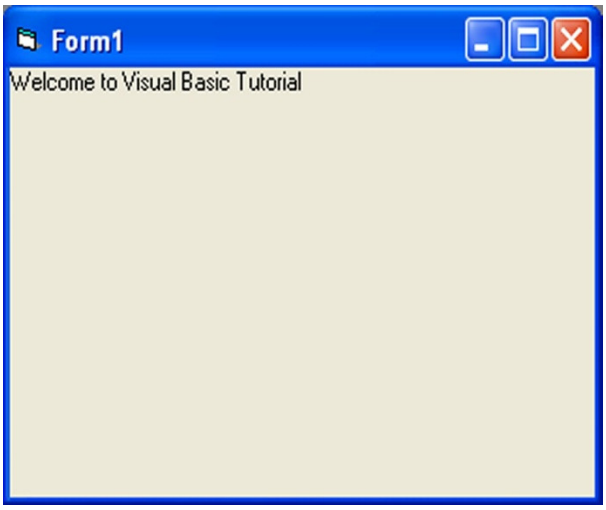


Example 2.1.1

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

Figure 2.4 : The output of example 2.1.1

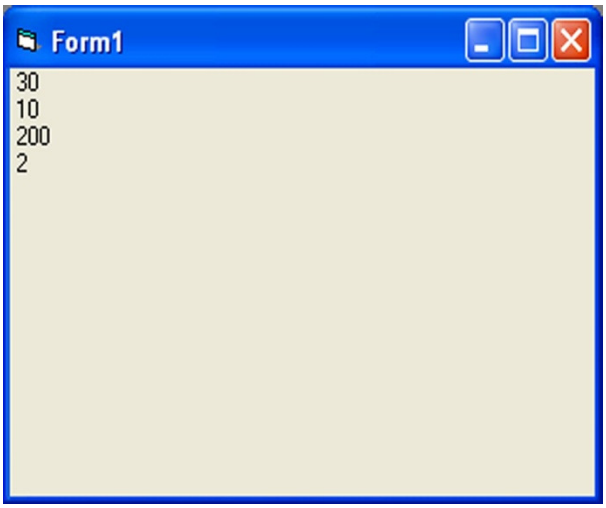
```
        tutorial”

End Sub
```



Example 2.1.2

```
Private Sub Form_Activate ( )

    Print 20 + 10
    Print 20 - 10
    Print 20 * 10
    Print 20 / 10

End Sub
```

Figure 2.5: The output of example 2.1.2



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
```

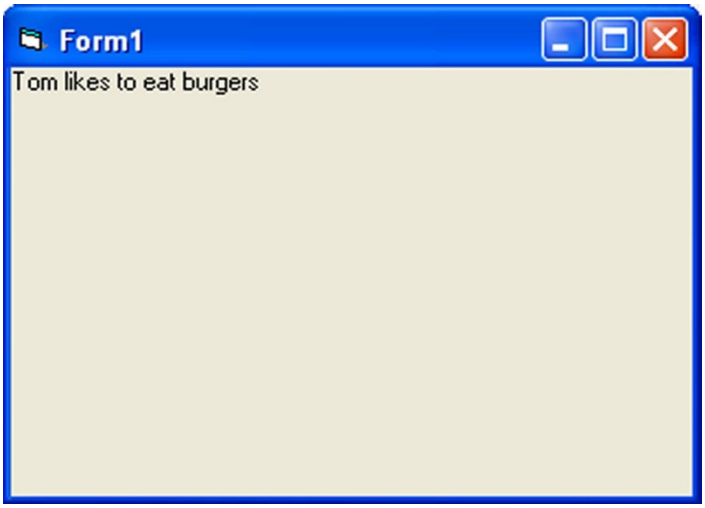
Example 2.1.4(b)

```
Private Sub

A = Tom
B = "likes"
```

```
B = "likes"
C = "to"
D = "eat"
E = "burger"
Print A + B + C + D + E

End Sub
```

```
C = "to"
D = "eat"
E = "burger"
Print A & B & C & D & E

End Sub
```

The Output of Example 2.1.4(a) &(b) is as shown in Figure 2.7.



2.2 Steps in Building a Visual Basic Application

Step 1 : Design the interface

Step 2 : Set properties of the controls (Objects)

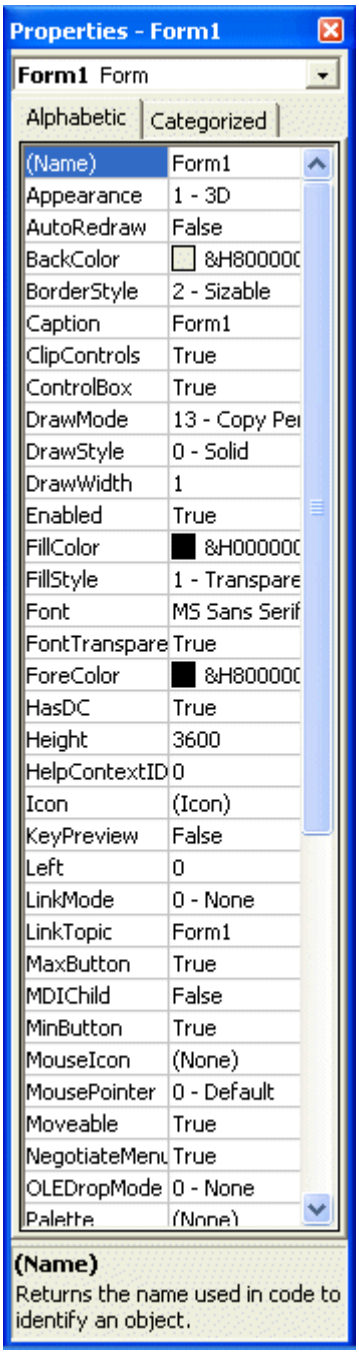
Step 3 : Write the event procedures

3.1 The Control Properties

Before writing an event procedure for the control to response to a user's input, you have to set certain properties for the control to determine its appearance and how it will work with the event procedure. You can set the properties of the controls in the properties window or at runtime.

Figure 3.1 below is a typical properties window for a form. You can rename the form

Figure 3.1



caption to any name that you like best. In the properties window, the item appears at the top part is the object currently selected (in Figure 3.1, the object selected is Form1). At the bottom part, the items listed in the left column represent the names of various properties associated with the selected object while the items listed in

the right column represent the states of the properties. Properties can be set by highlighting the items in the right column then change them by typing or selecting the options available.

For example, in order to change the caption, just highlight Form1 under the name Caption and change it to other names. You may also try to alter the appearance of the form by setting it to 3D or flat. Other things you can do are to change its foreground and background color, change the font type and font size, enable or disable minimize and maximize buttons and etc.

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. Later you will learn how to change the shapes randomly by using the **RND** function.

```
Private Sub Form_Load()

Shape1.Shape = 3

End Sub
```

I would like to stress that knowing how and when to set the objects' properties is very important as it can help you to write a good program or you may fail to write a good program. So, I advice you to spend a lot of time playing with the objects' properties.

I am not going into the details on how to set the properties. However, I would like to stress a few important points about setting up the properties.

- You should set the Caption Property of a control clearly so that a user knows what to

do with that command. For example, in the calculator program, all the captions of the command buttons such as +, - , MC, MR are commonly found in an ordinary calculator, a user should have no problem in manipulating the buttons.

- A lot of programmers like to use a meaningful name for the Name Property may be because it is easier for them to write and read the event procedure and easier to debug or modify the programs later. However, it is not a must to do that as long as you label your objects clearly and use comments in the program whenever you feel necessary. T
- One more important property is whether the control is enabled or not.
- Finally, you must also considering making the control visible or invisible at runtime, or when should it become visible or invisible.

3.2 Handling some of the common controls

3.2.1 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.

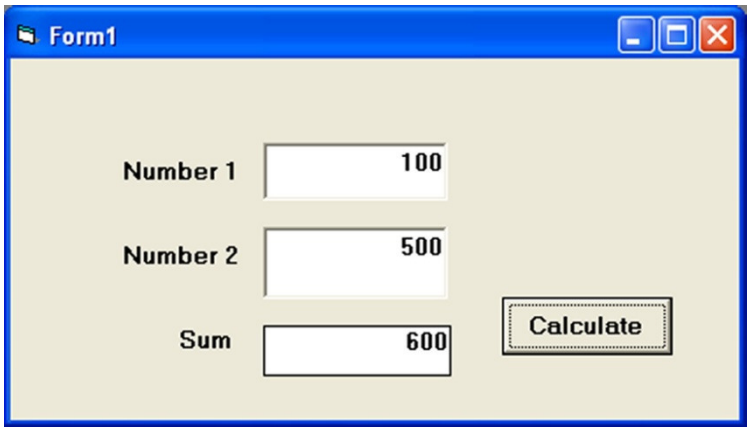
Example 3.1

In this program, two text boxes are inserted into the form together with a few labels. The two text boxes are used to accept inputs from the user and one of the labels will be used to display the sum of two numbers that are entered into the two text boxes. Besides, a command button is also programmed to calculate the sum of the two numbers using the plus operator. The program use creates a variable sum to accept the summation of values from text box 1 and text box 2.The procedure to calculate and to display the output on the label is shown below. The output is shown in Figure 3.2

```
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
```

Figure 3.2



3.2.2 The Label

The label is a very useful control for Visual Basic, as it is not only used to provide instructions and guides to the users, it can also be used to display outputs. One of its most important properties is **Caption**. Using the syntax **label.Caption**, it can display text and numeric data . You can change its caption in the properties window and also at runtime. Please refer to Example 3.1 and Figure 3.1 for the usage of label.

3.2.3 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
```

3.2.4 The Picture Box

The Picture Box is one of the controls that is used to handle graphics. You can load a picture at design phase by clicking on the picture item in the properties window and select the picture from the selected folder. You can also load the picture at runtime using the **LoadPicture** method. For example, the statement will load the picture grape.gif into the picture box.

```
Picture1.Picture=LoadPicture ("C:\VB program\Images\grape.gif")
```

You will learn more about the picture box in future lessons. The image in the picture box is not resizable.

3.2.5 The Image Box

The Image Box is another control that handles images and pictures. It functions almost identically to the picture box. However, there is one major difference, the image in an Image Box is stretchable, which means it can be resized. This feature is not available in the Picture Box. Similar to the Picture Box, it can also use the LoadPicture method to load the picture. For example, the statement loads the picture grape.gif into the image box.

```
Image1.Picture=LoadPicture ("C:\VB program\Images\grape.gif")
```

3.2.6 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

Example 3.2

```
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

3.2.7 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

Example 3.3

```
Private Sub Form_Load ( )

        Combo1.AddItem "Item1"
        Combo1.AddItem "Item2"
        Combo1.AddItem "Item3"
        Combo1.AddItem "Item4"

End Sub
```

3.2.8 The Check Box

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

Example 3.4

```
Private Sub Command1_Click()

If Check1.Value = 1 And Check2.Value = 0 Then
MsgBox "Apple is selected"
ElseIf Check2.Value = 1 And Check1.Value = 0 Then
MsgBox "Orange is selected"
Else
MsgBox "All are selected"
End If

End Sub
```

3.2.9 The Option Box

The Option Box control also lets the user selects 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.

Example 3.5

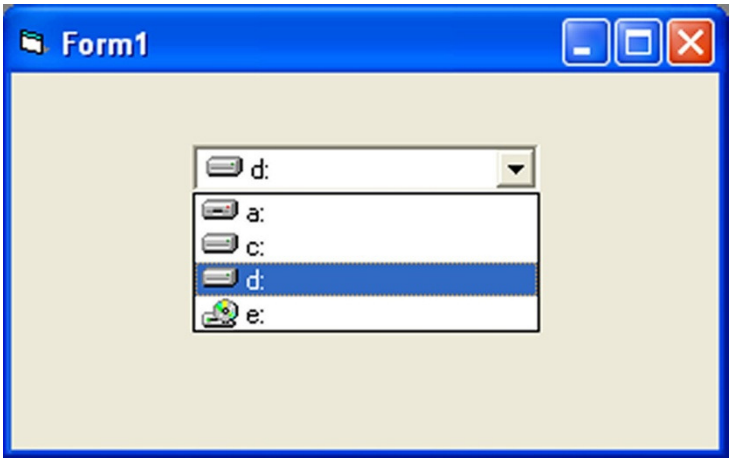
```
Private Sub Option1_Click ( )
        Shape1.Shape = 0
End Sub

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
```

3.2.10 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, you will be able to select different drives from your computer as shown in Figure 3.3

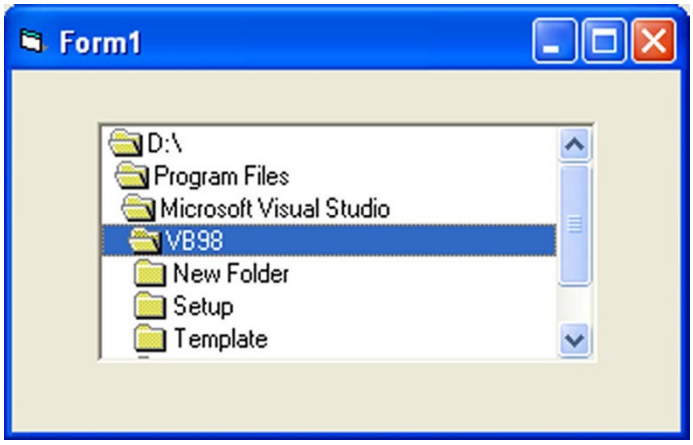
Figure 3.3 The Drive List Box



3.2.11 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, you will be able to select different directories from a selected drive in your computer as shown in Figure 3.4

Figure 3.4 The Directory List Box



3.2.12 The File List Box

The File List Box is for displaying the list of files in a selected directory or folder. When you place this control into the form and run the program, you will be able to shown the list of files in a selected directory as shown in Figure 3.5

You can coordinate the Drive List Box, the Directory List Box and the File List Box to search for the files you want. The procedure will be discussed in later lessons.

Lesson 4 : Writing the Code

In lesson 2, you have learned how to enter the program code and run the sample VB programs but without much understanding about the logics of VB programming. Now, let's get down to learning some basic rules about writing the VB program code.

Each control or object in VB can usually run many kinds of events or procedures; these events are listed in the dropdown list in the code window that is displayed when you double-click on an object and click on the procedures' box(refer to Figure 2.3). Among the events are loading a form, clicking of a command button, pressing a key on the keyboard or dragging an object and more. For each event, you need to write an event procedure so that it can perform an action or a series of actions

To start writing an event procedure, you need to double-click an object. For example, if you want to write an event procedure when a user clicks a command button, you double-click on the command button and an event procedure will appear as shown in Figure 2.1. It takes the following format:

```
Private Sub Command1_Click

(Key in your program code here)

End Sub
```

You then need to key-in the procedure in the space between Private Sub Command1_Click..... End Sub. Sub actually stands for sub procedure that made up a part of all the procedures in a program. The program code is made up of a number of statements that set certain properties or trigger some actions. The syntax of Visual Basic's program code is almost like the normal English language though not exactly the same, so it is very easy to learn.

The syntax to set the property of an object or to pass certain value to it is :

Object.Property

where **Object** and Property is separated by a period (or dot). For example, the statement **Form1.Show** means to show the form with the name Form1, **Iabel1.Visible=true** means

label1 is set to be visible, **Text1.text="VB"** is to assign the text VB to the text box with the name Text1, Text2.text=100 is to pass a value of 100 to the text box with the name text2, **Timer1.Enabled=False** is to disable the timer with the name Timer1 and so on. Let's examine a few examples below:

Example 4.1

```
Private Sub Command1_click
        Label1.Visible=false
        Label2.Visible=True
        Text1.Text="You are correct!"
        End sub
```

Example 4.2

```
Private Sub Command1_click
        Label1.Caption=" Welcome"
        Image1.visible=true
End sub
```

Example 4.3

```
Private Sub Command1_click
        Pictuire1.Show=true
        Timer1.Enabled=True
        Lable1.Caption="Start Counting
End sub
```

In Example 4.1, clicking on the command button will make label1 become invisible and label2 become visible; and the text" You are correct" will appear in TextBox1. In example 4.2, clicking on the command button will make the caption label1 change to "Welcome" and Image1 will become visible. In example 4.3 , clicking on the command button will make Picture1 show up, timer starts running and the caption of label1 change to "Start Counting".

Syntaxes that do not involve setting of properties are also English-like, some of the commands are **Print, If...Then....Else....End If, For...Next, Select Case.....End Select , End** and **Exit Sub.** For example, **Print " Visual Basic"** is to display the text Visual Basic on screen and **End** is to end the program. Other commands will be explained in details in the coming lessons.

Program code that involve calculations is very easy to write, you need to write them almost like you do in mathematics. However, in order to write an event procedure that involves calculations, you need to know the basic arithmetic operators in VB as they are not exactly the same as the normal operators we use, except for **+** and **- .** For multiplication, we use *****, for division we use **/**, for raising a number x to the power of n, we use **x ^n** and for square root, we use **Sqr(x).** VB offers many more advanced mathematical functions such as **Sin, Cos, Tan** and **Log,** they will be discussed in lesson 10. There are also two important functions that are related to arithmetic operations, i.e. the functions **Val** and **Str$** where Val is to convert text entered into a textbox to numerical value and Str$ is to display a numerical value in a textbox as a string (text). While the function Str$ is as important as VB can display a numeric values as string implicitly, failure to use Val will results in wrong calculation. Let's examine example 4.4 and example 4.5.

Example 4.4

```
Private Sub Form_Activate()

  Text3.text=text1.text+text2.text

End Sub
```

Example 4.5

```
Private Sub Form_Activate()

  Text3.text=val(text1.text)+val(text2.text)

End Sub
```

When you run the program in example 4.4 and enter 12 in textbox1 and 3 in textbox2 will give you a result of 123, which is wrong. It is because VB treat the numbers as string and so it just joins up the two strings. On the other hand, running exampled 4.5 will give you the correct result, i.e., 15.

Lesson 5: Managing Visual Basic Data

There are many types of data that we come across in our daily life. For example, we need to handle data such as names, addresses, money, date, stock quotes, statistics and more everyday. Similarly in Visual Basic, we have to deal with all sorts of of data, some can be mathematically calculated while some are in the form of text or other forms. VB divides data into different types so that it is easier to manage when we need to write the code involving those data.

5.1 Visual Basic Data Types

Visual Basic classifies the information mentioned above into two major data types, they are the numeric data types and the non-numeric data types.

5.1.1 Numeric Data Types

Numeric data types are types of data that consist of numbers, which can be computed mathematically with various standard operators such as add, minus, multiply, divide and more. Examples of numeric data types are examination marks, height, weight, the number of students in a class, share values, price of goods, monthly bills, fees and others. In Visual Basic, numeric data are divided into 7 types, depending on the range of values they can store. Calculations that only involve round figures or data that does not need precision can use Integer or Long integer in the computation. Programs that require high precision calculation need to use Single and Double decision data types, they are also called floating

point numbers. For currency calculation , you can use the currency data types. Lastly, if even more precision is required to perform calculations that involve a many decimal points, we can use the decimal data types. These data types summarized in Table 5.1

Table 5.1: Numeric Data Types

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

5.1.2 Non-numeric Data Types

Nonnumeric data types are data that cannot be manipulated mathematically using standard arithmetic operators. The non-numeric data comprises text or string data types, the Date data types, the Boolean data types that store only two values (true or false), Object data type and Variant data type .They are summarized in Table 5.2

Table 5.2: Nonnumeric Data Types

Data Type	Storage	Range
String(fixed length)	Length of string	1 to 65,400 characters
String(variable length)	Length + 10 bytes	0 to 2 billion characters
Date	8 bytes	January 1, 100 to December 31, 9999
Boolean	2 bytes	True or False
Object	4 bytes	Any embedded object
Variant(numeric)	16 bytes	Any value as large as Double
Variant(text)	Length+22 bytes	Same as variable-length string

5.1.3 Suffixes for Literals

Literals are values that you assign to data. In some cases, we need to add a suffix behind a literal so that VB can handle the calculation more accurately. For example, we can use num=1.3089# for a Double type data. Some of the suffixes are displayed in Table 5.3.

Table 5.3

Suffix	Data Type
&	Long
!	Single
#	Double
@	Currency

In addition, we need to enclose string literals within two quotations and date and time literals within two # sign. Strings can contain any characters, including numbers. The following are few examples:

```
memberName="Turban, John."
TelNumber="1800-900-888-777"
LastDay=#31-Dec-00#
ExpTime=#12:00 am#
```

5.2 Managing Variables

Variables are like mail boxes in the post office. The contents of the variables changes every now and then, just like the mail boxes. In term of VB, variables are areas allocated by the computer memory to hold data. Like the mail boxes, each variable must be given a name. To name a variable in Visual Basic, you have to follow a set of rules.

5.2.1 Variable Names

The following are the rules when naming the variables in Visual Basic

- It must be less than 255 characters
- No spacing is allowed
- It must not begin with a number
- Period is not permitted

Examples of valid and invalid variable names are displayed in Table 5.4

Table 5.4

Valid Name	Invalid Name
My_Car	My.Car
ThisYear	1NewBoy
Long_Name_Can_beUSE	He&HisFather *& is not acceptable

5.2.2 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*

Example 5.1

```
Dim password As String
Dim yourName As String
Dim firstnum As Integer
Dim secondnum As Integer
Dim total As Integer
Dim doDate As Date
```

You may also combine them in one line , separating each variable with a comma, as follows:

Dim password As String, yourName As String, firstnum As Integer,.....

If data type is not specified, VB will automatically declare the variable as a Variant.
For string declaration, there are two possible formats, one for the variable-length string and another for the fixed-length string. For the variable-length string, just use the same format as example 5.1 above. However, for the fixed-length string, you have to use the format as shown below:

*Dim VariableName as String * n*, where n defines the number of characters the string can hold.

Example 5.2:

Dim yourName as String * 10

yourName can holds no more than 10 Characters.

5.3 Constants

Constants are different from variables in the sense that their values do not change during the running of the program.

5.3.1 Declaring a Constant

The format to declare a constant is

Const *Constant Name* As *Data Type = Value*

Example 5.3

Const Pi As Single=3.142

Const Temp As Single=37

Const Score As Single=100

Lesson 6: Working with Variables

6.1 Assigning Values to 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
```

6.2 Operators in Visual Basic

To compute inputs from users and to generate results, we need to use various mathematical operators. In Visual Basic, except for + and -, the symbols for the operators are different from normal mathematical operators, as shown in Table 6.1.

Table 6.1: Arithmetic Operators

Operator	Mathematical	Example

	function	
^	Exponential	2^4=16
*	Multiplication	4*3=12, (5*6))2=60
/	Division	12/4=3
Mod	Modulus(return the remainder from an integer division)	15 Mod 4=3 255 mod 10=5
\	Integer Division(discards the decimal places)	19\4=4
+ or &	String concatenation	"Visual"&"Basic"="Visual Basic"

Example 6.1

```
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
```

Example 6.2

```
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
```

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.	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.

Lesson 7 : Controlling Program Flow

In previous lessons, we have learned how to create Visual Basic code that can accept input from the user and display the output without controlling the program flow. In this chapter, you will learn how to crreate 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.

7.1 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. These operators are shown in Table 7.1.

7.2 Logical Operators

In addition to conditional operators, there are a few logical operators which offer added power to the VB programs. There are shown in Table 7.2.

Table 7.1: Conditional Operators	**Table 7.2:Logical Operators**
	Operator **Meaning**

Operator	Meaning
=	Equal to
>	More than
<	Less Than
>=	More than and equal
<=	Less than and equal
<>	Not Equal to

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

* You can also compare strings with the above operators. However, there are certain rules to follows: Upper case letters are less than lowercase letters, "A"<"B"<"C"<"D".....<"Z" and number are less than letters.

7.3 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.

Example:

```
Private Sub OK_Click()
  firstnum=Val(usernum1.Text)
  secondnum=Val(usernum2.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
```

Lesson 8 : 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
           .
       .
   Case Else
       Block of one or more VB Statements

End Select
```

Example 8.1

```
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
```

Example 8.2

```
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
```

Example 8.3

Example 8.2 could be rewritten as follows:

```
Dim mark As Single

Private Sub Compute_Click()

'Examination Marks
 mark = mrk.Text

Select Case mark
 Case 0 to 49

    comment.Caption = "Need to work harder"

Case 50 to 59

   comment.Caption = "Average"

Case 60 to 69
  comment.Caption = "Above Average"

Case 70 to 84
comment.Caption = "Good"

Case Else
comment.Caption = "Excellence"

End Select

End Sub
```

Lesson 9: Looping

Visual Basic allows a procedure to be repeated many times as long as the processor until a condition or a set of conditions is fulfilled. This is generally called looping . Looping is a very useful feature of Visual Basic because it makes repetitive works easier. There are two kinds of loops in Visual Basic, the Do...Loop and the For.....Next loop

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

9.2 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. You can examine Example 9.2 for its usage.

9.3 For....Next Loop

The format is:

```
For counter=startNumber to endNumber (Step increment)
   One or more VB statements
Next
```

Please refer to example 9.3a,9.3b and 9.3 c for its usage.

Sometimes the user might want to get out from the loop before the whole repetitive process is executed, the command to use is **Exit For**. To exit a For….Next Loop, you can place the **Exit For** statement within the loop; and it is normally used together with the If…..Then… statement. Let's examine example 9.3 d.

Example 9.1

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

* The above example will keep on adding until counter >1000.
The above example can be rewritten as

```
Do
    num.Text=counter
    counter=counter+1
Loop until counter>1000
```

Example 9.2

```
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.

Example 9.3 a	**Example 9.3 b**
For counter=1 to 10 display.Text=counter Next	For counter=1 to 1000 step 10 counter=counter+1 Next

Example 9.3 c
For counter=1000 to 5 step -5
counter=counter-10
Next
*Notice that increment can be negative
	Example 9.3 d
	
	Private Sub Form_Activate( )
	For n=1 to 10
	If n>6 then
	Exit For
	End If

	Else
	Print n
	End If
	End Sub

Lesson 10: Introduction to VB Built-in Functions

A function is similar to a normal procedure but the main purpose of the function is to accept a certain input from the user and return a value which is passed on to the main program to finish the execution. There are two types of functions, the built-in functions (or internal functions) and the functions created by the programmers.

The general format of a function is

FunctionName (arguments)

The arguments are values that are passed on to the function.

In this lesson, we are going to learn two very basic but useful internal functions of Visual basic , i.e. the **MsgBox( )** and **InputBox ( )** functions.

10.1 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.

Table 10.1: Style Values

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

We can use named constant in place of integers for the second argument to make the programs more readable. In fact, VB6 will automatically shows up a list of names constant where you can select one of them.

Example: yourMsg=MsgBox("Click OK to Proceed", 1, "Startup Menu")

and yourMsg=Msg("Click OK to Proceed". vbOkCancel,"Startup Menu")

are the same.

yourMsg is a variable that holds values that are returned by the MsgBox ( ) function. The values are determined by the type of buttons being clicked by the users. It has to be declared as Integer data type in the procedure or in the general declaration section. **Table**

10.2 shows the values, the corresponding named constant and buttons.

Table 10.2 : Return Values and Command Buttons

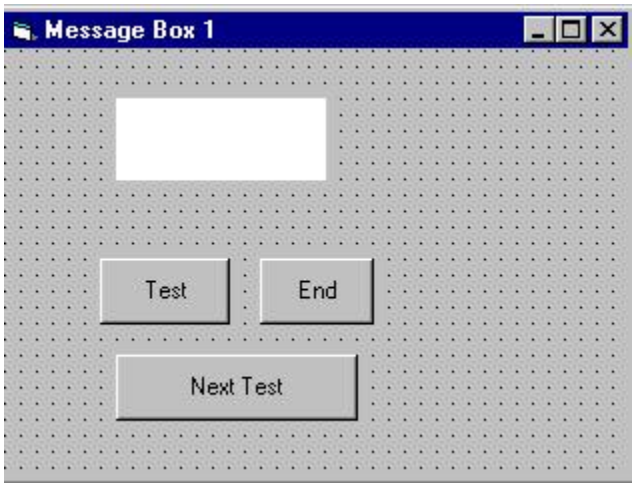
Value	Named Constant	Button Clicked
1	vbOk	Ok button
2	vbCancel	Cancel button
3	vbAbort	Abort button
4	vbRetry	Retry button
5	vbIgnore	Ignore button
6	vbYes	Yes button
7	vbNo	No button

Example 10.1

i. The Interface:

You draw three command buttons and a label as shown in Figure 10.1

Figure 10.1



ii. 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
```

When a user click on the test button, the image like the one shown in Figure 10.2 will appear. As the user click on the OK button, the message "Testing successful" will be displayed and when he/she clicks on the Cancel button, the message "Testing fail" will be displayed.

Figure 10.2

To make the message box looks more sophisticated, you can add an icon besides the message. There are four types of icons available in VB as shown in Table 10.3

Table 10.3

Value	Named Constant	Icon
16	vbCritical	
32	vbQuestion	
48	vbExclamation	
64	vbInformation	

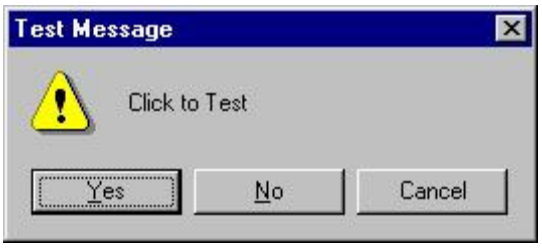
Example 10.2

You draw the same Interface as in example 10.1 but modify the codes as follows:

```
Private Sub test2_Click()

Dim testMsg2 As Integer
testMsg2 = MsgBox("Click to Test",
vbYesNoCancel + vbExclamation, "Test
Message")
If testMsg2 = 6 Then
display2.Caption = "Testing successful"
ElseIf testMsg2 = 7 Then
display2.Caption = "Are you sure?"
Else
display2.Caption = "Testing fail"
End If

End Sub
```

In this example, the following message box will be displayed:

Figure 10.3



10.2 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)

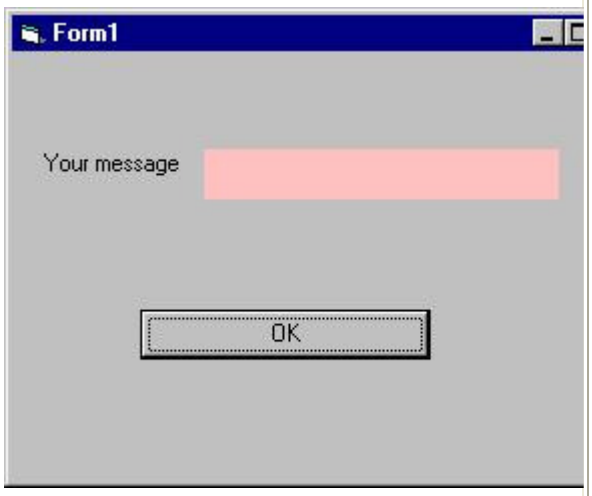
myMessage is a variant data type but typically it is declared as string, which accept the message input by the users. The arguments are explained as follows:

- 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.

Example 10.3

i. The Interface

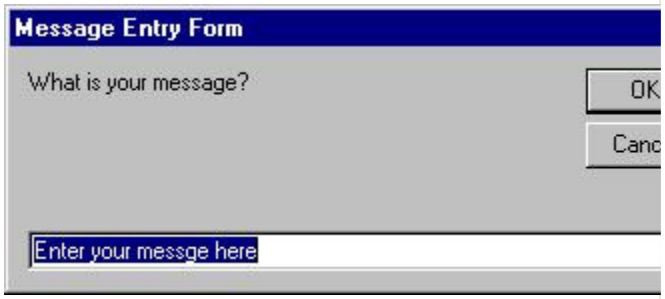
Figure 10.4



ii. The procedure for the OK button

```
Private Sub OK_Click()

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

End Sub
```

When a user click the OK button, the input box as shown in Figure 10.5 will appear. After user entering the message and click OK, the message will be displayed on the caption, if he click Cancel, "No message" will be displayed.



Lesson 11: Mathematical Functions

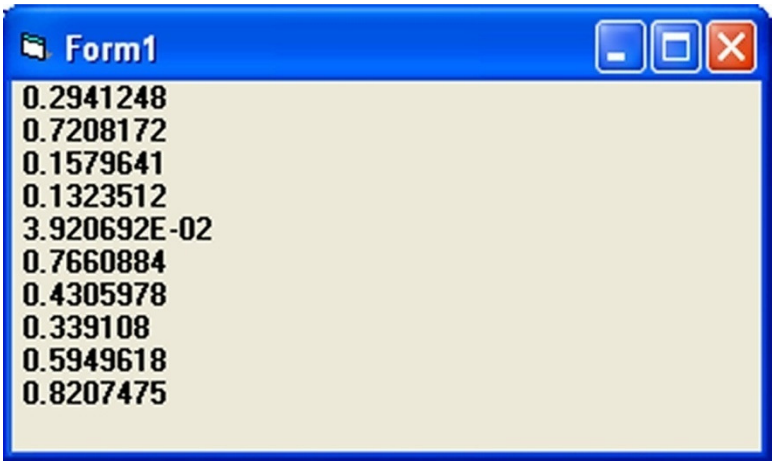
The mathematical functions are very useful and important in programming because very often we need to deal with mathematical concepts in programming such as chance and probability, variables, mathematical logics, calculations, coordinates, time intervals and etc. The common mathematical functions in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

(i) Rnd is very useful when we deal with the concept of chance and probability. The Rnd function returns a random value between 0 and 1. In Example 1. When you run the program, you will get an output of 10 random numbers between 0 and 1. Randomize Timer is a vital statement here as it will randomize the process.

Example 1:

```
Private Sub Form_Activate
        Randomize Timer
        For x=1 to 10
        Print  Rnd
        Next x
End Sub
```

The Output for example 1 is shown below:



Random numbers in its original form are not very useful in programming until we convert them to integers. For example, if we need to obtain a random output of 6 random integers ranging from 1 to 6, which make the program behave as a virtual die, we need to convert the random numbers using the format **Int(Rnd*6)+1**. Let's study the following example:

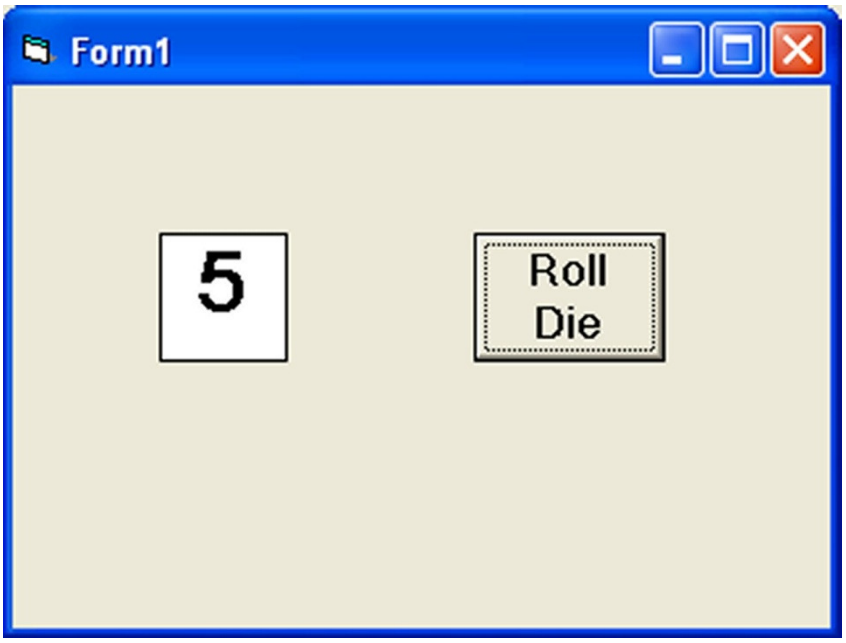
In this example, Int(Rnd*6) will generate a random integer between 0 and 5 because the function **Int** truncates the decimal part of the random number and returns an integer. After adding 1, you will get a random number between 1 and 6 every time you click the command button. For example, let say the random number generated is 0.98, after multiplying it by 6, it becomes 5.88, and using the integer function Int(5.88) will convert the number to 5; and after adding 1 you will get 6.

In this example, you place a command button and change its caption to 'roll die'. You also need to insert a label into the form and clear its caption at the designing phase and make its font bigger and bold. Then set the border value to 1 so that it displays a border; and after that set the alignment to center. The statement Label1.Caption=Num means the integer generated will be displayed as the caption of the label.

Example 2:

```
 Dim num as integer
Private Sub Command1_Click ( )
        Randomize Timer
        Num=Int(Rnd*6)+1
        Label1.Caption=Num
End Sub
```

Now, run the program and then click on the roll die button, you will get an output like the figure below:



The Numeric Functions

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

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

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

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

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

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

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

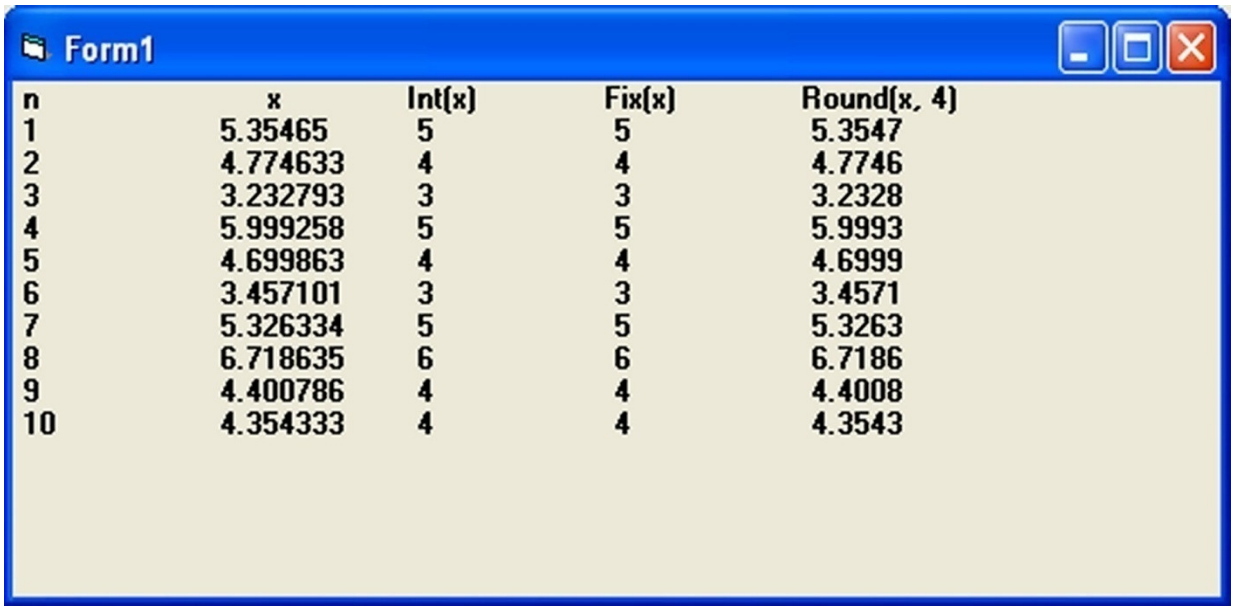
g) **Log** is the function that returns the natural Logarithm of a number. For example,

Log 10= 2.302585

Example 3

This example computes the values of Int(x), Fix(x) and Round(x,n) in a table form. It uses the Do Loop statement and the Rnd function to generate 10 numbers. The statement x = Round (Rnd * 7, 7) rounds a random number between 0 and 7 to 7 decimal places. Using commas in between items will create spaces between them and hence a table of values can be created. The program and output are shown below

```
Private Sub Form_Activate ()

    n = 1

    Print " n", "      x", "Int(x)", "Fix(x)", "Round(x, 4)"

    Do While n < 11

    Randomize Timer

    x = Round (Rnd * 7, 7)

    Print n, x, Int(x), Fix(x), Round(x, 4)

    n = n + 1

    Loop

End Sub
```



Lesson 12: Formatting Functions

Formatting output is a very important part of programming so that the data can be presented systematically and clearly to the users. Data in the previous lesson were presented fairly systematically through the use of commas and some of the functions like Int, Fix and Round. However, to have better control of the output format, we can use a number of formatting functions in Visual basic.

The three most common formatting functions in VB are **Tab**, **Space**, and **Format**

(i) The Tab function

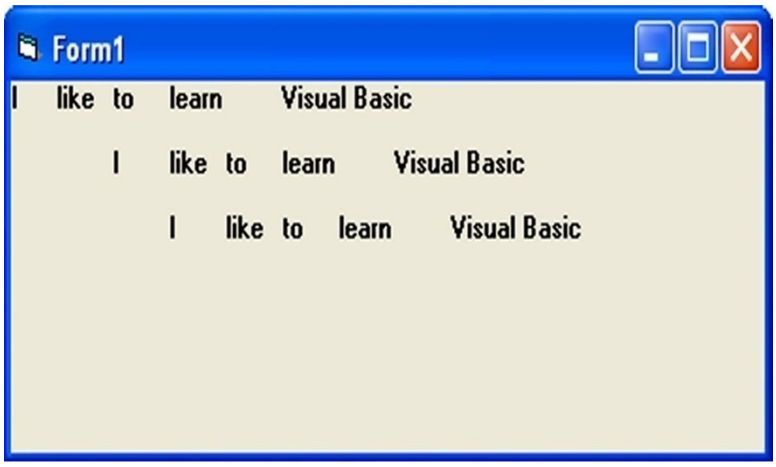
Tab (n); x

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

Example1

```
.Private Sub Form_Activate
        Print "I"; Tab(5); "like"; Tab(10); "to"; Tab(15); "learn"; Tab(20); "VB"
        Print
        Print Tab(10); "I"; Tab(15); "like"; Tab(20); "to"; Tab(25); "learn"; Tab(20); "VB"
        Print
        Print Tab(15); "I"; Tab(20); ; "like"; Tab(25); "to"; Tab(30); "learn"; Tab(35); "VB"
End sub
```

The Output for example 1 is shown below:



(ii) The Space function

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

Example 2

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

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

(iii) The Format function

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

(i) The format of the predefined Format function is

Format (n, "style argument")

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

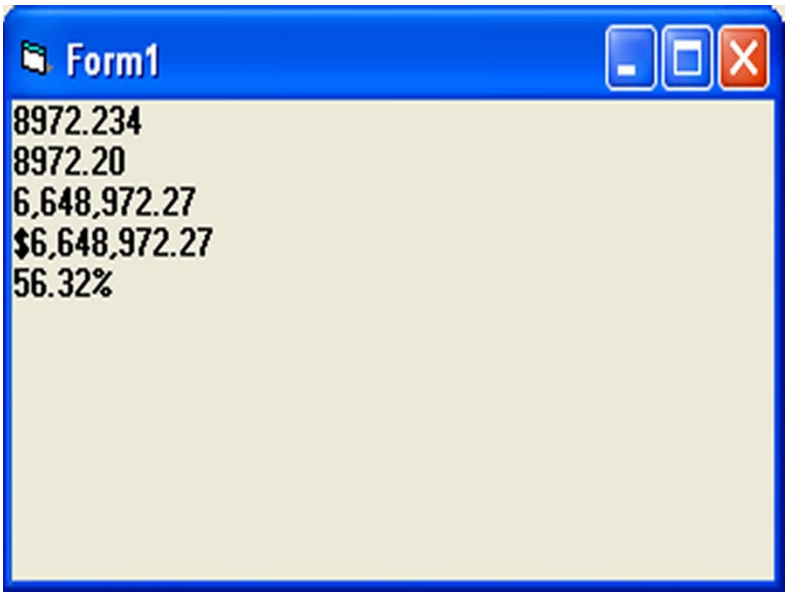
Style argument	**Explanation**	**Example**
General Number	To display the number without having separators between thousands.	Format(8972.234, "General Number")=8972.234
Fixed	To display the number without having separators between thousands and rounds it up to two decimal places.	Format(8972.2, "Fixed")=8972.23
Standard	To display the number with separators or separators between thousands and rounds it up to two decimal	Format(6648972.265, "Standard")= 6,648,972.27

	places.	
Currency	To display the number with the dollar sign in front, has separators between thousands as well as rounding it up to two decimal places.	Format(6648972.265, "Currency")= $6,648,972.27
Percent	Converts the number to the percentage form and displays a % sign and rounds it up to two decimal places.	Format(0.56324, "Percent")=56.32 %

Example 3

```
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
```

Now, run the program and you will get an output like the figure below:



In this lesson, we will learn how to use some of the string manipulation function such as Len, Right, Left, Mid, Trim, Ltrim, Rtrim, Ucase, Lcase, Instr, Val, Str ,Chr and Asc.

(i)The Len Function

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

Len ("Phrase")

For example,

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

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

```
Private sub Form_Activate ( )

X=sqr (16)

Y=1234

Z#=10#

Print Len(x), Len(y), and Len (z)

End Sub
```

will produce the output 1, 4 , 8. The reason why the last value is 8 is because z# is a double precision number and so it is allocated more memory spaces.

(ii) The Right Function

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

Right ("Phrase", n)

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

Right("Visual Basic", 4) = asic

(iii)The Left Function

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

Left("Phrase", n)

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

Left ("Visual Basic", 4) = Visu

(iv) The Ltrim Function

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

Ltrim("Phrase")

.For example,

Ltrim (" Visual Basic", 4)= Visual basic

(v) The Rtrim Function

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

Rtrim("Phrase")

.For example,

Rtrim ("Visual Basic ", 4) = Visual basic

(vi) The Trim function

The Ttrim function trims the empty spaces on both side of the phrase. The format is

Trim("Phrase")

.For example,

Trim (“ Visual Basic ”) = Visual basic

(viii) The Mid Function

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

Mid(phrase, position, n)

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

Mid(“Visual Basic”, 3, 6) = ual Bas

(ix) The InStr function

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

Instr (n, original phase, embedded phrase)

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

Instr(1, “Visual Basic”,” Basic”)=8

(x) The Ucase and the Lcase functions

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

Ucase(“Visual Basic”) =VISUAL BASiC

Lcase(“Visual Basic”) =visual basic

(xi) The Str and Val functions

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

(xii) The Chr and the Asc functions

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

Chr(charcode)

and the format of the Asc function is

Asc(Character)

The following are some examples:

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

Lesson 14: Creating User-Defined Functions

14.1 Creating Your Own Function

The general format of a function is as follows:

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

or

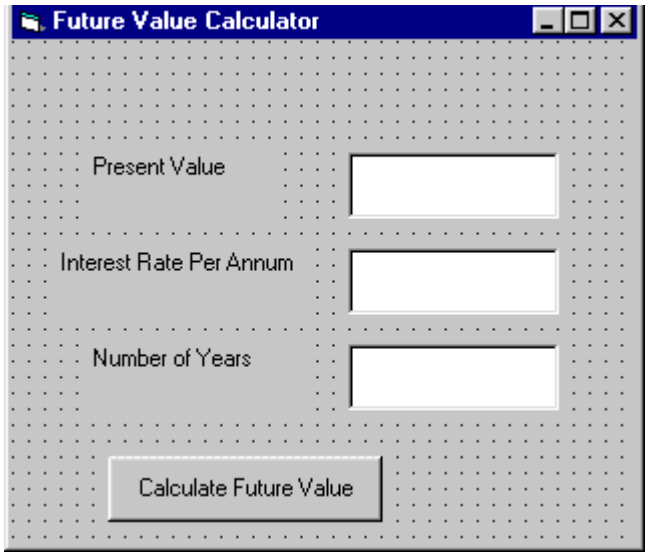
Private Function functionName (Arg As dataType,.....) As dataType

* 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.

Example 14.1

In this example, a user can calculate the future value of a certain amount of money he has today based on the interest rate and the number of years from now, supposing he will invest this amount of money somewhere .The calculation is based on the compound

interest rate.

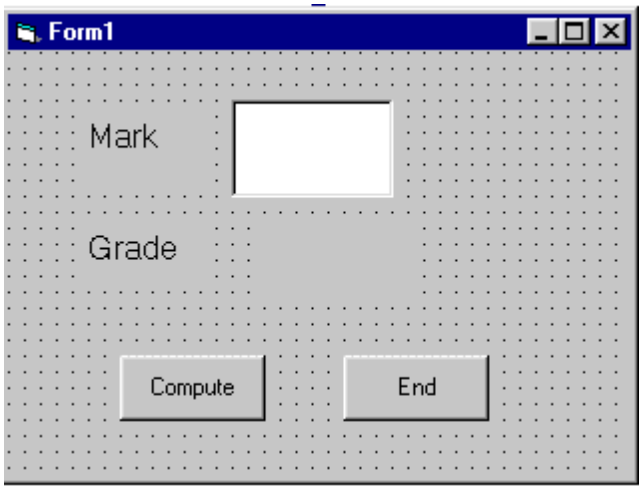


The code

```
Public Function FV(PV As Variant, i As Variant, n As Variant) As Variant

'Formula to calculate Future Value(FV)
'PV denotes Present Value
FV = PV * (1 + i / 100) ^ n

End Function

Private Sub compute_Click()

'This procedure will calculate Future Value
Dim FutureVal As Variant
Dim PresentVal As Variant
Dim interest As Variant
Dim period As Variant
PresentVal = PV.Text
interest = rate.Text
period = years.Text

'calling the funciton

FutureVal = FV(PresentVal, interest, period)
MsgBox ("The Future Value is " & FutureVal)

End Sub
```

Example 14.2	**The Code**

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




```
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
```

Lesson 15: Creating VBA Functions For MS Excel

15.1 The Needs to Create VBA Functions in MS-Excel

You can create your own functions to supplement the built-in functions in Microsoft Excel spreadsheet, which are quite limited in some aspects. These user-defined functions are also called **Visual Basic for Applications** functions, or simply VBA functions. They are very useful and powerful if you know how to program them properly. One main reason we need to create user defined functions is to enable us to customize our spreadsheet environment for individual needs. For example, we might need a function that could calculate commissions payment based on the sales volume, which is quite difficult if not impossible by using the built-in functions alone. The code for VBA is illustrated on the right.

Table 15.1: Commissions Payment Table

Sales Volume($)	Commissons
<500	3%

<1000	6%
<2000	9%
<5000	12%
>5000	15%

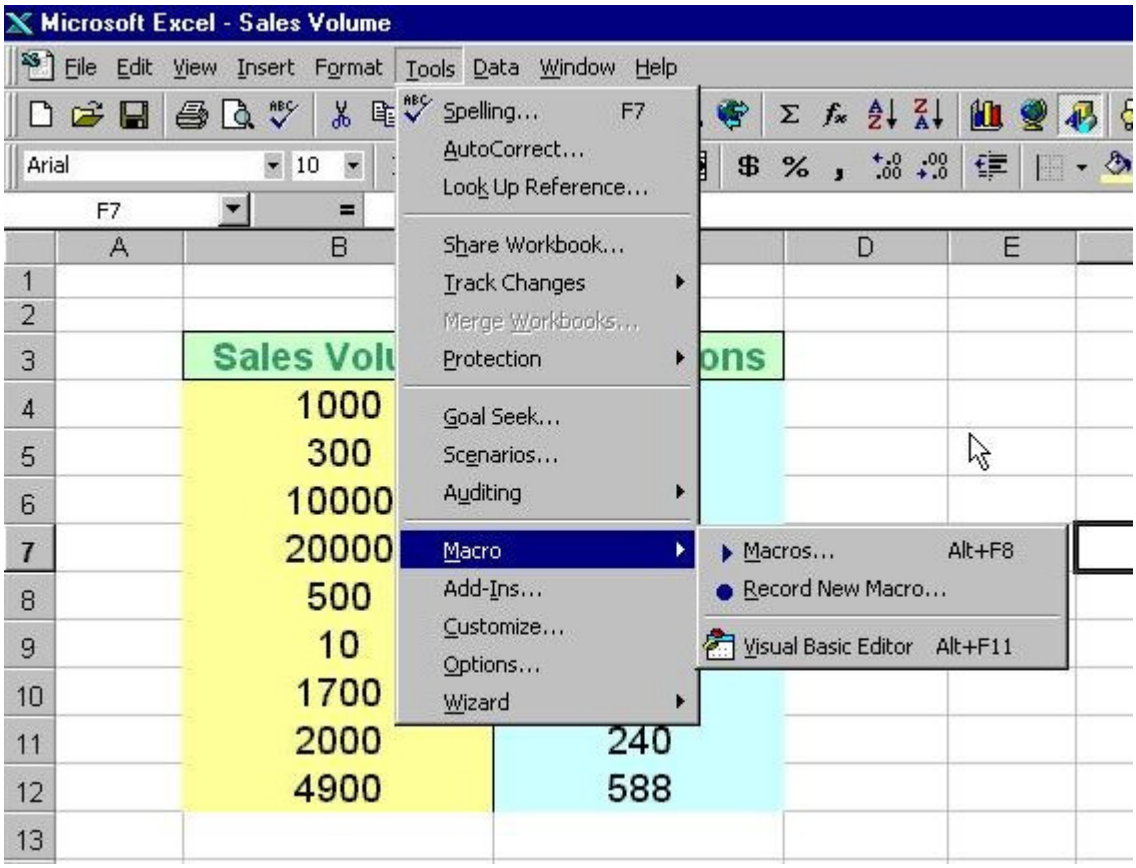
In table 15.1, when a salesman attain a sale volume of $6000, he will be paid $6000x15%=$720.00. A visual basic function to calculate the commissions can be written as follows:

```
Function Comm(Sales_V  As Variant) as Variant

If Sales_V <500 Then
Comm=Sales_V*0.03
Elseif Sales_V>=500 and Sales_V<1000 Then
Comm=Sales_V*0.06
Elseif Sales_V>=1000 and Sales_V<2000 Then
Comm=Sales_V*0.09
Elseif Sales_V>=200 and Sales_V<5000 Then
Comm=Sales_V*0.12
Elseif Sales_V>=5000  Then
Comm=Sales_V*0.15
End If

End Function
```

15.2 Using Microsoft Excel Visual Basic Editor

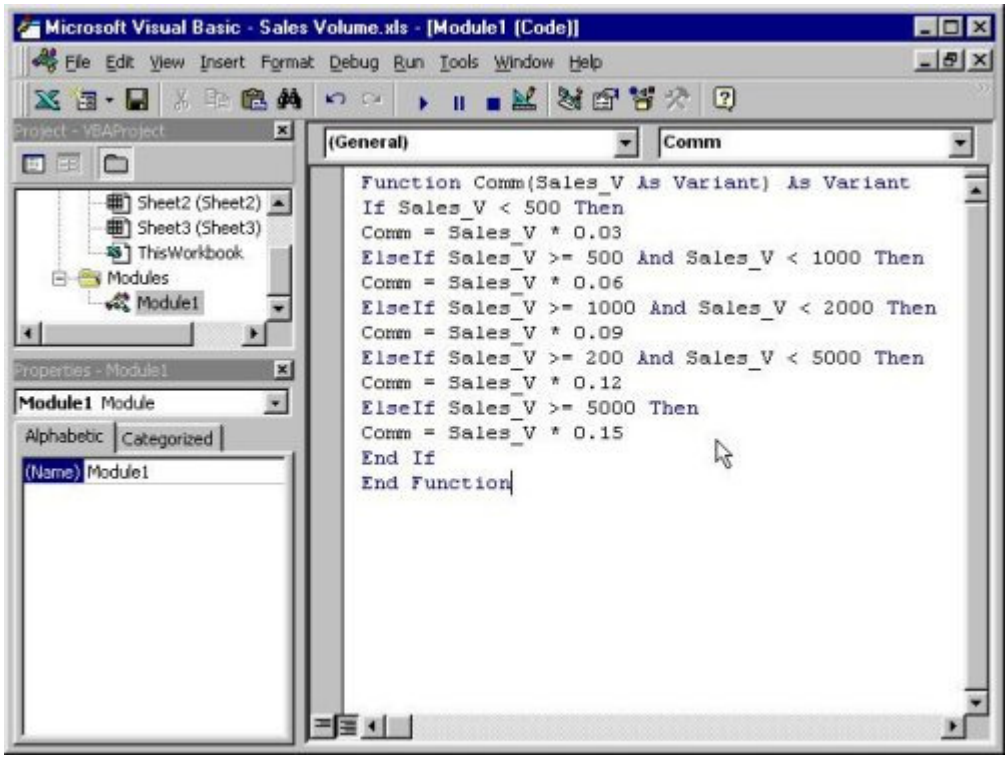
To create VBA functions in MS Excel, you can click on tools,
select macro and then click on Visual Basic Editor as shown in Figure 15.1

Figure 15.1: Inserting MS_Excel Visual Basic Editor



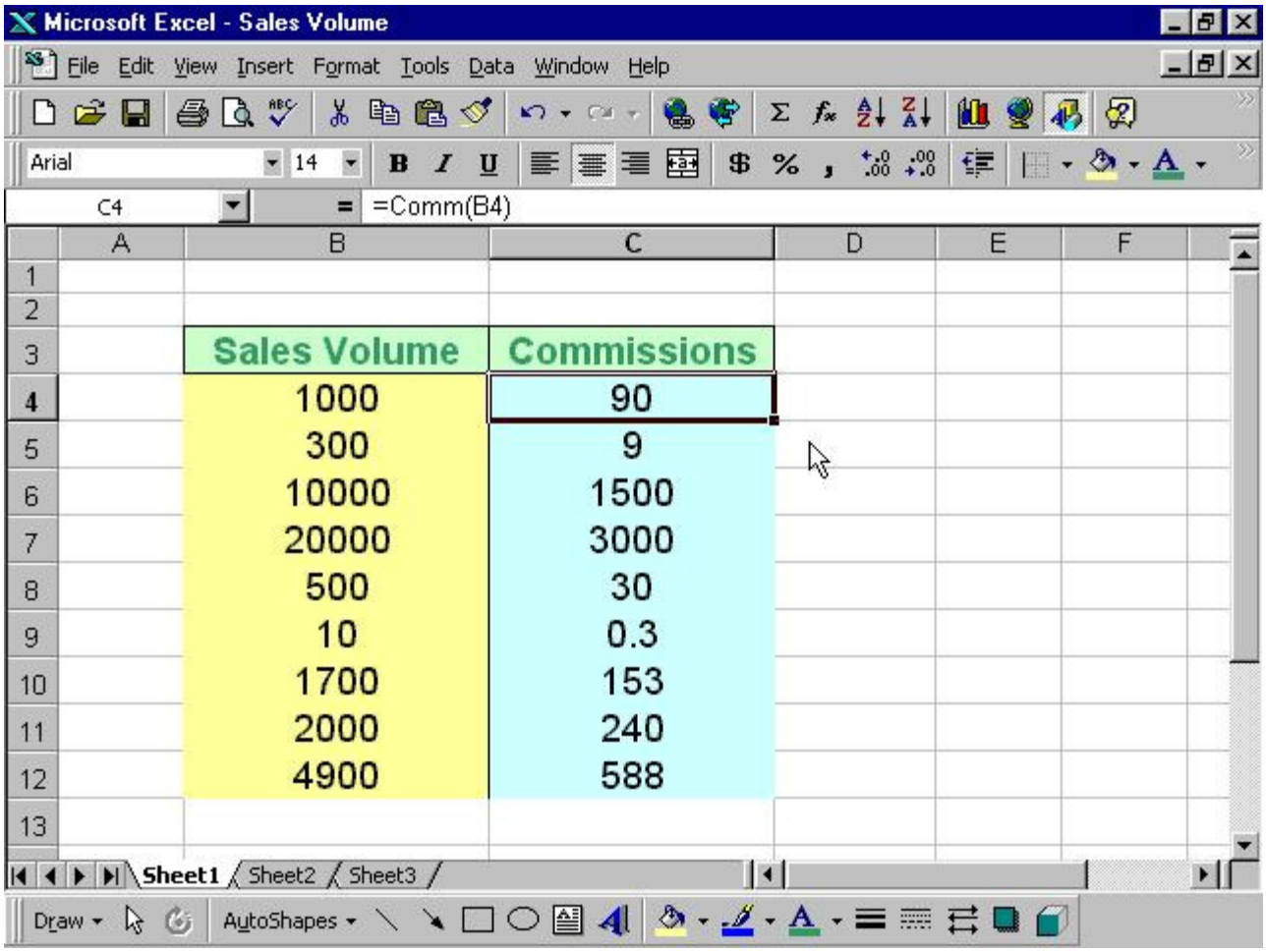
Upon clicking the Visual Basic Editor, the VB Editor windows will appear as shown in figure 15.2. To create a function, type in the function as illustrated in section 15.1 above After typing, save the file and then return to the Excel windows.

Figure 15.2 : The VB Editor



In the Excel window, type in the titles Sales Volume and Commissions in any two cells. By referring to figure 15.3, key-in the Comm function at cell C4 and by referencing the value in cell B4, using the format Comm(B4). Any value appear in cell B4 will pass the value to the Comm function in cell C4. For the rest of the rows, just copy the formula by dragging the bottom right corner of cell C4 to the required cells, a nice and neat table that shows the commissions will automatically appear (as shown in figure 15.3). It can also be updated anytime

Figure 15.3: MS Excel Windows- Sales Volume



Lesson 16: Arrays

16.1 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.

16.2 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.

Table 16.1. One dimensional Array

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

Table 16.2 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)

16.2 Declaring Arrays

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

The general format to declare a one dimensional array is as follow:

Dim arrayName(subs) as dataType

where subs indicates the last subscript in the array.

Example 16.1

Dim CusName(10) as String

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

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

Example 16.2

Dim Count(100 to 500) as Integer

declares an array that consists of the first element starting from Count(100) and ends at Count(500)

The general format to declare a two dimensional array is as follow:

Dim ArrayName(Sub1,Sub2) as dataType

Example 16.3

Dim StudentName(10,10) will declare a 10x10 table make up of 100 students' Names, starting with StudentName(1,1) and end with StudentName(10,10).

16.3 Sample Programs

(i) **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
```

The above program accepts data entry through an input box and displays the entries in the form itself. As you can see, this program will only allows a user to enter 10 names each time he click on the start button.

(ii)

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")
List1.AddItem
studentName(num)
Next
End Sub
Private Sub
Start_Click()
addName

End Sub
```

The above program accepts data entries through an InputBox and displays the items in a list box.

Lesson 17: Working with Files

17.1 Introduction

Up until lesson 13 we are only creating programs that could accept data at runtime, when the program is terminated, the data also disappear. Is it possible to save data accepted by a VB program into a storage device, such as a hard disk or diskette, or even CDRW? The answer is possible. In this chapter, we will learn how to create files by writing them into a storage device and then retrieve the data by reading the contents of the files using a customized VB program.

17.2 Creating files

To create a file , we use the following command

Open "fileName" For Output As #fileNumber

Each file created must have a file name and a file number for identification. As for the file name, you must also specify the path where the file will reside.

Examples:

Open "c:\My Documents\sample.txt" For Output As #1

will create a text file by the name of sample.txt in My Document folder. The accompany file number is 1. If you wish to create and save the file in A drive, simply change the path, as follows"

Open "A:\sample.txt" For Output As #1

If you wish to create a HTML file , simply change the extension to .html

Open "c:\My Documents\sample.html" For Output As # 2

17.2.1 Sample Program : Creating a text file

```
Private Sub create_Click()

Dim intMsg As String
Dim StudentName As String

Open "c:\My Documents\sample.txt" For Output As #1
intMsg = MsgBox("File sample.txt opened")
StudentName = InputBox("Enter the student Name")
Print #1, StudentName
intMsg = MsgBox("Writing a" & StudentName & " to sample.txt ")

Close #1

intMsg = MsgBox("File sample.txt closed")

End Sub
```

* The above program will create a file sample.txt in the My Documents' folder and ready to receive input from users. Any data input by users will be saved in this text file.

17.3 Reading a file

To read a file created in section 17.2, you can use the input # statement. However, we can only read the file according to the format when it was written. You have to open the file according to its file number and the variable that hold the data. We also need to declare the variable using the DIM command.

17.3.1 Sample Program: Reading file

```
Private Sub Reading_Click()
Dim variable1 As String
Open "c:\My Documents\sample.txt" For Input As #1
Input #1, variable1
Text1.Text = variable1
Close #1

End Sub
```

* This program will open the sample.txt file and display its contents in the Text1 textbox.

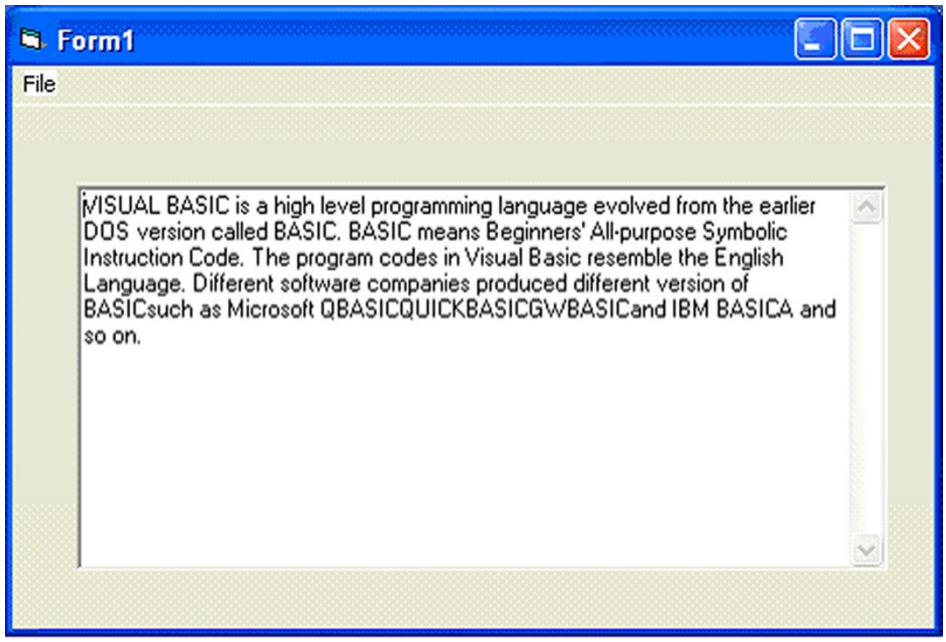
Example 17.3.2 Creating and Reading files using Common Dialog Box

This example uses the common dialog box to create and read the text file, which is much easier than the previous examples as many operations are handled by the common dialog box. The following is the program:

```
Dim linetext As String
Private Sub open_Click()
CommonDialog1.Filter = "Text files{*.txt)|*.txt"
CommonDialog1.ShowOpen

If CommonDialog1.FileName <> "" Then
Open CommonDialog1.FileName For Input As #1
Do
Input #1, linetext
Text1.Text = Text1.Text & linetext
Loop Until EOF(1)
End If
Close #1
End Sub
Private Sub save_Click()
CommonDialog1.Filter = "Text files{*.txt)|*.txt"
CommonDialog1.ShowSave
If CommonDialog1.FileName <> "" Then
Open CommonDialog1.FileName For Output As #1
Print #1, Text1.Text
Close #1
End If
End Sub
```

The syntax **CommonDialog1.Filter = "Text files{*.txt)|*.txt"** ensures that only the textfile is read or saved .The statement CommonDialog1.ShowOpen is to display the open file dialog box and the statement **CommonDialog1.ShowSave** is to display thesave file dialog box. Text1.Text = Text1.Text & linetext is to read the data and display them in the Text1 textbox

The Output window is shown below:



Lesson 18: Graphics

raphics is a very important part of visual basic programming as an attractive interface will be appealing to the users. In the old BASIC, drawing and designing graphics are considered as difficult jobs, as they have to be programmed line by line in a text-based environment. However, in Visual Basic, these jobs have been made easy. There are four basic controls in VB that you can use to draw graphics on your form: the line control, the shape control, the image box and the picture box

18.1The line and Shape controls

To draw a straight line, just click on the line control and then use your mouse to draw the line on the form. After drawing the line, you can then change its color, width and style using the BorderColor, BorderWidth and BorderStyle properties.

Similarly, to draw a shape, just click on the shape control and draw the shape on the form. The default shape is a rectangle, with the shape property set at 0. You can change the shape to square, oval, circle and rounded rectangle by changing the shape property's value to 1, 2, 3 4, and 5 respectively. In addition, you can change its background color using the

BackColor property, its border style using the BorderStyle property, its border color using the BorderColor pproperty as well its border width using the BorderWidth property.

Example 18.1

The program in this example allows the user to change the shape by selecting a particular shape from a list of options from a list box, as well as changing its color through a common dialog box.

The objects to be inserted in the form are a list box, a command button, a shape control and a common dialog box. The common dialog box can be inserted by clicking on 'project' on the menu and then select the Microsoft Common Dialog Control 6.0 by clicking the check box. After that, the Microsoft Common Dialog Control 6.0 will appear in the toolbox; and you can drag it into the form. The list of items can be added to the list box through the AddItem method. The procedure for the common dialog box to present the standard colors is as follows:

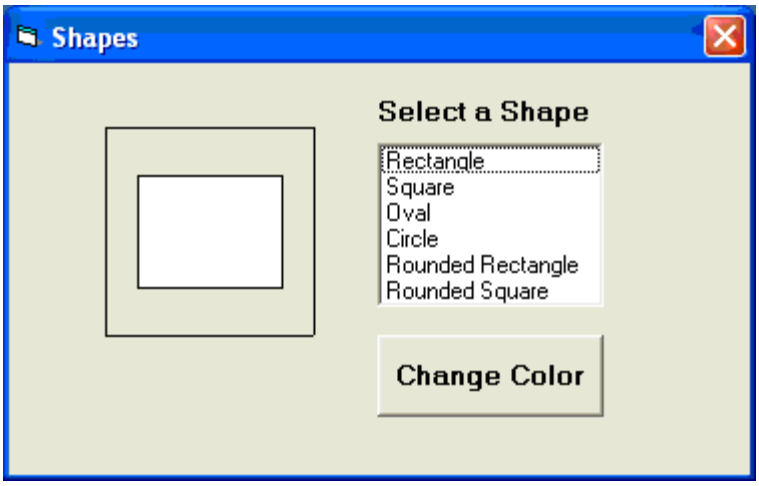
CommonDialog1.Flags = &H1&

CommonDialog1.ShowColor

Shape1.BackColor = CommonDialog1.Color

The last line will change the background color of the shape by clicking on a particular color on the common dialog box as shown in the Figure below:

The Interface.



The color dialog box

he Code

```
Private Sub Form_Load()
List1.AddItem "Rectangle"
List1.AddItem "Square"
List1.AddItem "Oval"
List1.AddItem "Circle"
List1.AddItem "Rounded Rectangle"
List1.AddItem "Rounded Square"
End Sub

Private Sub List1_Click()
Select Case List1.ListIndex
Case 0
Shape1.Shape = 0
Case 1
Shape1.Shape = 1
Case 2
Shape1.Shape = 2
Case 3
```

```
Shape1.Shape = 3
Case 4
Shape1.Shape = 4
Case 5
Shape1.Shape = 5
End Select
End Sub

Private Sub Command1_Click()
CommonDialog1.Flags = &H1&
CommonDialog1.ShowColor
Shape1.BackColor = CommonDialog1.Color
End Sub
```

18.2 The Image Box and the Picture Box

Using the line and shape controls to draw graphics will only enable you to create a simple design. In order to improve the look of the interface, you need to put in images and pictures of your own. Fortunately, there are two very powerful graphics tools you can use in Visual Basic which are the image box and the picture box.

To load a picture or image into an image box or a picture box, you can click on the picture property in the properties window and a dialog box will appear which will prompt the user to select a certain picture file. You can also load a picture at runtime by using the LoadPictrure ( ) method. The syntax is

Image1.Picture= LoadPicture("C:\path name\picture file name") or

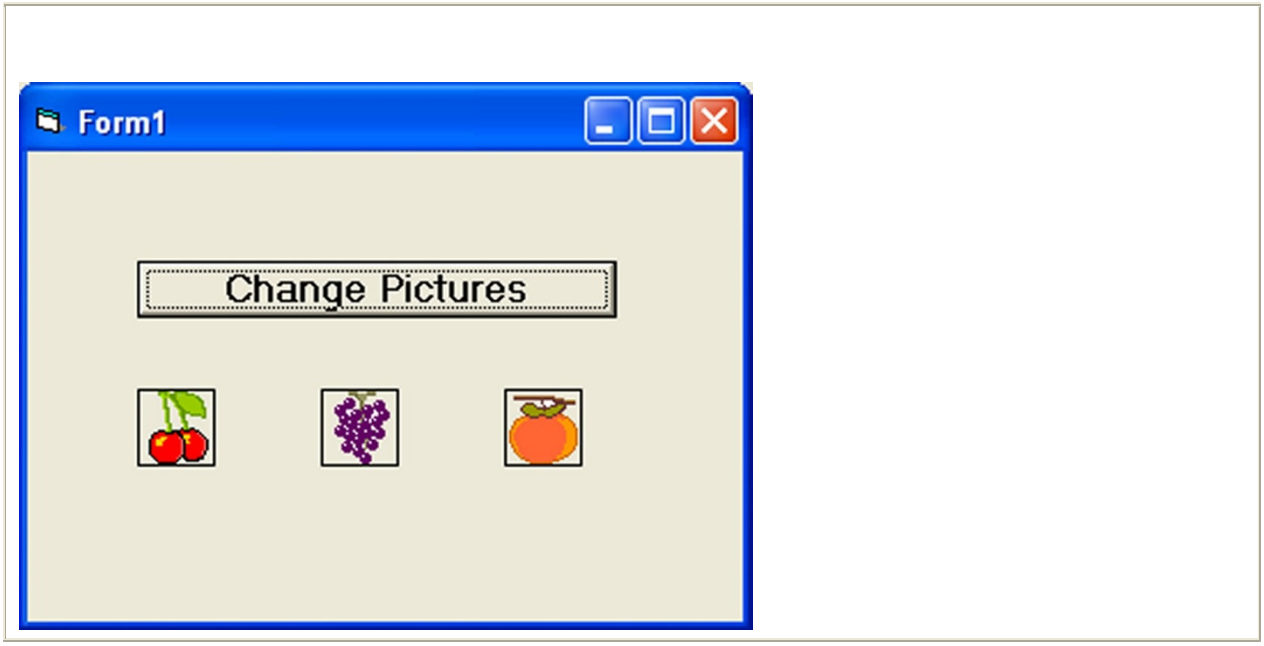
picture1.Picture= LoadPicture("C:\path name\picture name")

For example, the following statement will load the grape.gif picture into the image box.

Image1.Picture= LoadPicture("C:\My Folder\VB program\Images\grape.gif")

Example 18.2

In this example, each time you click on the 'change pictures' button as shown in Figure 19.2, you will be able to see three images loaded into the image boxes. This program uses the Rnd function to generate random integers and then uses the LoadPicture method to load different pictures into the image boxes using the If…Then…Statements based on the random numbers generated. The output is shown in Figure 19.2 below




```
Dim a, b, c As Integer
Private Sub Command1_Click ()
Randomize Timer
a = 3 + Int(Rnd * 3)
b = 3 + Int(Rnd * 3)
c = 3 + Int(Rnd * 3)

If a = 3 Then
Image1(0).Picture = LoadPicture("C:\My Folder\VB program\Images\grape.gif")
End If
If a = 4 Then
Image1(0).Picture = LoadPicture("C:\My Folder\VB program\Images\cherry.gif")
End If
If a = 5 Then
Image1(0).Picture = LoadPicture("C:\My Folder\VB program\Images\orange.gif")
End If
If b = 3 Then
Image1(1).Picture = LoadPicture("C:\My Folder\VB program\Images\grape.gif")
End If
If b = 4 Then
Image1(1).Picture = LoadPicture("C:\My Folder\VB program\Images\cherry.gif")
```

```
End If
If b = 5 Then
Image1(1).Picture = LoadPicture("C:\My Folder\VB program\Images\orange.gif")
End If
If c = 3 Then
Image1(2).Picture = LoadPicture("C:\My Folder\VB program\Images\grape.gif")
End If
If c = 4 Then
Image1(2).Picture = LoadPicture("C:\My Folder\VB program\Images\cherry.gif")
End If
If c = 5 Then
Image1(2).Picture = LoadPicture("C:\My Folder\VB program\Images\orange.gif")
End If
End Sub
```

18.3 PSet, Line and Circle Drawing Methods

Other than using the line and shape controls to draw graphics on the form, you can also use the Pset, Line and Circle methods to draw graphics on the form.

(a) The Pset Method

The Pset method draw a dot on the screen, it takes the format

Pset (x , y), color

(x,y) is the coordinates of the point and color is its color. To specify the color, you can use the color codes or the standard VB color constant such as VbRed, VbBlue, VbGeen and etc. For example, Pset(100,200), VbRed will display a red dot at the (100,200) coordinates.

The Pset method can also be used to draw a straight line on the form. The procedure is

```
For x= a to b
Pset(x,x)
Next x
```

This procedure will draw a line starting from the point (a,a) and to the point (b,b). For example, the following procedure will draw a magenta line from the point (0,0) to the point (1000,1000).

```
For x= 0 to 100
Pset(x,x) , vbMagenta
Next x
```

(b) The Line Method

Although the Pset method can be used to draw a straight line on the form, it is a little slow. It is better to use the Line method if you want to draw a straight line faster. The format of the Line command is shown below. It draws a line from the point (x1, y1) to the point (x2, y2) and the color constant will determine the color of the line.

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

For example, the following command will draw a red line from the point (0, 0) to the point (1000, 2000).

```
Line (0, 0)-(1000, 2000), VbRed
```

The Line method can also be used to draw a rectangle. The format is

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

The four corners of the rectangle are (x1-y1), (x2-y1), (x1-y2) and (x2, y2)

Another variation of the Line method is to fill the rectangle with a certain color. The format is

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

If you wish to draw the graphics in a picture box, you can use the following formats

- Picture1.Line (x1, y1)-(x2, y2), color
- Picture1.Line (x1-y1)-(x2, y2), color, B
- Picture1.Line (x1-y1)-(x2, y2), color, BF
- Picture1.Circle (x1, y1), radius, color

(c) The Circle Method

The circle method takes the following format

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

That draws a circle centered at (x1, y1), with a certain radius and a certain border color. For example, the procedure

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

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

Lesson 19: Creating Multimedia Applications- Part I

To be able to play multimedia files or multimedia devices, you have to insert **Microsoft Multimedia Control** into your VB applications that you are going to create. However, Microsoft Multimedia Control is not normally included in the startup toolbox, therefore you need to add the **MM control** by pressing Ctrl+T and select it from the components dialog box that is displayed.

19.1 Creating a CD player

In this program, you can create a CD player that resembles an actual CD player. It allows the user select a track to play, to fast forward, to rewind and also to eject the CD. It can also display the track being played. The interface and code are shown below.

You can create various multimedia applications in VB that could play audio CD, audiofiles, VCD , video files and more.

The Interface.

The Code

```
Private Sub Form_Load()
'To position the page at the center
Left = (Screen.Width - Width) \ 2
Top = (Screen.Height - Height) \ 2
'Initialize the CD
myCD.Command = "Open"
End Sub

Private Sub myCD_StatusUpdate()
```

```
'Update the track number
trackNum.Caption = myCD.Track
End Sub

Private Sub Next_Click()
myCD.Command = "Next"
End Sub

Private Sub Play_Click()
myCD.Command = "Play"
End Sub

Private Sub Previous_Click()
myCD.Command = "Prev"
End Sub

Private Sub Stop_Click()
myCD.Command = "Stop"
End Sub
```

Lesson 20: Creating Multimedia Applications-Part II

In previous lesson, we have programmed a CD player. Now, by making some modifications, you canl transform the CD player into an audio player. This player will be created in such a way that it could search for wave and midi files in your drives and play them.

In this project, you need to insert a **ComboBox**, a **DriveListBox**, a **DirListBox**, a **TextBox** and a **FileListBox** into your form. I Shall briefly discuss the function of each of the above controls. Besides, you must also insert **Microsoft Multimedia Control(MMControl)** into your form , you may make it visible or invisible. In my program, I choose to make it invisible so that I can use the command buttons created to control the player.

- ComboBox- to display and enable selection of different type of files.
- DriveListBox- to allow selection selection of different drives available on your PC.
- DirListBox - To display directories
- TextBox - To display selected files
- FileListBox- To display files that are available
- Relevant code must be written to coordinate all the above controls so that the application can work properly. The program should follow in the following logical way:
- Step 1: User chooses the type of files he wants to play.
- Step2:User selects the drive that might contains the relevant audio files.
- Step 3:User looks into directories and subdirectories for the files specified in step1. The files should be displayed in the **FileListBox**.
- Step 4: User selects the files from the **FileListBox** and click the **Play** button.

- Step 5: User clicks on the **Stop** button to stop playing and **Exit** button to end the application.
- **The Interface**
- 

The Code

```
Private Sub Combo1_Change()

' to determine file type

If ListIndex = 0 Then
File1.Pattern = ("*.wav")
ElseIf ListIndex = 1 Then
File1.Pattern = ("*.mid")
Else
Fiel1.Pattern = ("*.*")
End If
End Sub


Private Sub Dir1_Change()
'To change directories and subdirectories(or folders and subfolders)
File1.Path = Dir1.Path
If Combo1.ListIndex = 0 Then
File1.Pattern = ("*.wav")
ElseIf Combo1.ListIndex = 1 Then
File1.Pattern = ("*.mid")
Else
File1.Pattern = ("*.*")
End If
```

```
End Sub


Private Sub Drive1_Change()
'To change drives
Dir1.Path = Drive1.Drive
End Sub

Private Sub File1_Click()
If Combo1.ListIndex = 0 Then
File1.Pattern = ("*.wav")
ElseIf Combo1.ListIndex = 1 Then
File1.Pattern = ("*.mid")
Else
File1.Pattern = ("*.*")
End If

If Right(File1.Path, 1) <> "\" Then
filenam = File1.Path + "\" + File1.FileName
Else
filenam = File1.Path + File1.FileName
End If
Text1.Text = filenam

End Sub


Private Sub Form_Load()
'To center the Audioplayer startup page
Left = (Screen.Width - Width) \ 2
Top = (Screen.Height - Height) \ 2
Combo1.Text = "*.wav"
Combo1.AddItem "*.wav"
Combo1.AddItem "*.mid"
Combo1.AddItem "All files"
End Sub


Private Sub play_Click()
'To play WaveAudio file or Midi File
Command2_Click
If Combo1.ListIndex = 0 Then
AudioPlayer.DeviceType = "WaveAudio"
ElseIf Combo1.ListIndex = 1 Then
AudioPlayer.DeviceType = "Sequencer"
End If
AudioPlayer.FileName = Text1.Text
AudioPlayer.Command = "Open"
AudioPlayer.Command = "Play"
```

```
End Sub

Private Sub stop_Click()
If AudioPlayer.Mode = 524 Then Exit Sub
If AudioPlayer.Mode <> 525 Then
AudioPlayer.Wait = True
AudioPlayer.Command = "Stop"
End If
AudioPlayer.Wait = True
AudioPlayer.Command = "Close"

End Sub
```

Lesson 21: Creating Multimedia Applications-Part III

In lesson 20, we have created an audio player. Now, by making further modifications, you can transform the audio player into a picture viewer. This viewer will be created in such a way that it could search for all types of graphics files in your drives and displays them in a picture frame.

Similar to the previous project, in this project, you need to insert a **ComboBox**, a **DriveListBox**, a **DirListBox**, a **TextBox** and a **FileListBox** into your form. I Shall briefly explain again the function of each of the above controls.

- ComboBox- to display and enable selection of different type of files.
- DriveListBox- to allow selection selection of different drives available on your PC.
- DirListBox - To display directories
- TextBox - To display selected files
- FileListBox- To display files that are availabl

Relevant codes must be written to coordinate all the above controls so that the application can work properly. The program should flow in the following logical way:

Step 1: User chooses the type of files he wants to play.

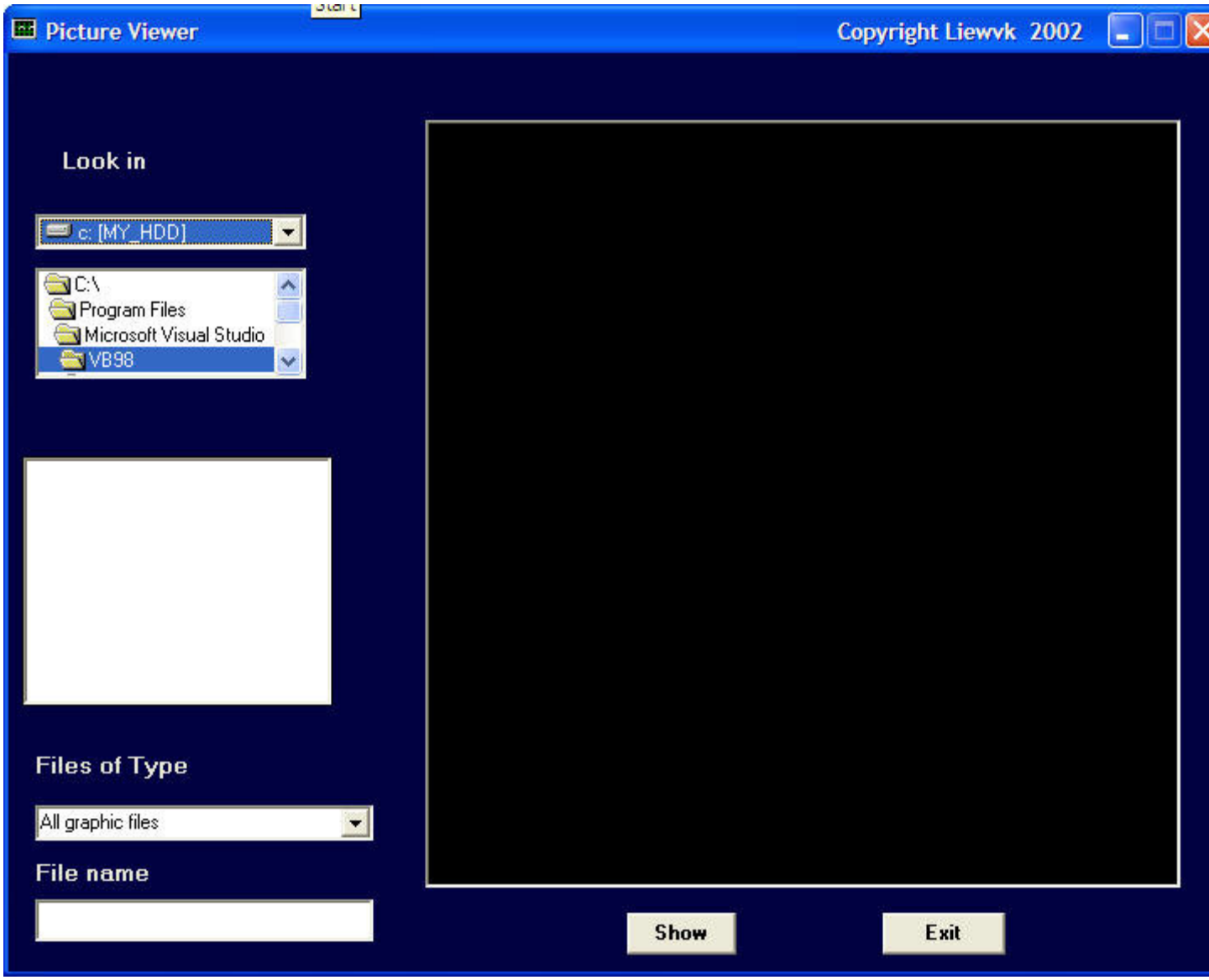
Step2:User selects the drive that might contains the relevant graphic files.

Step 3:User looks into directories and subdirectories for the files specified in step1. The files should be displayed in the FileListBox.

Step 4: User selects the files from the FileListBox and click the Show button.

Step 5: User clicks on Exit button to end the application.

The Interface



The Code

```
Private Sub Form_Load()

'To center the player
Left = (Screen.Width - Width) \ 2
Top = (Screen.Height - Height)\2


Combo1.Text = "All graphic files"
Combo1.AddItem "All graphic files"
Combo1.AddItem "All files"

End Sub
```

```
Private Sub Combo1_Change()

If ListIndex = 0 Then
File1.Pattern = ("*.bmp;*.wmf;*.jpg;*.gif")
Else
Fiel1.Pattern = ("*.*")
End If

End Sub

'Specific the types of files to load
Private Sub Dir1_Change()

File1.Path = Dir1.Path
File1.Pattern = ("*.bmp;*.wmf;*.jpg;*.gif")

End Sub

'Changing Drives

Private Sub Drive1_Change()

Dir1.Path = Drive1.Drive

End Sub

Private Sub File1_Click()

If Combo1.ListIndex = 0 Then
File1.Pattern = ("*.bmp;*.wmf;*.jpg;*.gif")
Else
File1.Pattern = ("*.*")
EnId If

If Right(File1.Path, 1) <> "\" Then
filenam = File1.Path + "\" + File1.FileName
Else
filenam = File1.Path + File1.FileName
End If
Text1.Text = filenam

End Sub


Private Sub show_Click()

If Right(File1.Path, 1) <> "\" Then
filenam = File1.Path + "\" + File1.FileName
Else
filenam = File1.Path + File1.FileName
End If
```

```
'To load the picture into the picture box
picture1.Picture = LoadPicture(filenam)

End Sub
```

Lesson 22: Creating Multimedia Applications- Part IV:

A Multimedia Player

In lesson 20, we have created an audio player. Now, by making more modifications, you can transform the audio player into a multimedia player. This player will be able to search for all types of movie files and audio files. your drives and play them.

In this project, you need to insert a **ComboBox**, a **DriveListBox**, a **DirListBox**, a **TextBox** ,a **FileListBox** and a **picture box** (for playing movie) into your form. I Shall briefly discuss the function of each of the above controls. Besides, you must also insert **Microsoft Multimedia Control(MMControl**) into your form , you may make it visible or invisible. In my program, I choose to make it invisible so that I could use the command buttons created to control the player.

- ComboBox- to display and enable selection of different type of files.
- DriveListBox- to allow selection selection of different drives available on your PC.
- DirListBox - To display directories
- TextBox - To display selected files
- FileListBox- To display files that are available

elevant codes must be written to coordinate all the above controls so that the application can work properly. The program should flow in the following logical way:

Step 1: User chooses the type of files he wants to play.

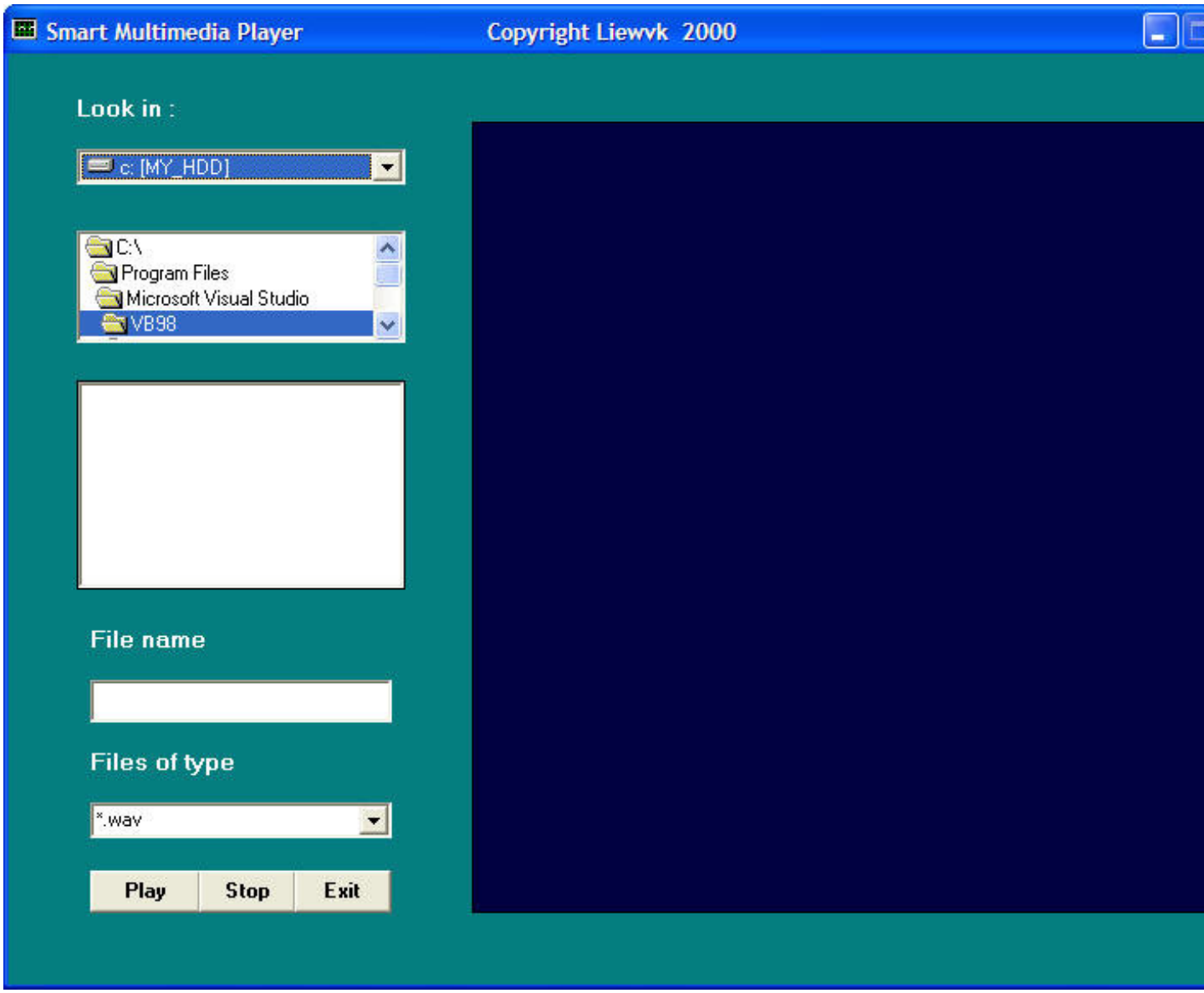
Step2:User selects the drive that might contains the relevant audio files.

Step 3:User looks into directories and subdirectories for the files specified in step1. The files should be displayed in the **FileListBox**.

Step 4: User selects the files from the **FileListBox** and clicks the **Play** button.

Step 5: User clicks on the **Stop** button to stop playing and Exit button to end the application.

The Interface



The Code

```
Private Sub Form_Load()

Left = (Screen.Width - Width) \ 2
Top = (Screen.Height - Height) \ 2
Combo1.Text = "*.wav"
Combo1.AddItem "*.wav"
Combo1.AddItem "*.mid"
Combo1.AddItem "*.avi;*.mpg"
Combo1.AddItem "All files"

End Sub

Private Sub Combo1_Change()
```

```
If ListIndex = 0 Then
File1.Pattern = ("*.wav")
ElseIf ListIndex = 1 Then
File1.Pattern = ("*.mid")
ElseIf ListIndex = 2 Then
File1.Pattern = ("*.avi;*.mpg")
Else
Fiel1.Pattern = ("*.*")
End If
End Sub

Private Sub Dir1_Change()
File1.Path = Dir1.Path
If Combo1.ListIndex = 0 Then
File1.Pattern = ("*.wav")
ElseIf Combo1.ListIndex = 1 Then
File1.Pattern = ("*.mid")
ElseIf Combo1.ListIndex = 2 Then
File1.Pattern = ("*.avi;*.mpg")
Else
File1.Pattern = ("*.*")
End If
End Sub
Private Sub Drive1_Change()
Dir1.Path = Drive1.Drive
End Sub


Private Sub File1_Click()
If Combo1.ListIndex = 0 Then
File1.Pattern = ("*.wav")
ElseIf Combo1.ListIndex = 1 Then
File1.Pattern = ("*.mid")
ElseIf Combo1.ListIndex = 2 Then
File1.Pattern = ("*.avi;*.mpg")
Else
File1.Pattern = ("*.*")
End If

If Right(File1.Path, 1) <> "\" Then
filenam = File1.Path + "\" + File1.FileName
Else
filenam = File1.Path + File1.FileName
End If
Text1.Text = filenam
End Sub

Private Sub play_Click()
MMPlayer.FileName = Text1.Text
MMPlayer.Command = "Open"
```

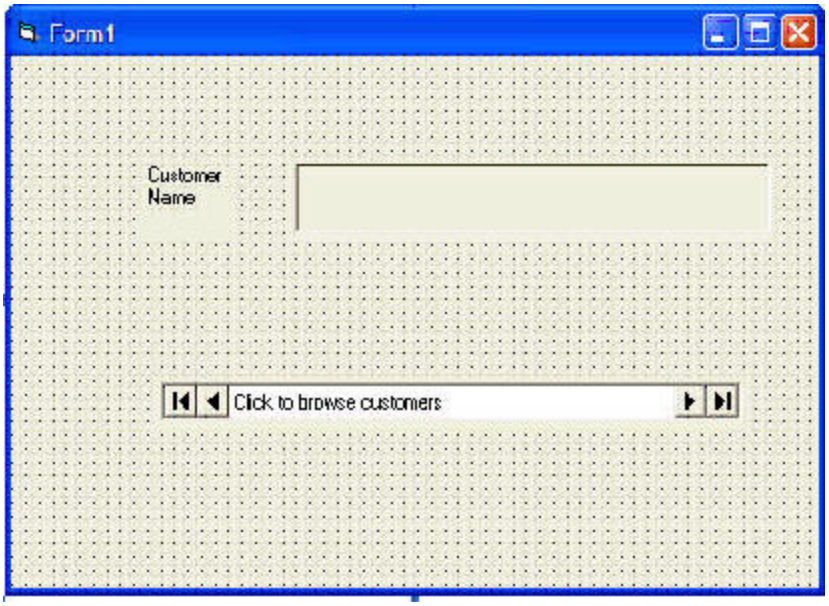
```
MMPlayer.Command = "Play"
MMPlayer.hWndDisplay = videoscreen.hWnd
End Sub


Private Sub stop_Click()
If MMPlayer.Mode = 524 Then Exit Sub
If MMPlayer.Mode <> 525 Then
MMPlayer.Wait = True
MMPlayer.Command = "Stop"
End If
MMPlayer.Wait = True
MMPlayer.Command = "Close"
End Sub
```

Lesson 23: 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. In the following example, we will create a simple database application which enable one to browse customers' names. To create this application, insert the data control into the new form. Place the data control somewhere at the bottom of the form. Name the data control as data_navigator. To be able to use the data control, we need to connect it to any database. We can create a database file using any database application but I suggest we use the database files that come with VB6. Let select NWIND.MDB as our database file.

To connect the data control to this database, double-click the DatabaseName property in the properties window and select the above file, i.e NWIND.MDB. Next, double-click on the RecordSource property to select the customers table from the database. You can also change the caption of the data control to anything but I use "Click to browse Customers" here. After that, we will place a label and change its caption to Customer Name. Last but not least, insert another label and name it as cus_name and leave the label empty as customers' names will appear here when we click the arrows on the data control. We need to bind this label to the data control for the application to work. To do this, open the label's DataSource and select data_navigator that will appear automatically. One more thing that we need to do is to bind the label to the correct field so that data in this field will appear on this label. To do this, open the DataField property and select ContactName. Now, press F5 and run the program. You should be able to browse all the customers' names by clicking the arrows on the data control.

The Design Interface.



The Runtime Interface

You can also add other fields using exactly the same method. For example, you can add adress, City and telephone number to the database browser.

Lesson 24: Creating database applications in VB-Part II

In Lesson 23, you have learned how to create a simple database application using data control. In this lesson, you will work on the same application but use some slightly more advance commands. The data control support some methods that are useful in manipulating the database, for example, to move the pointer to a certain location. The following are some of the commands that you can use to move the pointer around:

data_navigator.RecordSet.MoveFirst ' Move to the first record

data_navigator.RecordSet.MoveLast ' Move to the last record

data_navigator.RecordSet.MoveNext ' Move to the next record

data_navigator.RecordSet.Previous ' Move to the first record

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

data_navigator.RecordSet.AddNew ' Adds a new record

data_navigator.RecordSet.Update ' Updates and saves the new record

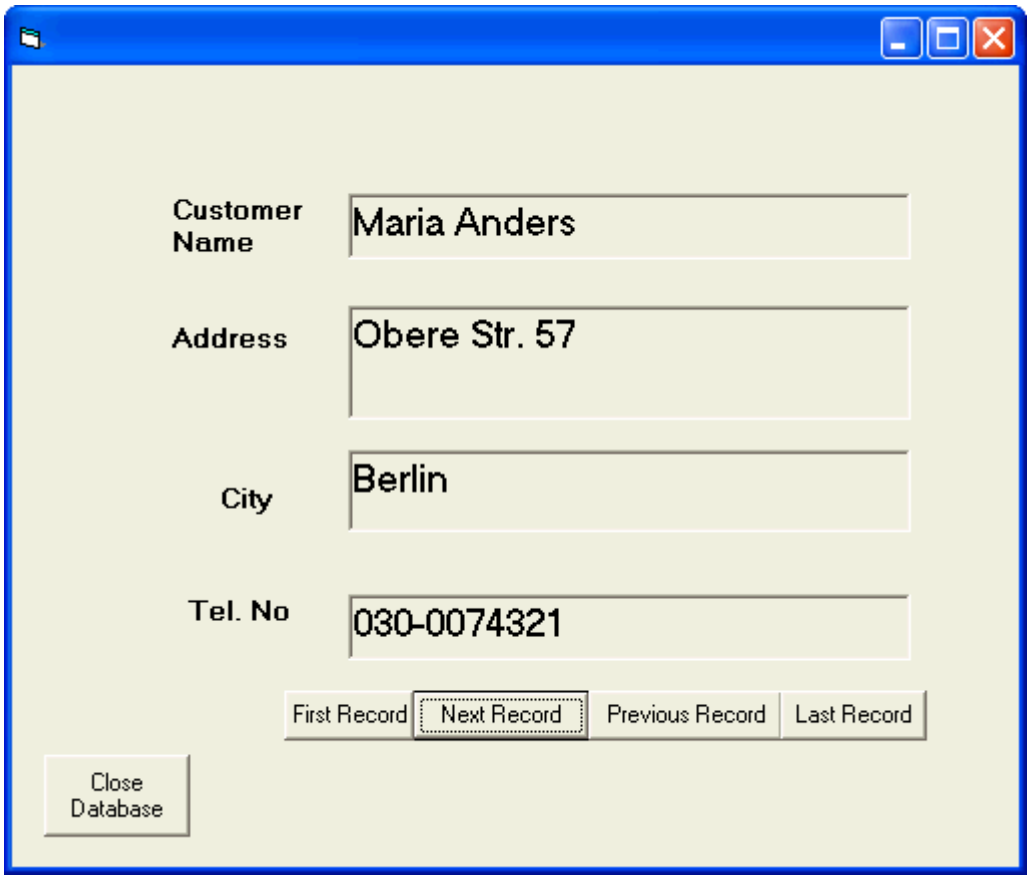
data_navigator.RecordSet.Delete ' Deletes a current record

*note: data_navigator is the name of data control

In the following example, you shall insert four commands and label them as First Record, Next Record, Previous Record and Last Record . They will be used to navigator around the database without using the data control. You still need to retain the same data control (from example in lesson 19) but set the property Visible to no so that users will not see the data control but use the button to browse through the database instead. Now, double-click on the command button and key in the codes according to the labels.

```
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
```

Run the application and you shall obtain the interface below and you will be able to browse the database using the four command buttons.

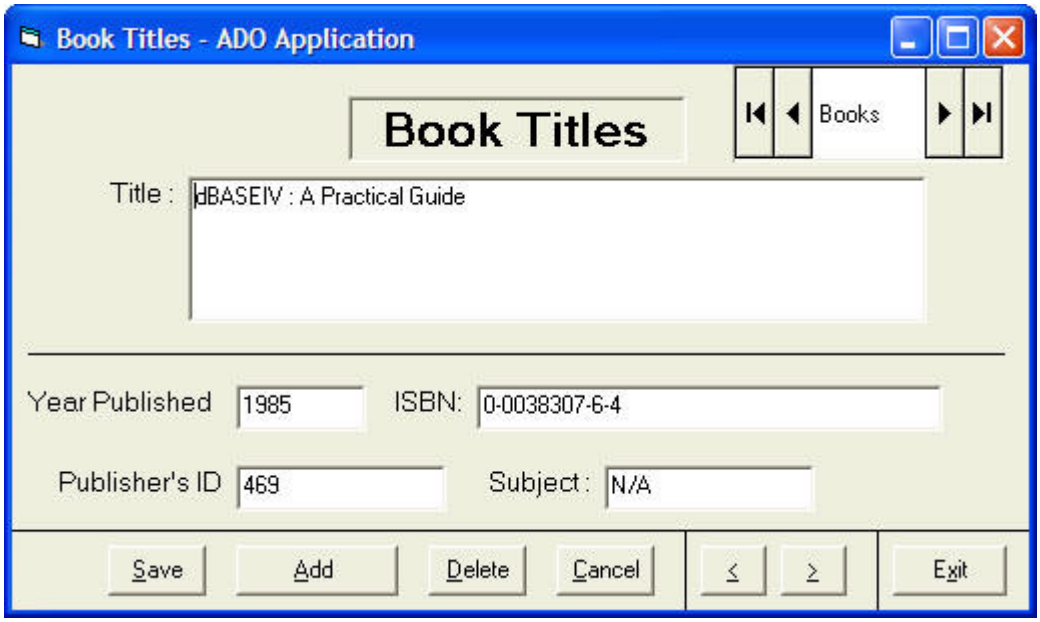


Lesson 25: Creating VB database applications using ADO control

In Lesson 22 and Lesson 23, we have learned how 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.



The properties of all the controls are listed as follow:

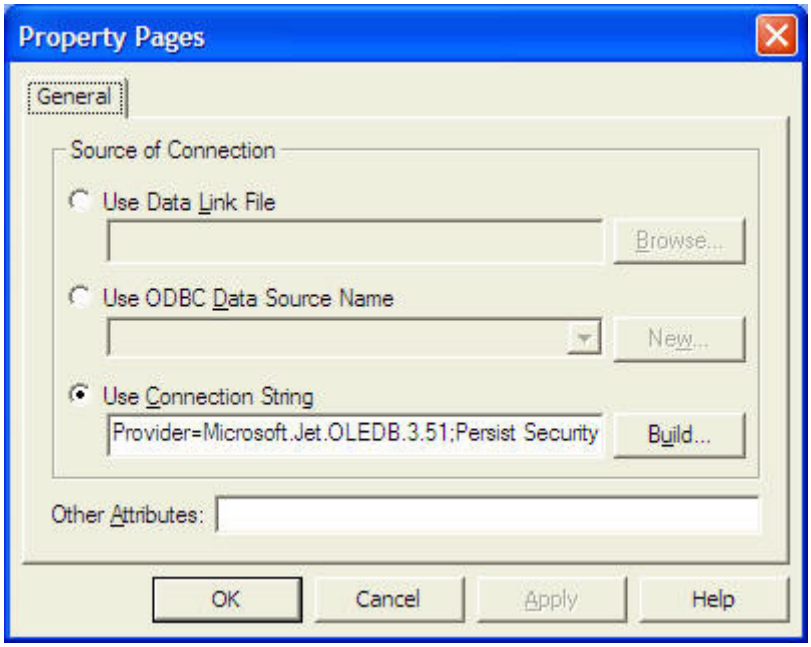
Form Name	frmBookTitle
Form Caption	Book Titles - ADOApplication
ADO Name	adoBooks
Label1 Name	lblApp
Label1 Caption	Book Titles
Label 2 Name	lblTitle
Label2 Caption	Title :
Label3 Name	lblYear
Label3 Caption	Year Published:
Label4 Name	lblISBN
Label4 Caption	ISBN:
Labe5 Name	lblPubID
Label5 Caption	Publisher's ID:
Label6 Name	lblSubject
Label6 Caption	Subject :
TextBox1 Name	txtitle
TextBox1 DataField	Title
TextBox1 DataSource	adoBooks
TextBox2 Name	txtPub
TextBox2 DataField	Year Published
TextBox2 DataSource	adoBooks

TextBox3 Name	txtISBN
TextBox3 DataField	ISBN
TextBox3 DataSource	adoBooks
TextBox4 Name	txtPubID
TextBox4 DataField	PubID
TextBox4 DataSource	adoBooks
TextBox5 Name	txtSubject
TextBox5 DataField	Subject
TextBox5 DataSource	adoBooks
Command Button1 Name	cmdSave
Command Button1 Caption	&Save
Command Button2 Name	cmdAdd
Command Button2 Caption	&Add
Command Button3 Name	cmdDelete
Command Button3 Caption	&Delete
Command Button4 Name	cmdCancel
Command Button4 Caption	&Cancel
Command Button5 Name	cmdPrev
Command Button5 Caption	&<
Command Button6 Name	cmdNext
Command Button6 Caption	&>
Command Button7 Name	cmdExit
Command Button7 Caption	E&xit

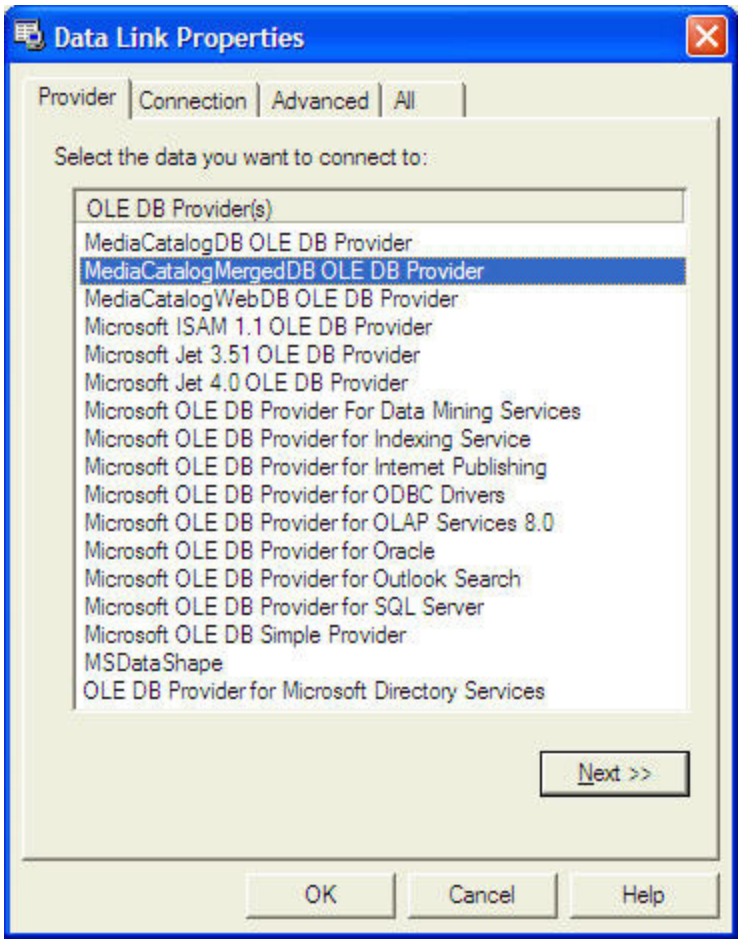
To be able to access and manage a database, you need to connect the ADO data control to a database file. We are going to use **BIBLIO.MDB** that comes with VB6. To connect ADO to this database file , follow the steps below:

a) Click on the ADO control on the form and open up the properties window.

b) Click on the ConnectionString property, the following dialog box will appear.

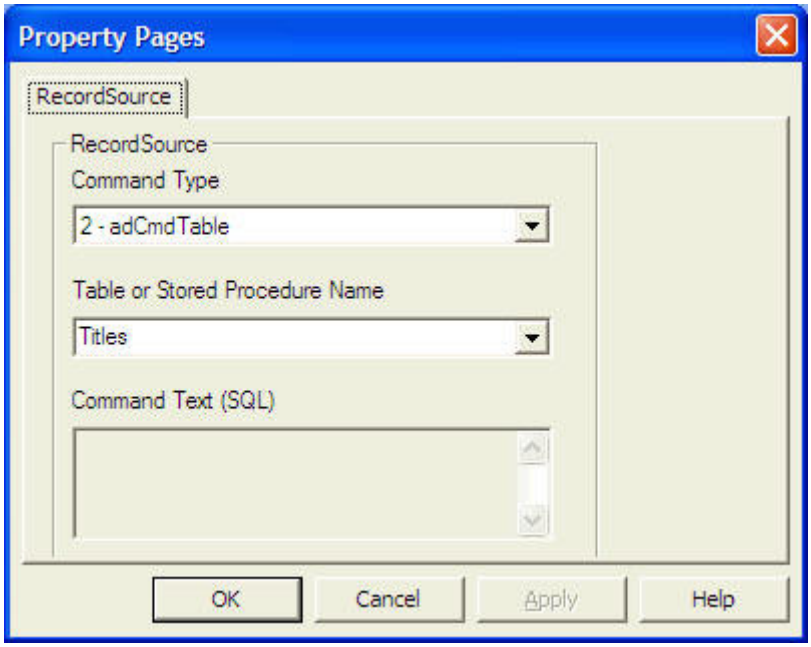


when the dialog box appear, select the Use **Connection String**'s Option. Next, click build and at the **Data Link dialog box**, double-Click the option labeled **Microsoft Jet 3.51 OLE DB provider.**



After that, click the Next button to select the file **BIBLO.MDB.** You can click on Text Connection to ensure proper connection of the database file. Click OK to finish the connection.

Finally, click on the RecordSource property and set the **command type** to **adCmd Table** and **Table name** to **Titles**. Now you are ready to use the database file.



Now, you need to write code for all the command buttons. After which, you can make the ADO control invisible.

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.BOF Then
adoBooks.Recordset.MovePrevious
If adoBooks.Recordset.BOF 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
```

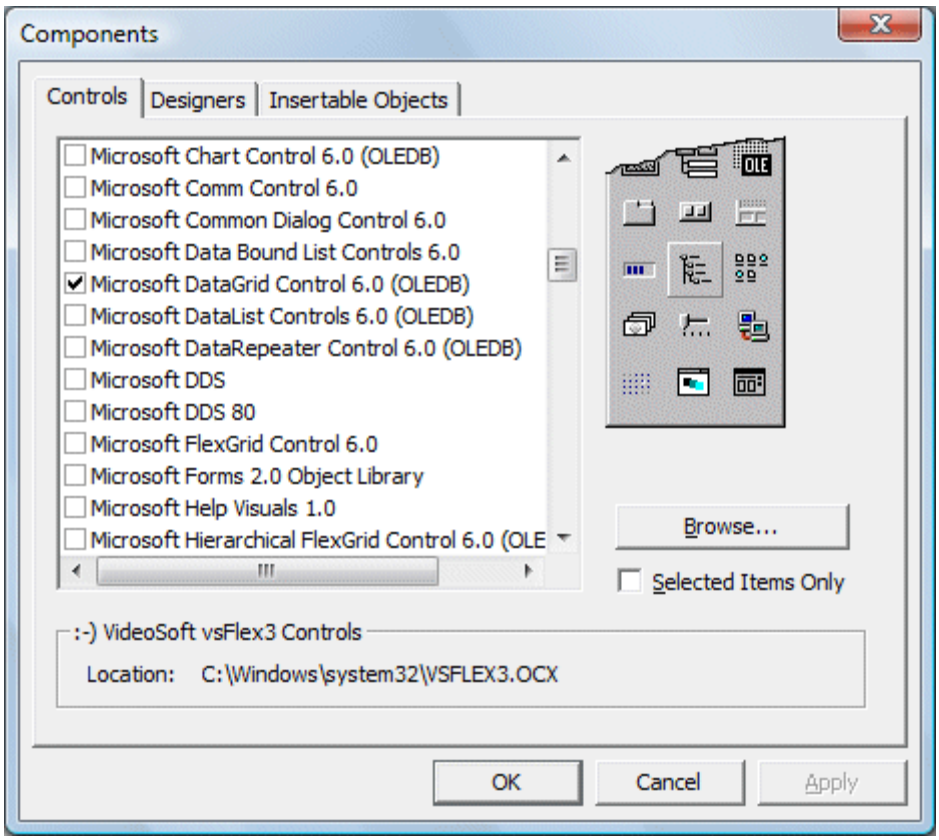
```
If adoBooks.Recordset.EOF Then
adoBooks.Recordset.MovePrevious
End If
End If

End Sub
```

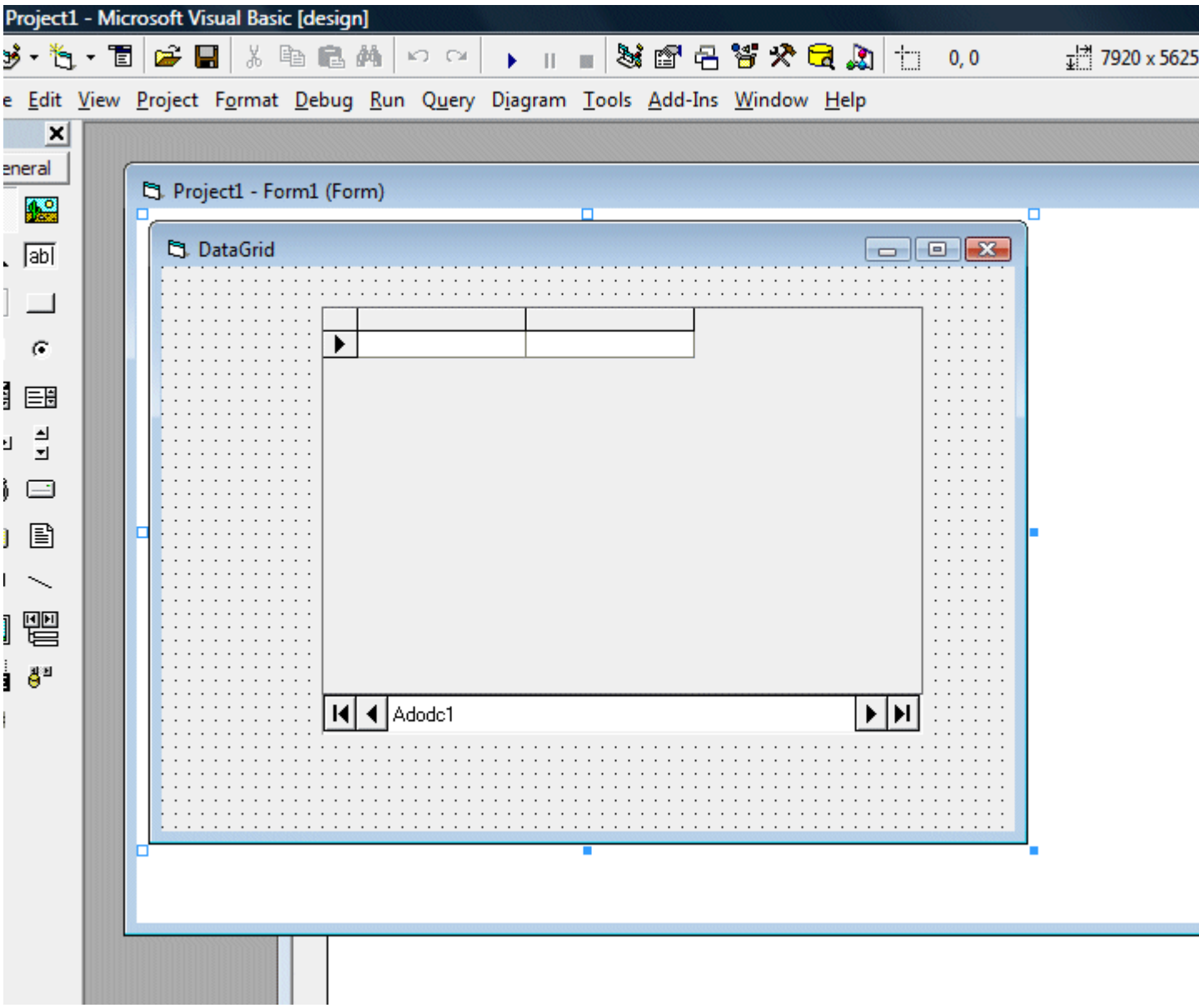
Lesson 26: Using Microsoft DataGrid Control 6.0

In the previous chapter, we use textboxes to display data by connecting them to a database via Microsoft ADO data Control 6.0. The textbox is not the only control that can display data from a database, many other controls in Visual Basic can display data. One of the them is the DataGrid control. DataGrid control can be used to display the entire table of a recordset of a database. It allows users to view and edit the data.

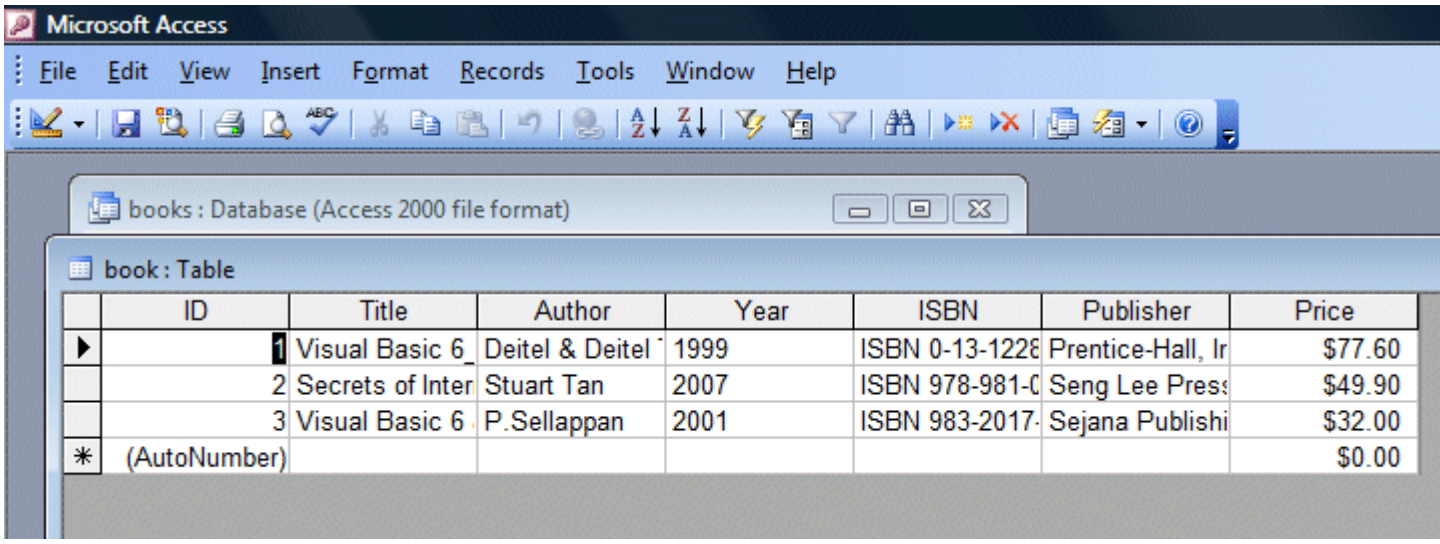
DataGrid control is the not the default item in the Visual Basic control toolbox, you have add it from the VB6 components. To add the DataGrid control, click on the project in the menu bar and select components where a dialog box that displays all the available VB6 components. Select Microsoft DataGrid Control 6.0 by clicking the checkbox beside this item. Before you exit the dialog box, you also need to select the Microsoft ADO data control so that you are able to access the database. Lastly, click on the OK button to exit the dialog box. Now you should be able to see that the DataGrid control and the ADO data control are added to the toolbox. The next step is to drag the DataGrid control and the ADO data control into the form.

The components dialog box is shown below:

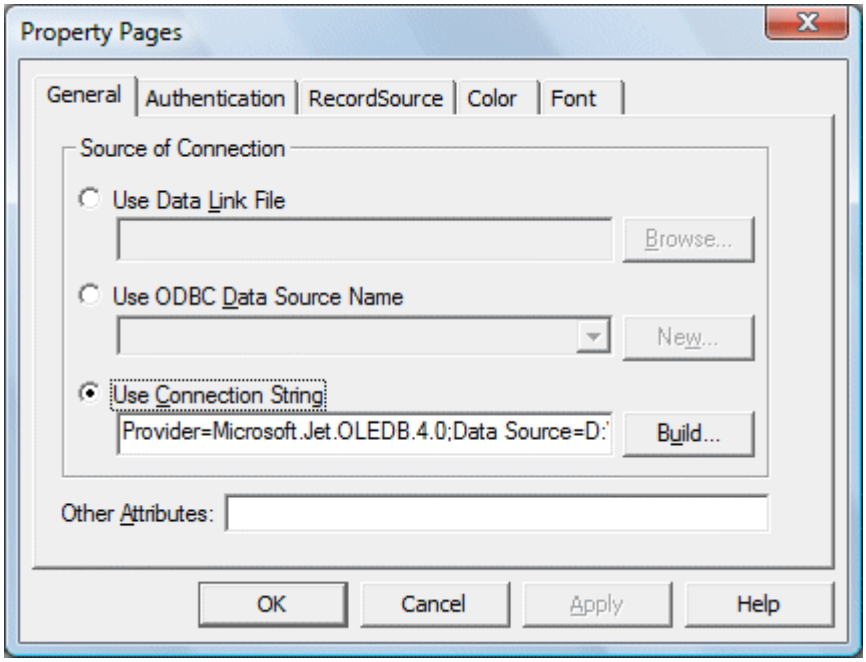

Components
Controls
Designers
Insertable Objects
Microsoft Chart Control 6.0 (OLEDB)
Microsoft Comm Control 6.0
Microsoft Common Dialog Control 6.0
Microsoft Data Bound List Controls 6.0
Microsoft DataGrid Control 6.0 (OLEDB)
Microsoft DataList Controls 6.0 (OLEDB)
Microsoft DataRepeater Control 6.0 (OLEDB)
Microsoft DDS
Microsoft DDS 80
Microsoft FlexGrid Control 6.0
Microsoft Forms 2.0 Object Library
Microsoft Help Visuals 1.0
Microsoft Hierarchical FlexGrid Control 6.0 (OLE
Browse...
Selected Items Only
:-) VideoSoft vsFlex3 Controls
Location: C:\Windows\system32\VSFLEX3.OCX
OK
Cancel
Apply

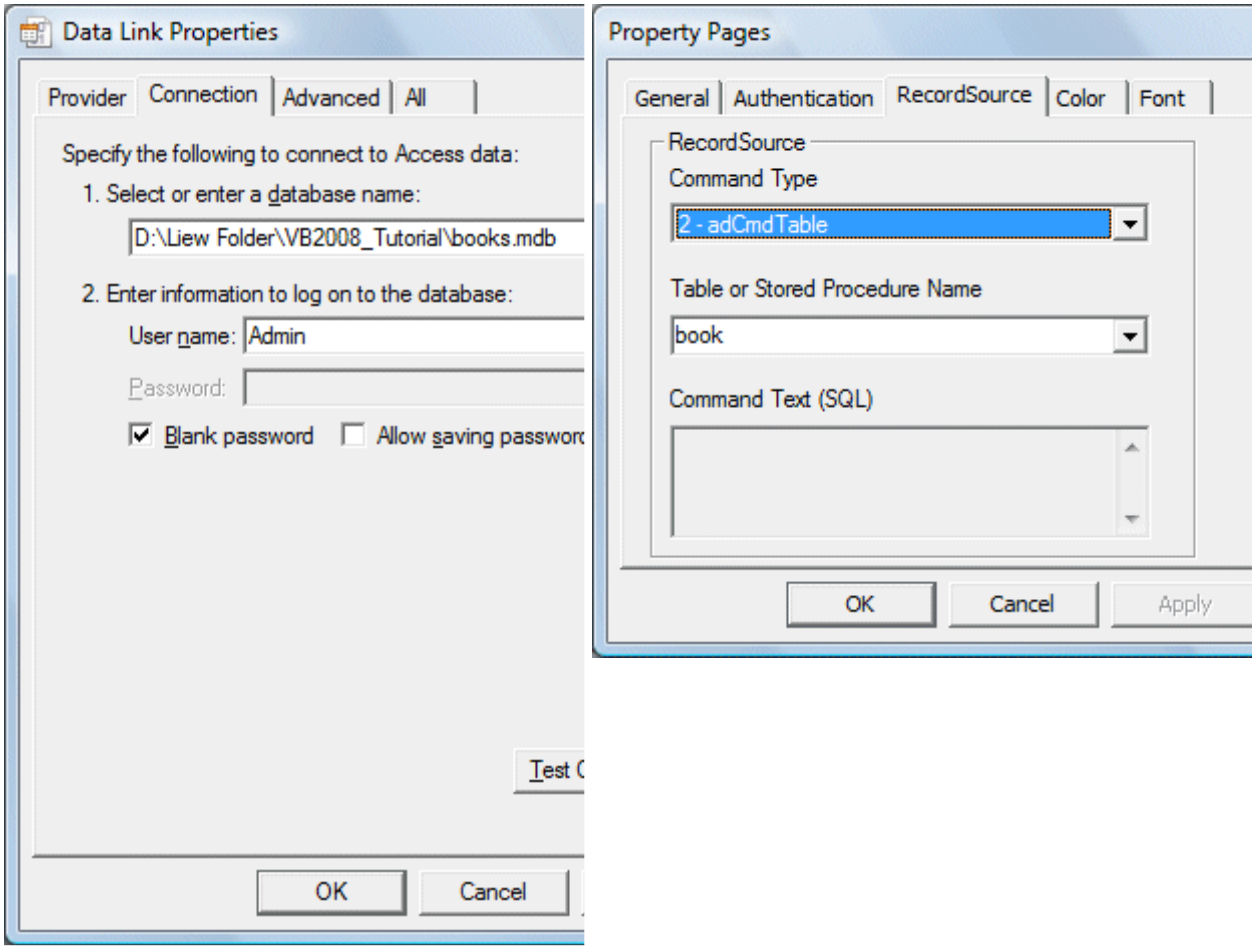


Before you proceed , you need to create a database file using Microsoft Access. Here I created a file to store my the information of my books and I name the table book. After you have created the table, enter a few records such as mine. The table is shown below:

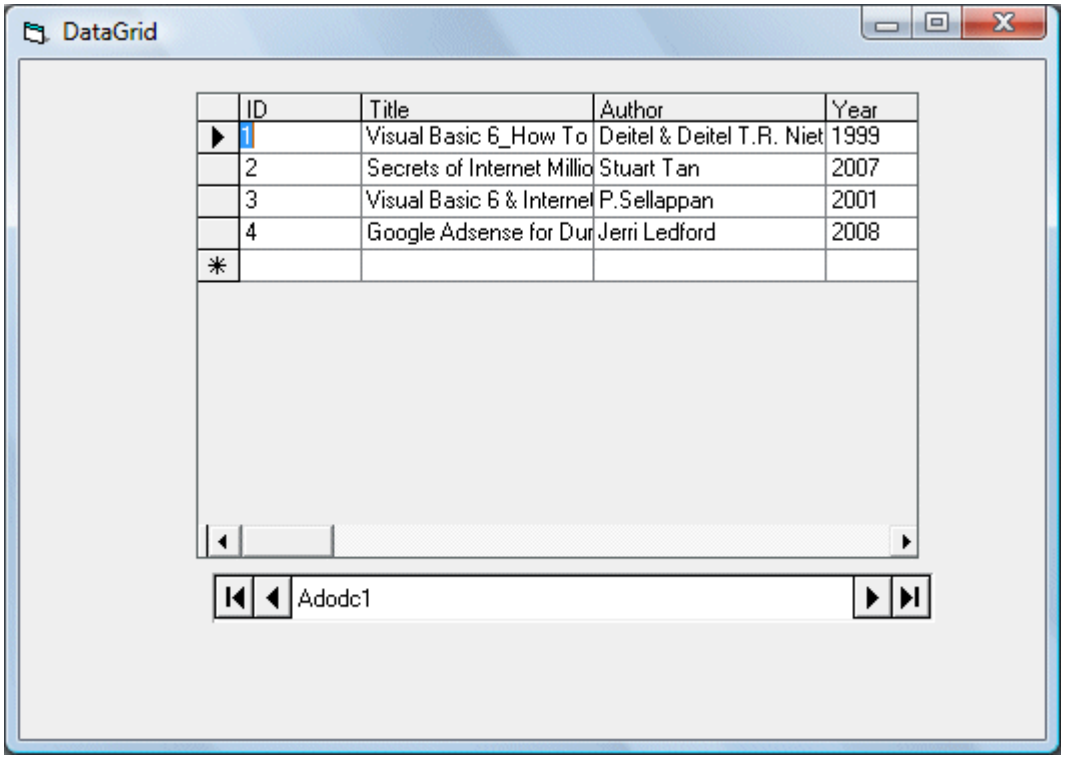


Now you need to connect the database to the ADO data control. To do that, right click on the ADO data control and select the ADODC properties, the following dialog box will appear.





Finally you need to display the data in the DataGrid control. To accomplish this, go to the properties window and set the DataSource property of the DataGrid to Adodc1. You can also permit the user to add and edit your records by setting the AllowUpdate property to True. If you set this property to false, the user cannot edit the records. Now run the program and the output window is shown below:



Lesson 27: Using SQL queries in Visual Basic 6

In the previous chapter, we have learned to use the DataGrid Control to display data from a database in Visual Basic 6 environment. However, it does not allow users to search for and select the information they want to see. In order to search for a certain information, we need to use SQL query. SQL stands for **S**tructures **Q**uery **L**anguage. Using SQL keywords, we are able to select specific information to be displayed based on certain criteria. The most basic SQL keyword is SELECT, it is used together with the keyword FROM to select information from one or more tables from a database. The syntax is:

SELECT fieldname1,fieldname2,.....,fieldnameN FROM TableName

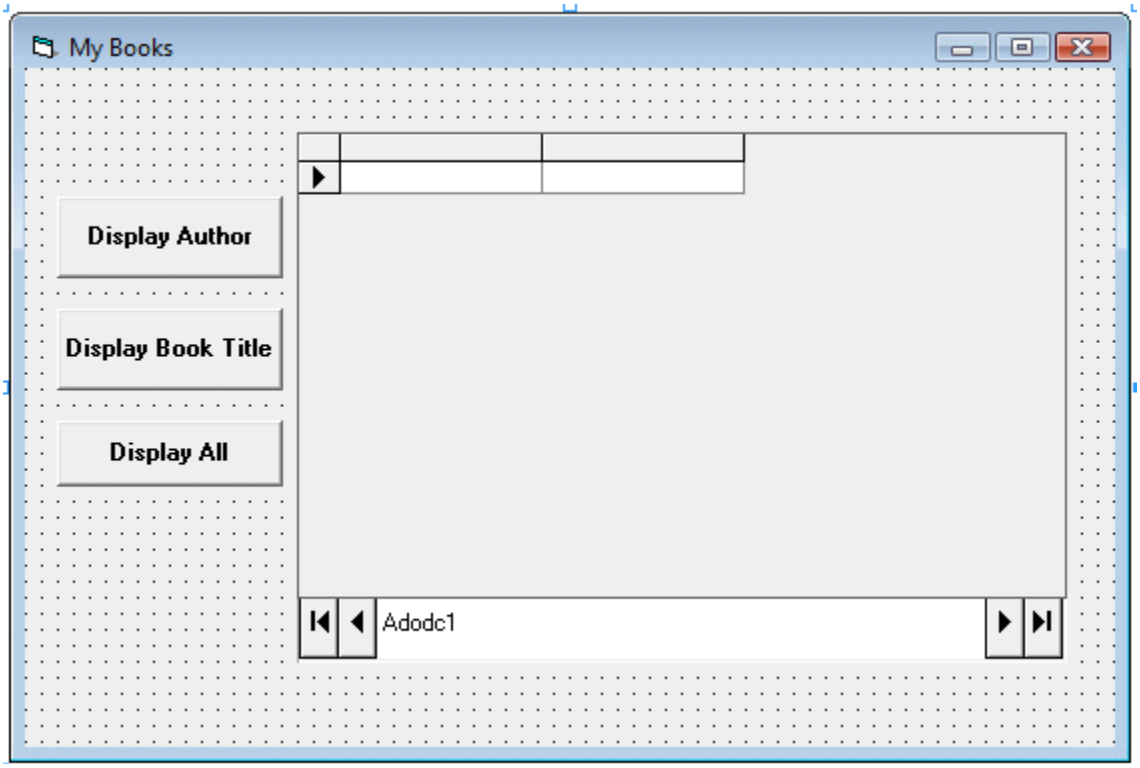
fieldname1, fieldname2,.....fieldnameN are headings of the columns from a table of a database. You can select any number of fieldname in the query. If you wish to select all the information, you can use the following syntax:

SELECT * FROM TableName

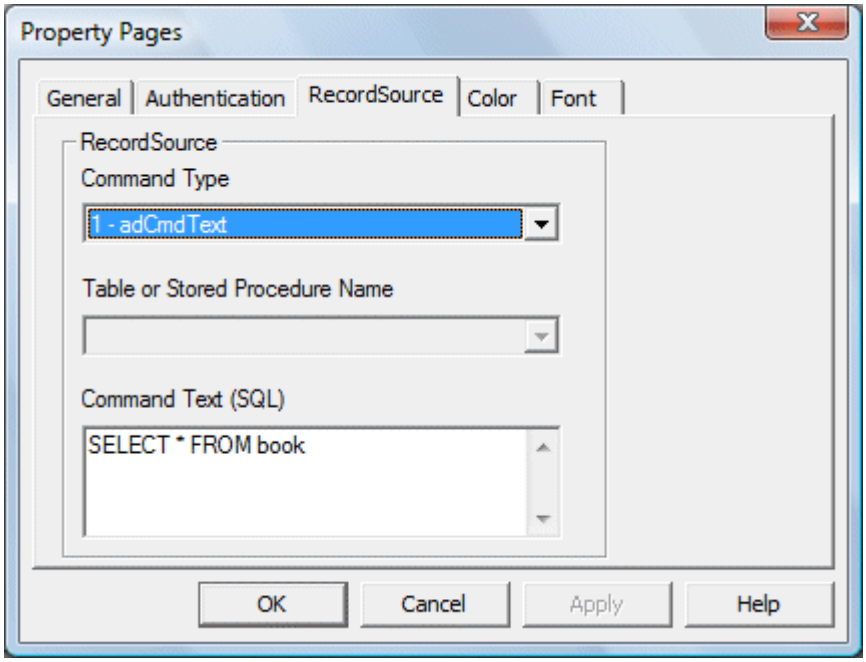
In order to illustrate the usage of SQL queries, lets create a new database in Microsoft Access with the following filenames **ID, Title, Author, Year, ISBN, Publisher, Price** and save the table as **book** and the database as **books.mdb** in a designated folder.

Next, we will start Visual Basic and insert an ADO control, a DataGrid and three command buttons. Name the three command buttons as **cmdAuthor, cmdTitle** and **cmdAll**. Change

their captions to **Display Author** ,**Display Book Title** and **Display All** respectively. You can also change the caption of the form to My Books. The design interface is shown below:



Now you need to connect the database to the ADO data control. Please refer to lesson 25 for the details. However, you need to make one change. At the ADODC property pages dialog box, click on the Recordsource tab and select **1-adCmdText** under command type and under Command Text(SQL) key in **SELECT * FROM book.**



Next, click on the command buttton cmdAuthor and key in the following statements:

```
Private Sub cmdAuthor_Click()

Adodc1.RecordSource = "SELECT Author FROM book"
Adodc1.Refresh
Adodc1.Caption = Adodc1.RecordSource

End Sub
```

and for the command button cmdTitle, key in

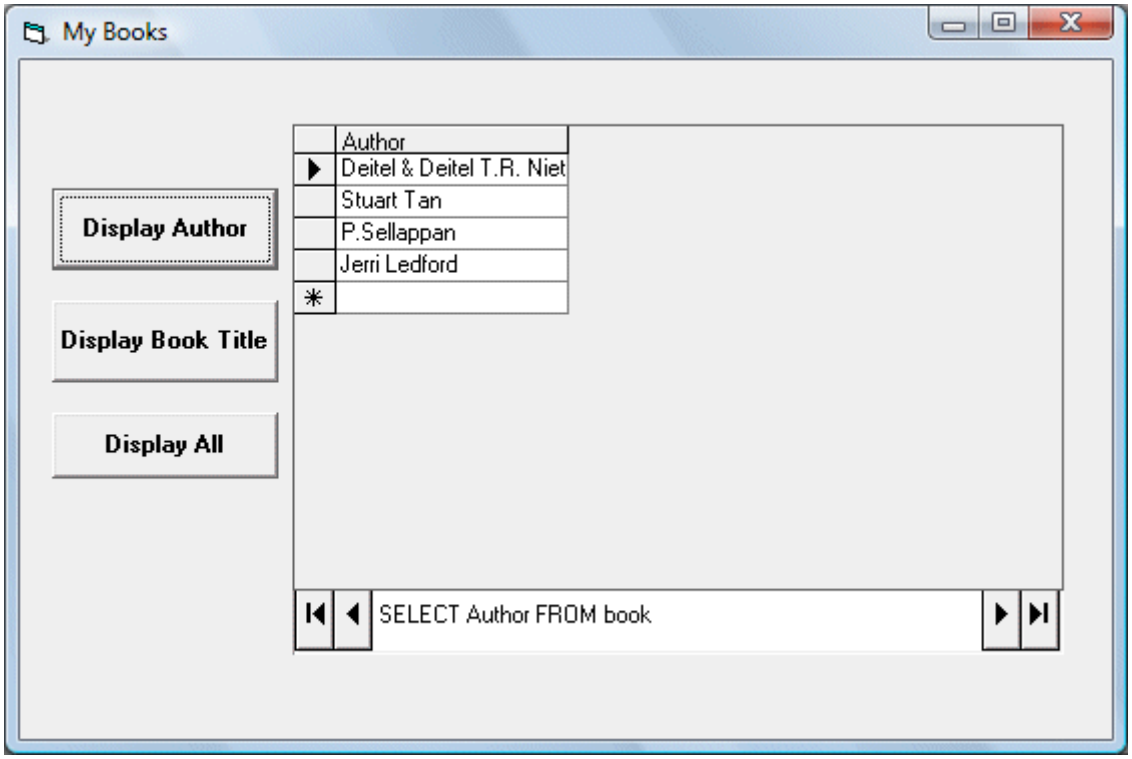
```
Private Sub cmdTitle_Click()

Adodc1.RecordSource = "SELECT Title FROM book"
Adodc1.Refresh
Adodc1.Caption = Adodc1.RecordSource

End Sub
```

Finally for the command button cmdAll, key in

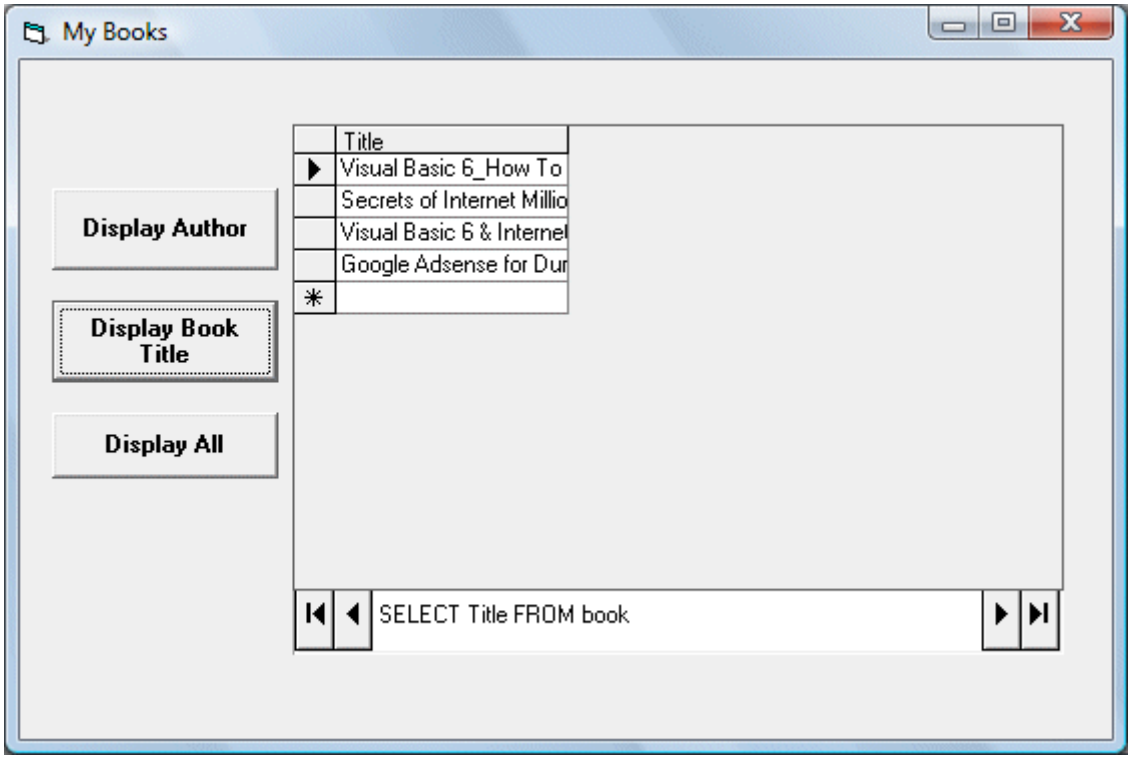
```
Private Sub cmdAll_Click()

Adodc1.RecordSource = "SELECT * FROM book"
Adodc1.Refresh
Adodc1.Caption = Adodc1.RecordSource
```

End Sub

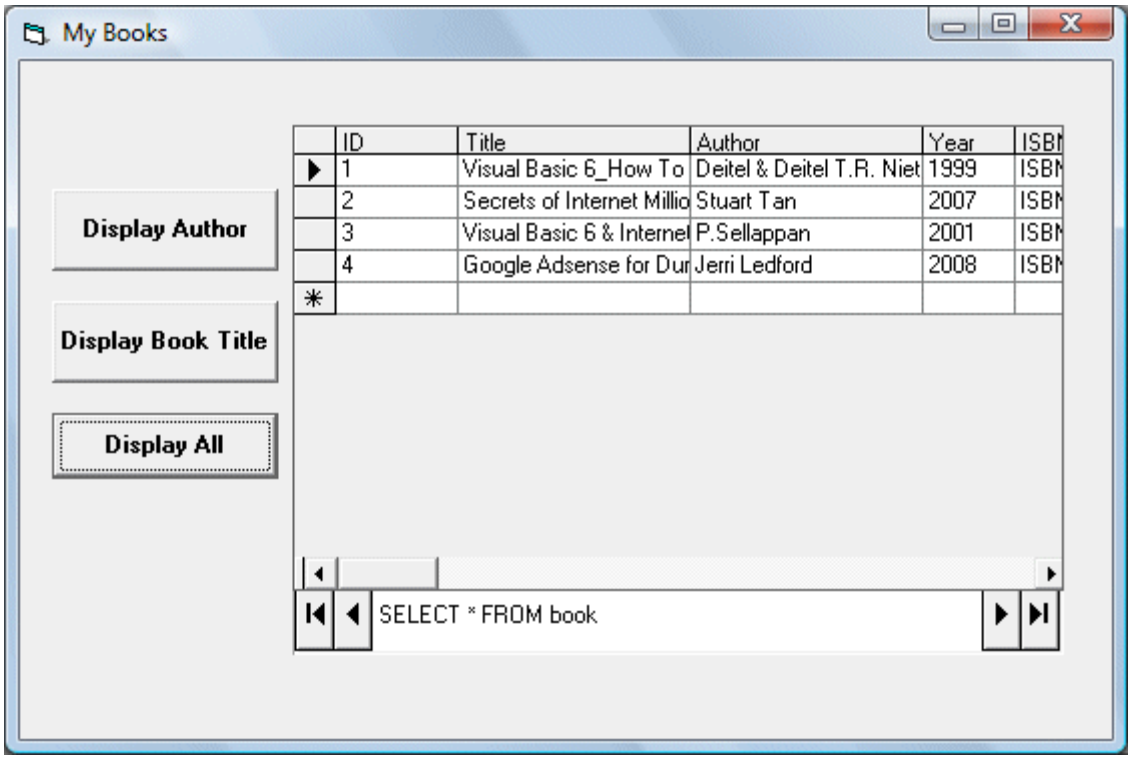
Now, run the program and when you click on the Display Author button, only the names of authors will be displayed, as shown below:



and when you click on the Display Book Title button, ony the book titles will be displayed, as show below:



Lastly, click on the Display All button and all the information will be displayed.



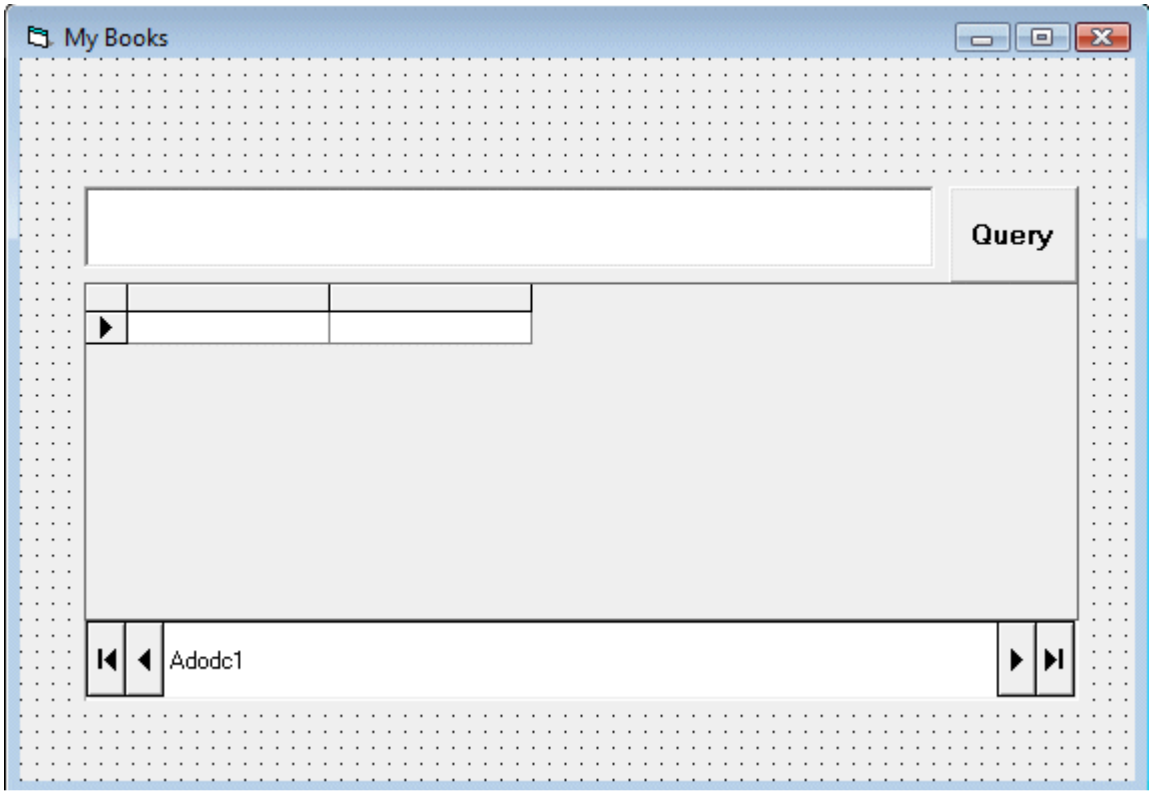
Lesson 28: More SQL Keywords

In the previous chapter, we have learned to use the basic SQL keywords SELECT and FROM to manipulate database in Visual Basic 6 environment. In this lesson, you will learn to use more SQL keywords. One of the more important SQL keywords is WHERE. This keyword allow the user to search for data that fulfill certain criteria. The Syntax is as follows:

SELECT fieldname1,fieldname2,.....,fieldnameN FROM TableName WHERE Criteria

The criteria can be specified using operators such as =, >,<, <=, >=, <> and Like.

Using the database books.mdb created in the previous chapter, we will show you a few examples. First of all, start a new project and insert a DataGrid control and an ADO control into the form. . At the ADODC property pages dialog box, click on the Recordsource tab and select **1-adCmdText** under command type and under Command Text(SQL) key in **SELECT * FROM book.** Next, insert one textbox and put it on top of the DataGrid control, this will be the place where the user can enter SQL query text. Insert one command button and change the caption to Query. The design interface is shown below:



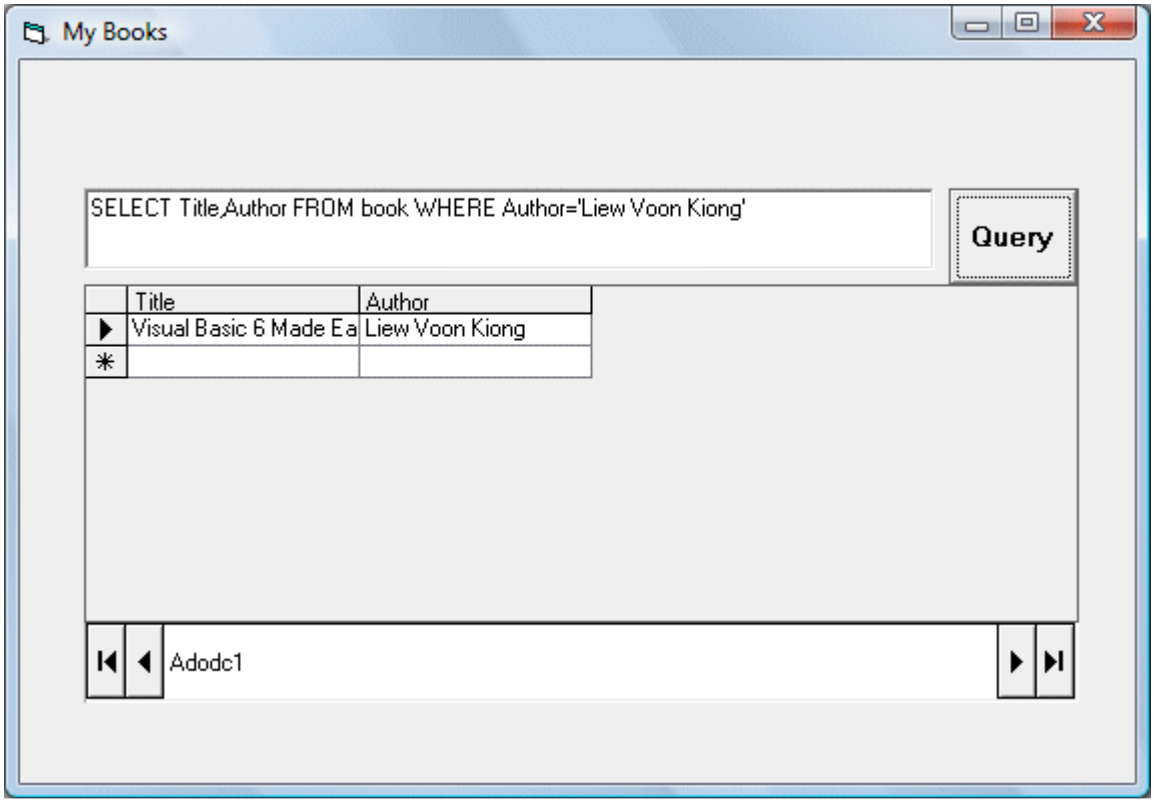
Example 21d1: Query based on Author

Run the program and key in the following SQL query statement

SELECT Title, Author FROM book WHERE Author='Liew Voon Kiong'

Where you click on the query button, the DataGrid will display the author name Liew Voon

Kiong. as shown below:

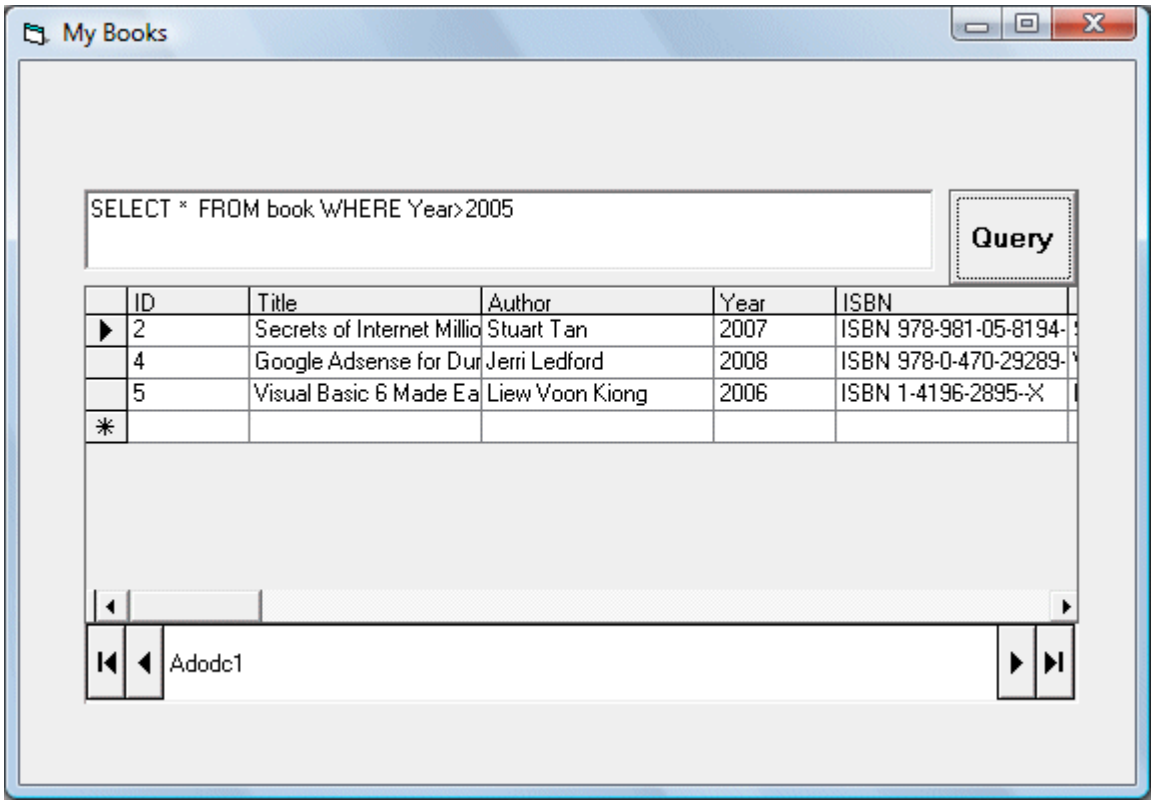


Example 21d2:Query based on year

Run the program and key in the following SQL query statement:

SELECT * FROM book WHERE Year>2005

Where you click on the query button, the DataGrid will display all the books that were published after the year 2005.



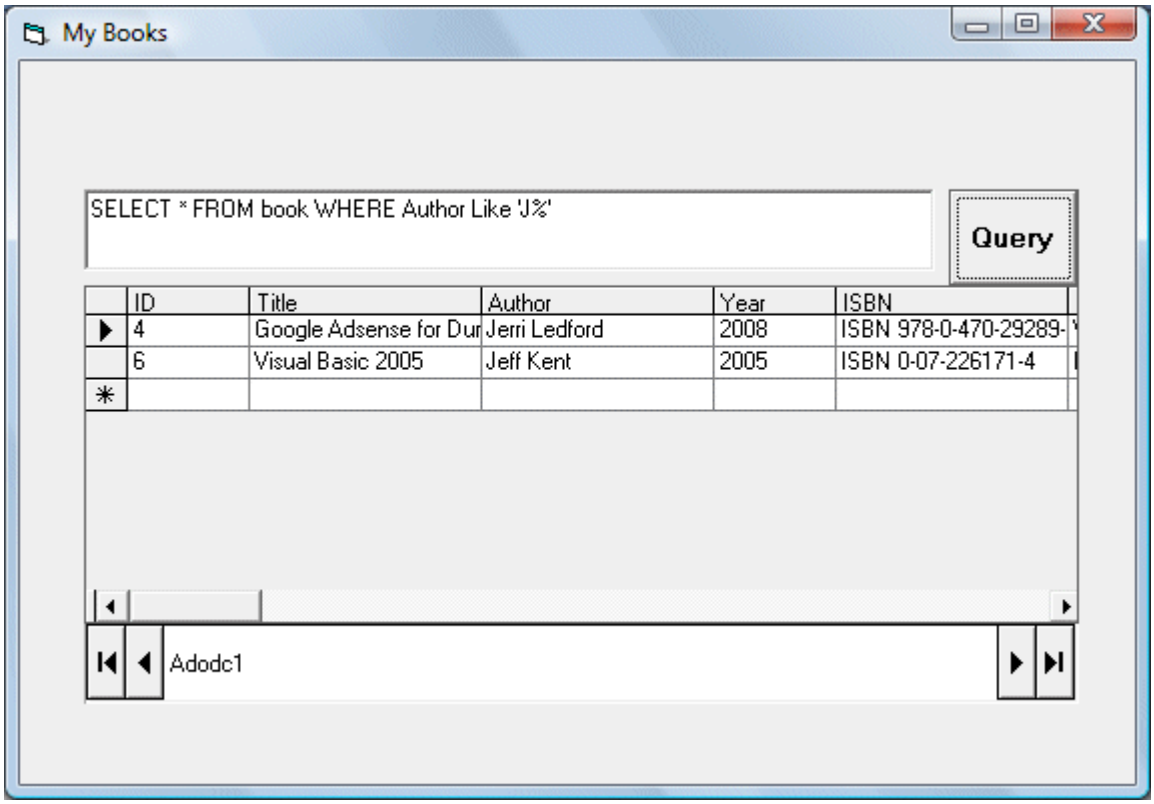
You can also try following queries:

- SELECT * FROM book WHERE Price<=80
- SELECT * FROM book WHERE Year=2008
- SELECT * FROM book WHERE Author<>'Liew Voon Kiong'

You may also search for data that contain certain characters by pattern matching. It involves using the **Like** operator and the **%** symbol. For example, if you want to search for a author name that begins with alphabet J, you can use the following query statement

SELECT * FROM book WHERE Author Like 'J%'

Where you click on the query command button, the records where authors' name start with the alphabet J will be displayed, as shown below:



Next, if you wish to rank order the data, either in ascending or descending order, you can use the **ORDER By , ASC (for ascending) and DESC(Descending)** SQL keywords.

The general formats are

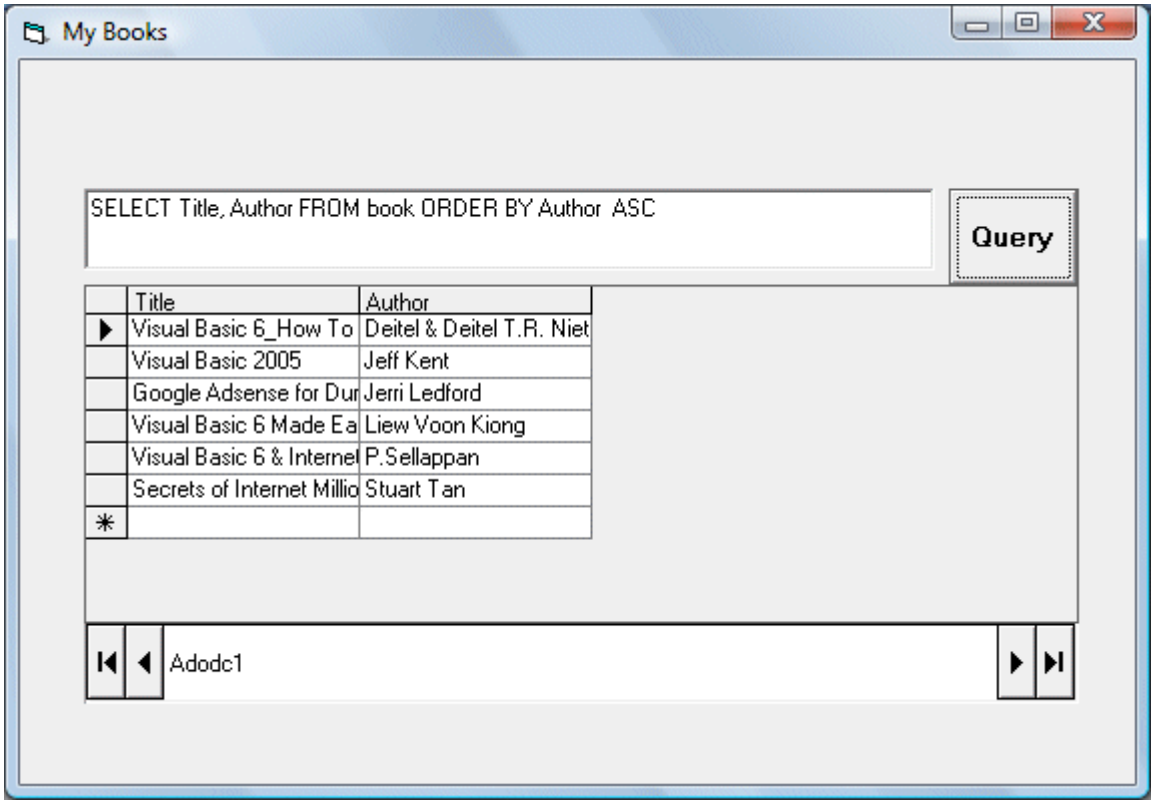
SELECT fieldname1, fieldname2…..FROM table ORDER BY fieldname ASC

SELECT fieldname1, fieldname2…..FROM table ORDER BY fieldname DESC

Example 21d3:

The following query statement will rank the records according to Author in ascending order.

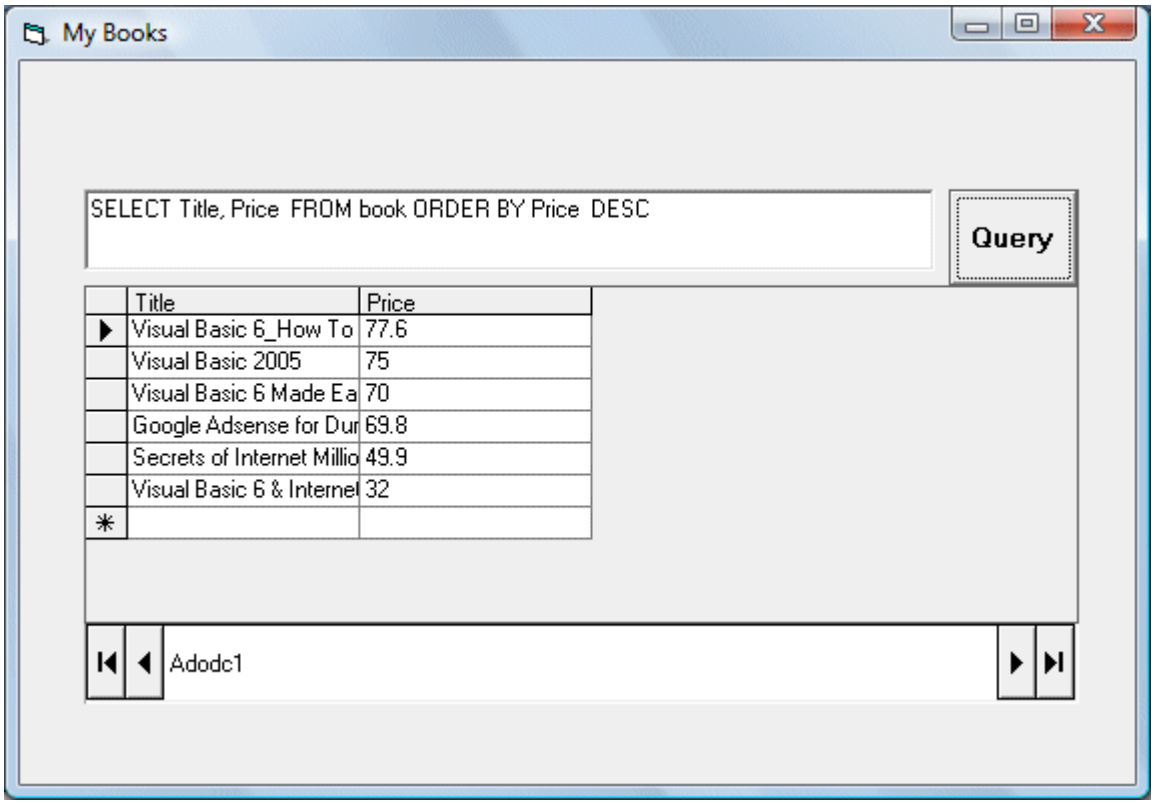
SELECT Title, Author FROM book ORDER BY Author ASC



Example 21d4

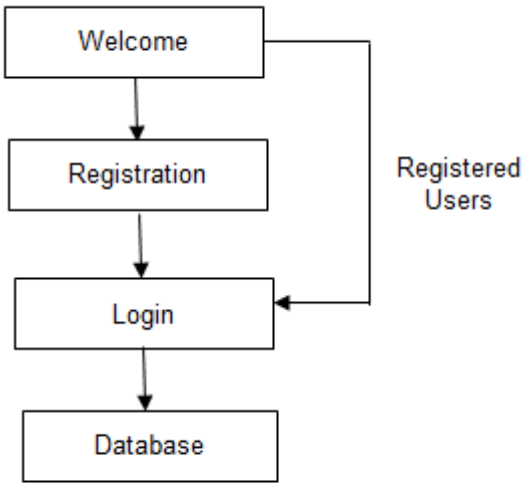
The following query statement will rank the records according to price in descending order.

SELECT Title, Price FROM book ORDER BY Price DESC



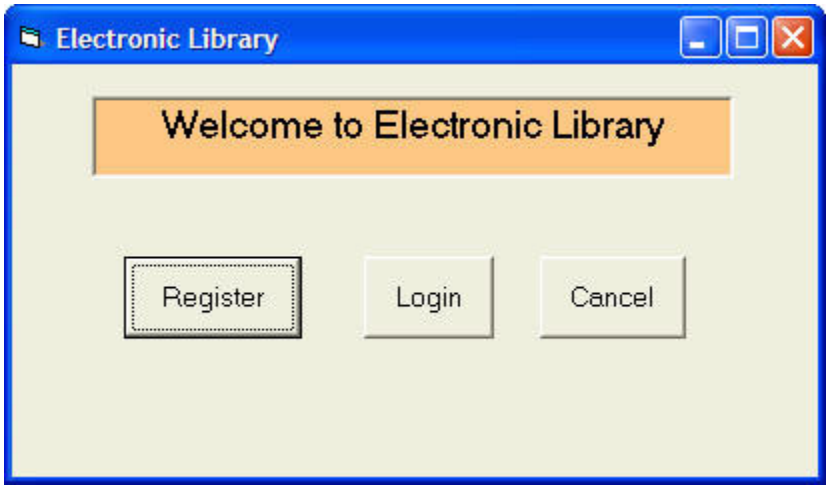
Lesson 29: Creating Advanced VB database application using ADO control

In previous lessons, you have learned how to design database applications using data control and ADO control. However, those applications are very simple and plain . In this lesson, you will learn how to create a more advanced database application using ADO control. The application you are going to create is known as an electronic library. This electronic library will be able to accept the user registration as well as handling login command that require the user's password, thus enhancing the security aspect of the database. Basically, the application will constitute a welcome menu, a registration menu, a Login menu and the main database menu. The sequence of the menus are illustrated as follow:



2.1 The Welcome Menu

First of all, you need to design the Welcome menu. You can follow the example as follow:



In this form, you need to insert three command buttons and set their properties as follow:

Form name	main_menu
command button 1 Name	cmdRegister
command button 1 Caption	Register
command button 2 Name	cmdLogin
command button 2	Login

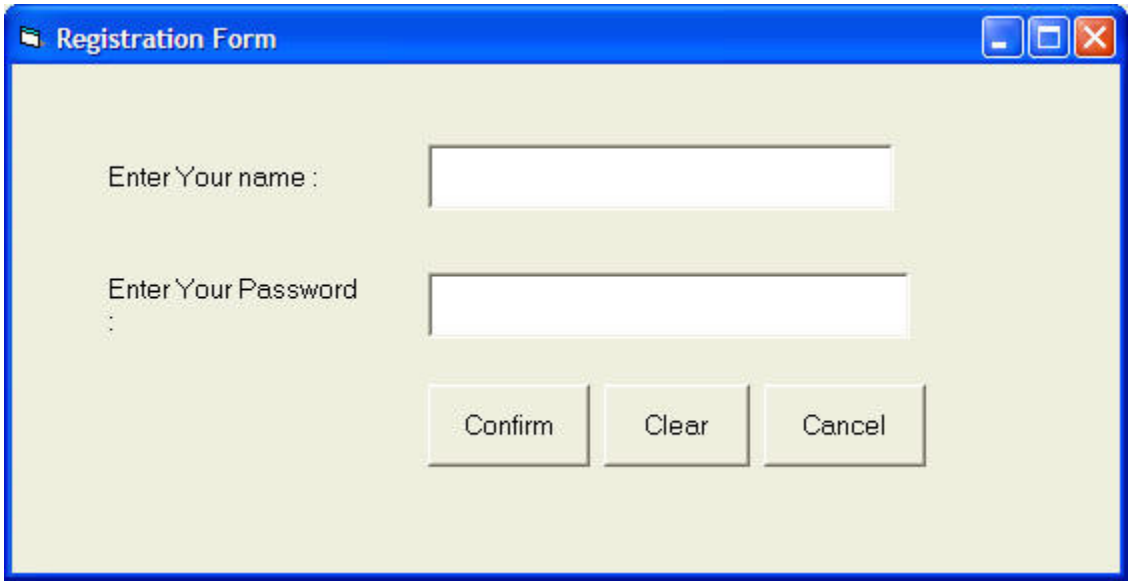
Caption	
command button 3 Name	cmdCancel
command button 3 Caption	Cancel

The code is as follows:

```
Private Sub cmdCancel_Click()
End
End Sub

Private Sub cmdLogin_Click()
main_menu.Hide
Login_form.Show
End Sub

Private Sub cmdRegister_Click()
main_menu.Hide
Register.Show
End Sub
```

29.2 The Registration Form

If a new user click the Register button, the registration form will appear. An example is illustrated as follow:



This registration forms consist of two text boxes , three command buttons and an ADO control. Their properties are set as follow:

Form name	Register
textbox 1 name	txtName
textbox 2 name	txtpassword
textbox 2 PasswordChar	*
command button 1 name	cmdConfirm
command button 1 Caption	Confirm
command button 2 name	cmdClear
command button 2 Caption	Clear
command button 3 name	cmdCancel
command button 3 Caption	Cancel
ADO control name	UserInfo

note that the PasswordChar of textbox 2 is set as * which means users will not be able to see the actual characters they enter, they will only see the * symbol.

The codes are as follow:

```
Private Sub cancel_Click( )
End
End Sub

Private Sub cmdClear_Click( )
txtName.Text = ""
txtpassword.Text = ""

End Sub

Private Sub cmdConfirm_Click()

UserInfo.Recordset.Fields("username") = txtName.Text
UserInfo.Recordset.Fields("password") = txtpassword.Text
UserInfo.Recordset.Update

Register.Hide

Login_form.Show

End Sub


Private Sub Form_Load()
UserInfo.Recordset.AddNew
End Sub
```

29.3 The Login Menu

The Login menu is illustrated as follow:

There are two text boxes and a command button, their properties are set as follow:

Textbox 1 name	txtName
Textbox 2 name	txtpassword
Command button 1 name	cmdLogin
Command button 1 Caption	Login
Form name	Login_form

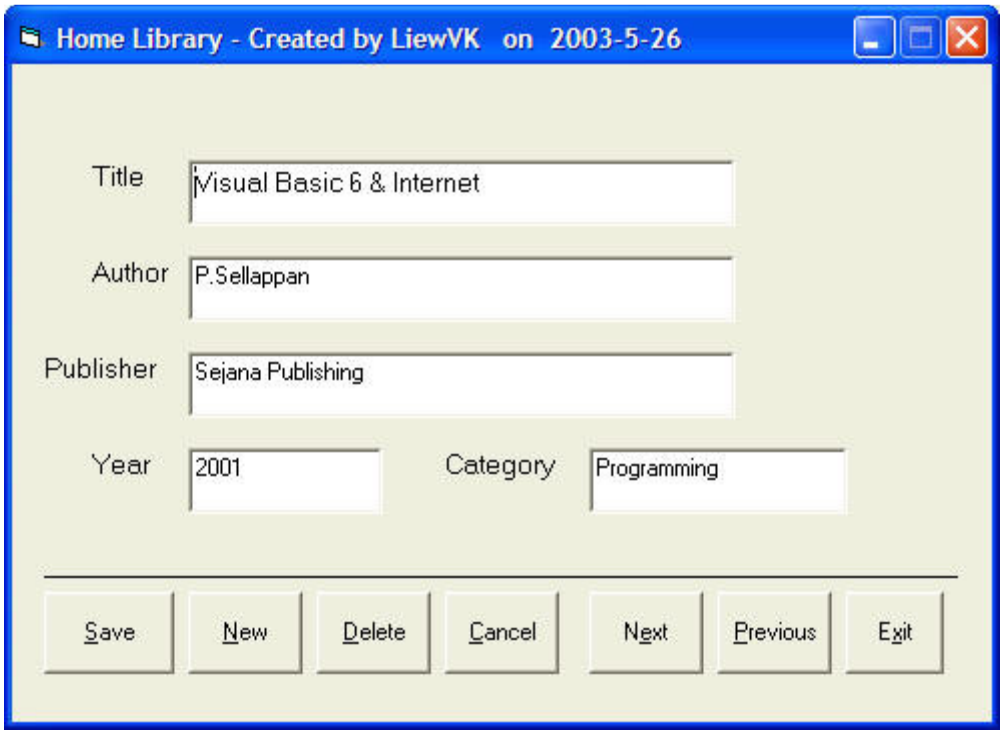
The codes are as follow:

```
Private Sub cmdLogin_Click()

Dim usrname As String
Dim psword As String
Dim usernam As String
Dim pssword As String
Dim Msg As String


Register.UserInfo.Refresh
usrname = txtName.Text
psword = txtpassword.Text


Do Until Register.UserInfo.Recordset.EOF
If Register.UserInfo.Recordset.Fields("username").Value = usrname And
```

```
Register.UserInfo.Recordset.Fields("password").Value = psword Then
Login_form.Hide
frmLibrary.Show
Exit Sub

Else
Register.UserInfo.Recordset.MoveNext
End If

Loop

Msg = MsgBox("Invalid password, try again!", vbOKCancel)
If (Msg = 1) Then
Login_form.Show
txtName.Text = ""
txtpassword = ""

Else
End
End If


End Sub
```

29.4 The Main Database Manager

The main database manager is illustrated as follow:



The properties of all controls are listed in the table below:

Form name	frmLibrary
ADO control name	adoLibrary
ADO visible	False
TextBox 1 name	txtTitleA
TextBox 2 name	txtAuthor
TextBox 3name	txtPublisher
TextBox 4 name	txtYear
TextBox 5 name	txtCategory
Command button 1 name	cmdSave
Command button 1 caption	&Save
Command button 2 name	cmdNew
Command button 2 caption	&New
Command button 3 name	cmdDelete
Command button 3 caption	&Delete
Command button 4 name	cmdCancel
Command button 4 caption	&Cancel
Command button 5 name	cmdNext
Command button 5 caption	N&ext
Command button 6 name	cmdPrevious
Command button 6 caption	&Previous
Command button 7 name	cmdExit
Command button 7 caption	E&xit

The codes are as follow:

```
Private Sub cmdCancel_Click()
txtTitle.Text = ""
txtAuthor.Text = ""
```

```
txtPublisher.Text = ""
txtYear.Text = ""
txtCategory.Text = ""
End Sub

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

End Sub

Private Sub cmdExit_Click()
End
End Sub

Private Sub cmdNew_Click()
adoLibrary.Recordset.AddNew

End Sub

Private Sub cmdNext_Click()
If Not adoLibrary.Recordset.EOF Then
adoLibrary.Recordset.MoveNext
If adoLibrary.Recordset.EOF Then
adoLibrary.Recordset.MovePrevious
End If
End If
End Sub

Private Sub cmdPrevious_Click()
If Not adoLibrary.Recordset.BOF Then
adoLibrary.Recordset.MovePrevious
If adoLibrary.Recordset.BOF Then
adoLibrary.Recordset.MoveNext
End If
End If
End Sub

Private Sub cmdSave_Click()

adoLibrary.Recordset.Fields("Title").Value = txtTitle.Text
adoLibrary.Recordset.Fields("Author").Value = txtAuthor.Text
adoLibrary.Recordset.Update

End Sub
```

Lesson 30 : Animation-Part I

Animation is always an interesting and exciting part of programming. Although visual basic is not designed to handle advance animations, you can still create some interesting animated effects if you put in some hard thinking. There are many ways to create animated effects in VB6, but for a start we will focus on some easy methods.

The simplest way to create animation is to set the VISIBLE property of a group of images or pictures or texts and labels to true or false by triggering a set of events such as clicking a button. Let's examine the following example:

This is a program that create the illusion of moving the jet plane in four directions, North, South ,East, West. In order to do this, insert five images of the same picture into the form. Set the visible property of the image in the center to be true while the rest set to false. On start-up, a user will only be able to see the image in the center. Next, insert four command buttons into the form and change the labels to Move North, Move East, Move West and Move South respectively. Double click on the move north button and key in the following procedure:

```
Sub Command1_click( )

Image1.Visible = False
Image3.Visible = True
Image2.Visible = False
Image4.Visible = False
Image5.Visible = False

End Sub
```

By clicking on the move north button, only image 3 is displayed. This will give an illusion that the jet plane has moved north. Key in similar procedures by double clicking other command buttons. You can also insert an addition command button and label it as Reset and key in the following codes:

```
Image1.Visible = True
Image3.Visible = False
Image2.Visible = False
Image4.Visible = False
Image5.Visible = False
```

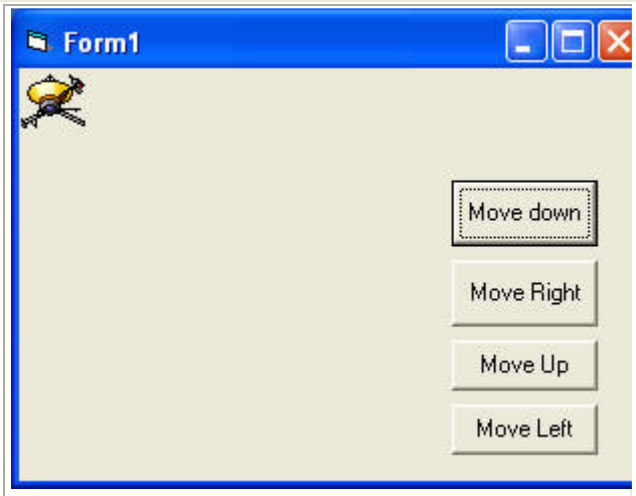
Clicking on the reset button will make the image in the center visible again while other images become invisible, this will give the false impression that the jet plane has move back to the original position.

You can also issue the commands using a textbox, this idea actually came from my son Liew Xun (10 years old). His program is shown below:

```
Private Sub Command1_Click()

If Text1.Text = "n" Then
Image1.Visible = False
Image3.Visible = True
Image2.Visible = False
Image4.Visible = False
Image5.Visible = False

ElseIf Text1.Text = "e" Then
Image1.Visible = False
Image4.Visible = True
Image2.Visible = False
```

Another simple way to simulate animation in VB6 is by using the Left and Top properties of an object. Image.Left give the distance of the image in twips from the left border of the screen, and Image.Top give the distance of the image in twips from the top border of the screen, where 1 twip is equivalent to 1/1440 inch. Using a statement such as Image.Left-100 will move the image 100 twips to the left, Image.Left+100 will move the image 100 twip away from the left(or 100 twips to the right), Image.Top-100 will move the image 100 twips to the top and Image.Top+100 will move the image 100 twips away from the top border (or 100 twips down).Below is a program that can move an object up, down. left, and right every time you click on a relevant command button.

```
Image3.Visible = False
Image5.Visible = False


ElseIf Text1.Text = "w" Then
Image1.Visible = False
Image3.Visible = False
Image2.Visible = False
Image4.Visible = False
Image5.Visible = True


ElseIf Text1.Text = "s" Then
Image1.Visible = False
Image3.Visible = False
Image2.Visible = True
Image4.Visible = False
Image5.Visible = False
End If


End Sub
```



The Code

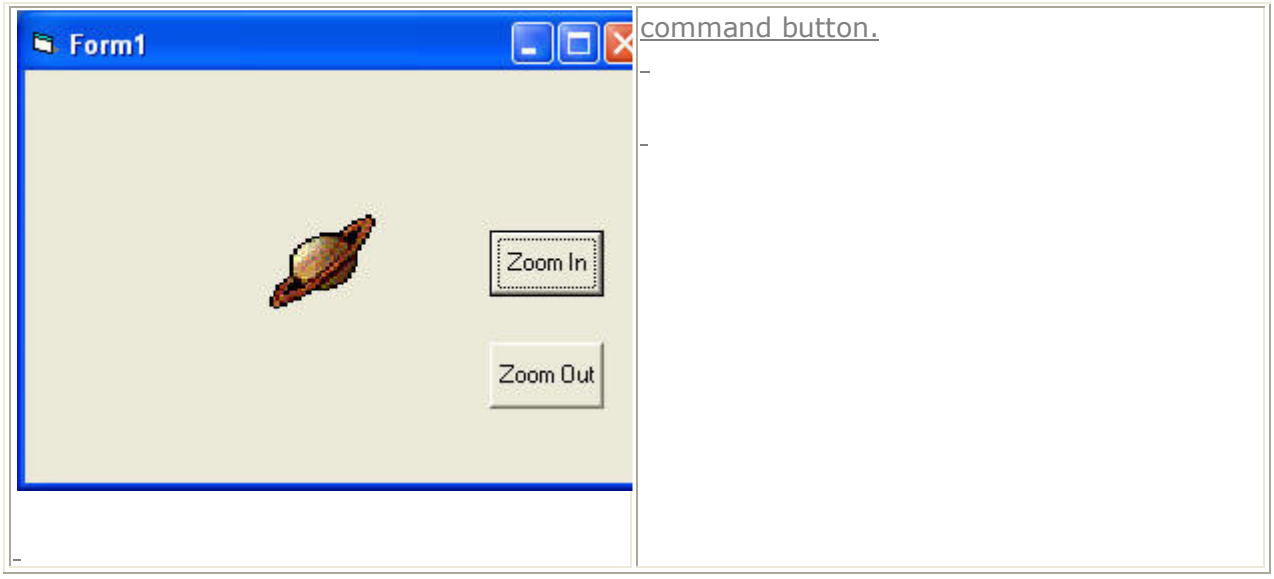
```
Private Sub Command1_Click()
Image1.Top = Image1.Top + 100
End Sub

Private Sub Command2_Click()
Image1.Top = Image1.Top - 100
End Sub

Private Sub Command3_Click()
Image1.Left = Image1.Left + 100
End Sub

Private Sub Command4_Click()
Image1.Left = Image1.Left - 100
End Sub
```

The fourth example let user magnify and diminish an object by changing the height and width properties of an object. It is quite similar to the previous example. The statements Image1.Height = Image1.Height + 100 and Image1.Width = Image1.Width + 100 will increase the height and the width of an object by 100 twips each time a user click on the relevant command button. On the other hand, The statements Image1.Height = Image1.Height - 100 and Image1.Width = Image1.Width -100 will decrease the height and the width of an object by 100 twips each time a user click on the relevant command button

The Code

```
Private Sub Command1_Click()
Image1.Height = Image1.Height + 100
Image1.Width = Image1.Width + 100
End Sub

Private Sub Command2_Click()

Image1.Height = Image1.Height - 100
Image1.Width = Image1.Width - 100

End Sub
```

You can try to combine both programs above and make an object move and increases or decreases in size each time a user click a



Lesson 31: Animation - Part II

31.1 Animation using a DragDrop Procedure

Drag and drop is a common windows application where you can drag and drop an object such as a file into a folder or into a recycle bin. This capability can be easily programmed in visual basic. In the following example, I am creating a simulation of dragging the objects into a recycle bin, then drop a fire and burn them away.

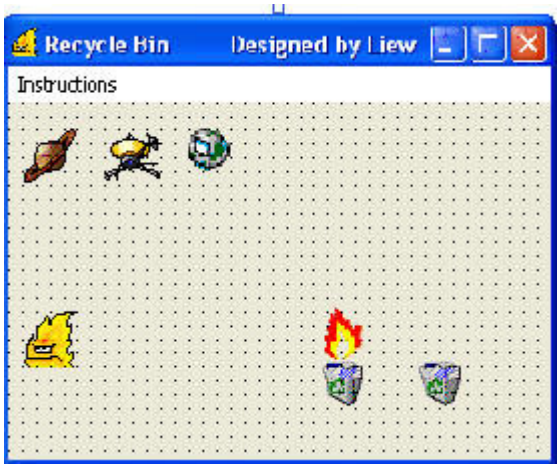
In this program, I put 6 images on the form, one of them is a recycle bin, another is a burning recycle bin , one more is the fire, and three more images. In addition, set the property **dragmode** of all the images(including the fire) that are to be dragged to **1**(**Automatic**) so that dragging is enabled, and set the visible property of burning recycle bin to false at start-up. Besides, label the tag of fire as fire in its properties windows. If you want to have better dragging effects, you need to load an appropriate icon under the dragIcon properties for those images to be dragged, preferably the icon should be the same as the image so that when you drag the image, it is like you are dragging the image along.

The essential event procedure in this program is as follows:

```
Private Sub Image4_DragDrop(Source As Control, X As Single, Y As Single)

Source.Visible = False
If Source.Tag = "Fire" Then
Image4.Picture = Image5.Picture
End If

End Sub
```

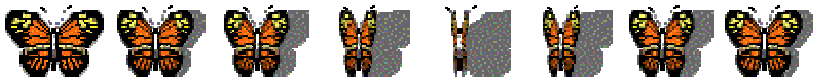
Source refer to the image to be dragged. Using the code **Source.Visible=False** means it will disappear after being dragged into the recycle bin(Image4).If the source is Fire, then the recycle bin will changed into a burning recycle bin , which is accomplished by using the code Image4.Picture = Image5.Picture, where Image 5 is the burning recycle bin.

For details of this program, please refer to my game and fun programming page or click this link, Recycle Bin.



31.2 Animation for a complete motion

So far those examples of animation shown in lesson 23 only involve movement of static images. In this lesson, you will be able to create true animation where an action finish in a complete cycle, for example, a butterfly flapping its wings. In the following example, I used eight picture frames of a butterfly which display a butterfly flapping its wing at different stages.



You can actually copy the above images and use them in your program. You need to put all the above images overlapping one another, make image1 visible while all other images invisible at start-up. Next, insert a command button and label it as Animate. Click on the command button and key in the statements that make the images appear and disappear successively by using the properties image.visible=true and image.visible=false. I use If..... Then and Elseif to control the program flow. When you run the program, you should be able to get the following animation.

.

The Interface

The Code

```
Private Sub Command1_Click()

If Image1.Visible = True Then
Image1.Visible = False
 Image2.Visible = True
ElseIf Image2.Visible = True Then
Image2.Visible = False
Image3.Visible = True
ElseIf Image3.Visible = True Then
Image3.Visible = False
Image4.Visible = True
ElseIf Image4.Visible = True Then
Image4.Visible = False
Image5.Visible = True
ElseIf Image5.Visible = True Then
Image5.Visible = False
Image6.Visible = True
ElseIf Image6.Visible = True Then
Image6.Visible = False
Image7.Visible = True
ElseIf Image7.Visible = True Then
Image7.Visible = False
Image8.Visible = True
ElseIf Image8.Visible = True Then
Image8.Visible = False
Image1.Visible = True
End If

End Sub
```

If you wish to create the effect of the butterfly flapping its wing and flying at the same time, then you could use the Left and Top properties of an object, such as the one used in the examples of lesson 23. Below is an example of a subroutine where the butterfly will flap its wing and move up at the same time. You can also write subroutines that move the butterfly to the left, to the right and to the bottom.

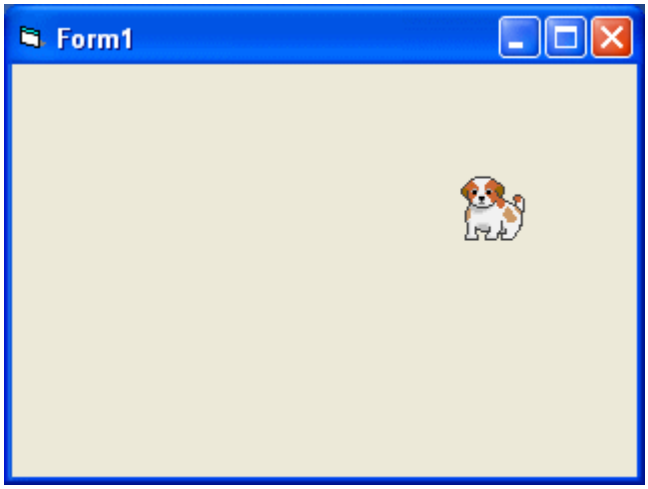
```
Sub move_up( )

If Image1.Visible = True Then
Image1.Visible = False
Image2.Visible = True
Image2.Top = Image2.Top - 100

ElseIf Image2.Visible = True Then
Image2.Visible = False
Image3.Visible = True
Image3.Top = Image3.Top - 100

ElseIf Image3.Visible = True Then
Image3.Visible = False
Image4.Visible = True
Image4.Top = Image4.Top - 100
ElseIf Image4.Visible = True Then
Image4.Visible = False
Image5.Visible = True
Image5.Top = Image5.Top - 100
ElseIf Image5.Visible = True Then
Image5.Visible = False
Image6.Visible = True
Image6.Top = Image6.Top - 100

ElseIf Image6.Visible = True Then
Image6.Visible = False
Image7.Visible = True
Image7.Top = Image7.Top - 100

ElseIf Image7.Visible = True Then
Image7.Visible = False
Image8.Visible = True
Image8.Top = Image8.Top - 100
ElseIf Image8.Visible = True Then
Image8.Visible = False
Image1.Visible = True
Image1.Top = Image1.Top - 100
End If

End Sub
```

Lesson 32: Animation - Part III

32.1 Animation using Timer

All preceding examples of animation that you have learn in lesson 23 and lesson 24 only involve manual animation, which means you need to keep on clicking a certain command button or pressing a key to make an object animate. In order to make it move automatically, you need to use a timer. The first step in creating automatic animation is to drag the timer from the toolbox into the form and set its interval to a certain value other than 0. A value of 1 is 1 milliseconds which means a value of 1000 represents 1 second. The value of the timer interval will determine the speed on an animation.

In the following example, I use a very simple technique to show animation by using the properties Visible=False and Visible=true to show and hide two images alternately. When you click on the program, you should see the following animation.



The Code

```
Private Sub
Timer1_Timer()

If Image1.Visible =
True Then
Image1.Visible = False
Image2.Visible = True
ElseIf Image2.Visible =
True Then
Image2.Visible = False
Image1.Visible = True
End If

End Sub
```

Next example shows a complete cycle of a motion such as the butterfly flapping its wing. Previous examples show only manual animation while this example will display an automatic animation once you start the program or by clicking a command button. Similar to the example under lesson 24.2, you need to insert a group of eight images of a butterfly flapping its wings at different stages. Next, insert a timer into the form and set the interval to 10 or any value you like. Remember to make image1 visible while other images invisible at start-up. Finally, insert a command button, rename its caption as Animate and key in the following statements by double clicking on this button. Bear in mind that you should enter the statements for hiding and showing the images under the timer1_timer subroutine otherwise the animation would work. Clicking on the animate button make timer start ticking and the event will run after every interval of 10 milliseconds or whatever interval you have set at design time. In future lesson, I will show you how to adjust the interval at runtime by using a slider bar or a scroll bar. When you run the program, you should see the following animation:


```
Private Sub
Form_Load()
Image1.Visible = True
x = 0
End Sub

Private Sub
Command1_Click()
Timer1.Enabled = True
End Sub

Private Sub
Timer1_Timer()
If Image1.Visible =
True Then
Image1.Visible = False
Image2.Visible = True

ElseIf Image2.Visible
= True Then
Image2.Visible = False
Image3.Visible = True
ElseIf
Image3.Visible =
True Then
Image3.Visible =
False
Image4.Visible =
True
ElseIf
Image4.Visible =
True Then
Image4.Visible =
False
Image5.Visible =
True
ElseIf
Image5.Visible =
True Then
Image5.Visible =
False
Image6.Visible =
True
ElseIf
Image6.Visible =
True Then
Image6.Visible =
False
Image7.Visible =
True
ElseIf
Image7.Visible =
True Then
Image7.Visible =
False
Image8.Visible =
True
ElseIf
Image8.Visible =
True Then
Image8.Visible =
False
Image1.Visible =
True
End If
End Sub
```

Please refer to my fun and games page for more advanced usage of the above animation , especially the

Lesson 33: Internet and Web Applications Part1-The web Browser

In order to create the web browser, you have to press Ctrl+T to open up the components window and select **Microsoft Internet Control.** After you have selected the control, you will see the control appear in the toolbox as a small globe. To insert the Microsoft Internet Control into the form, just drag the globe into the form and a white rectangle will appears in the form. You can resize this control as you wish. This control is given the default name **WebBrowser1**.

To design the interface, you need to insert one combo box which will be used to display the URLs. In addition, you need to insert a few images which will function as command buttons for the user to navigate the Internet; they are the **Go** command, the **Back** command, the **Forward** command, the **Refresh** command and the **Home** command. You can actually put in the command buttons instead of the images, but using images will definitely improve the look of the browser.

The procedures for all the commands are relatively easy to write. There are many methods, events, and properties associated with the web browser but you need to know just a few of them to come up with a functional Internet browser

The method navigate is to go the website specified by its Uniform Resource Locator(URL). The syntax is WebBrowser1.Navigate ("URL"). In this program, I want to load the www.vbtutor.net web page at start-up, so I type in its URL.

```
Private Sub Form_Load()

WebBrowser1.Navigate ("http://www.vbtutor.net")

End Sub
```

In order to show the URL in the combo box and also to display the page title at the form caption after the page is completely downloaded, I use the following statements:

```
    Private Sub

     WebBrowser1_DocumentComplete (ByVal pDisp As Object, URL As Variant)
```

```
Combo1.Text = URL
Form1.Caption = WebBrowser1.LocationName
Combo1.AddItem URL

End Sub
```

The following procedure will tell the user to wait while the page is loading.

```
Private Sub

WebBrowser1_DownloadBegin ()
Combo1.Text = "Page loading, please wait"

End Sub
```

Lesson 34: Internet and Web Applications Part 2-The FTP Program

FTP stands for **F**ile **T**ransfer **P**rotocol .The File Transfer Protocol is a system for transferring files between two computers connected by the Internet .One of the computers is known as the server and the other one is the client. The FTP program is very useful for website management. The webmaster can update the web pages by uploading the local files to the web server easily , at a much faster speed than the web browser. For normal PC users, the FTP program can also be used to download files from many FTP sites that offer a lot of useful stuffs such as free software, free games, product information, applications, tools, utilities, drivers, fixes and many more things.

The FTP program usually comprises an interface that shows the directories of the local computer and the remote server. Files can be transferred just by clicking the relevant arrows. To log into the FTP site, we have to key in the user name and the password; however, for public domains, we just need to type the word anonymous as the user name and you can leave out the password. The FTP host name takes the form ftp.servername.com, for example, the Microsoft FTP site's host name is ftp.microsoft.com .If you need to use a FTP program, you can purchase one or you can download a couple of the programs that are available free of charge from the Internet. However, you can also create your very own FTP program with Visual Basic. Visual Basic allows you to build a fully functionally FTP program which may be just as good as the commercial FTP programs. The engine behind it is the **Microsoft Internet Transfer Control 6.0** in which you need to insert it into your form before you can create the FTP program. The name of the Microsoft Internet Transfer Control 6.0.is **Inet** and if you only put in one control, its name will be **Inet1.**

Inet1 comprises three important properties namely **Inet1.URL** that is used to identify the FTP hostname, **inet1.UserName** that is used to accept the username and the **Inet1.Password** that is used to accept the user's passwords. The statements for the program to read the hostname of the server, the username and the password entered into Textbox1, Textbox2 and Textbox3 by the user are shown below:

```
Inet1.URL=Text1.Text
Inet1.UserName=Text2.Text
Inet1.Passoword=Text3.Text
```

After the user entered the above information, the program will attempt to connect to the server using the following commands, where **Execute** is the method and **DIR** is the FTP command that will read the list of files from the specified directory of the remote computer and you need to use the **getChunk** method to actually retrieve the directory's information.

Inet1.Execute, "DIR

After connecting to the server, you can choose the file from the remote computer to download by using the statement below:

Inet1.Execute, , "get" & remotefile & localfile

where remotefile is the file of the remote site and localfile is the file of the local system. However, very often you need to provide the full path of the local file, which you can do that by modifying the above syntax to the following syntax:

Inet1.Execute , , "get" & remotefile & localpath & remotefile

The above statements will ensure that the remote file will be downloaded to the location specified by the localpath and the file downloaded will assume the same name as the remote file. For example, if the remote file is **readme.txt** and the localpath is **C:\temp** , so the downloaded file will be saved in **C:\temp\readme.txt.**

In order to monitor the status of the connection, you can use the **StateChanged** event that is associated with Inet1 together with a set of the state constants that are listed in the following table.

Constant	Value	Description
icHostResolvingHost	1	The control is looking up the IP address of the

		specified host computer.
icHostResolved	2	The control successfully found the IP address of the specified host computer.
icConnecting	3	The control is connecting to the host computer.
icConnected	4	The control successfully connected to the host computer.
icRequesting	5	The control is sending a request to the host computer.
icRequestSent	6	The control successfully sent the request.
icReceivingResponse	7	The control is receiving a response from the host computer.
icResponseReceived	8	The control successfully received a response from the host computer.
icDisconnecting	9	The control is disconnecting from the host computer.
icDisconnected	10	The control successfully disconnected from the host computer.
icError	11	An error occurred in communicating with the host computer.
icResponseCompleted	12	The request has been completed and all data has been received.

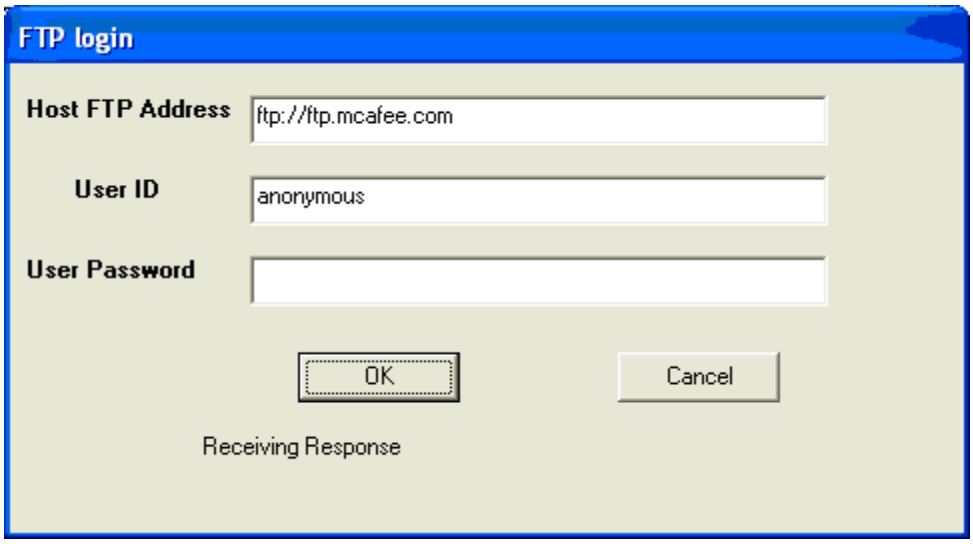
Under the StateChanged event, you use the Select Case…End Select statements to notify the users regarding the various states of the connection. The procedure is shown below:

```
Private Sub Inet1_StateChanged(ByVal State As Integer)
    Select Case State
    Case icError
    MsgBox Inet1.ResponseInfo, , "File failed to transfer"
    Case icResolvingHost
    Label6.Caption = "Resolving Host"
    Case icHostResolved
```

```
            Label6.Caption = "Host Resolved"
            Case icConnecting
            Label6.Caption = "Connecting Host"
            Case icConnected
            Label6.Caption = "Host connected"
            Case icReceivingResponse
            Label6.Caption = "Receiving Response"
            Case icResponseReceived
            Label6.Caption = "Got Response"
            Case icResponseCompleted
            Dim data1 As String
            Dim data2 As String
            MsgBox "Download Completed"
            End Select
    End Sub
```

The FTP program that I have created contains a form and a dialog box. The dialog box can be added by clicking on the Project item on the menu bar and then selecting the Add Form item on the drop-down list. You can either choose a normal dialog box or a login dialog box. The function of the dialog box is to accept the FTP address, the username and the password and then to connect to the server. After successful login, the dialog box will be hidden and the main form will be presented for the user to browse the remote directory and to choose certain files to download.

The interface of the login dialog is shown on the right.



The program for the login dialog is,

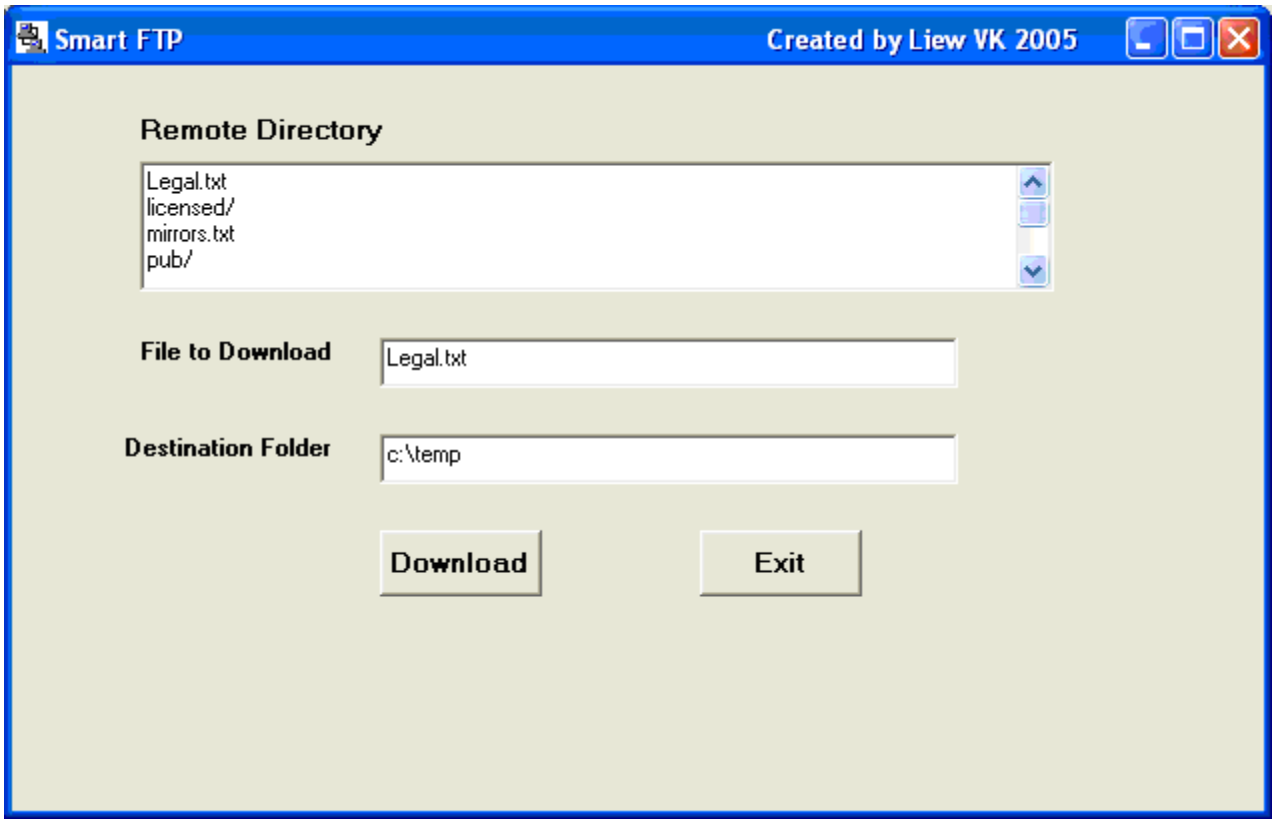
```
Option Explicit

Private Sub OKButton_Click()
Inet1.URL = Text1.Text
Inet1.UserName = Text2.Text
Inet1.Password = Text3.Text
Inet1.Execute , "DIR"
Form1.Show
Dialog.Hide
End Sub

Private Sub Inet1_StateChanged(ByVal State As Integer)
Select Case State
        Case icError
                MsgBox Inet1.ResponseInfo, , "File failed to transfer"
        Case icResolvingHost
                Label6.Caption = "Resolving Host"
        Case icHostResolved
                Label6.Caption = "Host Resolved"
```

```
        Case icConnecting
                Label6.Caption = "Connecting Host"
        Case icConnected
                Label6.Caption = "Host connected"
        Case icReceivingResponse
                Label6.Caption = "Receiving Response"
        Case icResponseReceived
                Label6.Caption = "Got Response"
        Case icResponseCompleted
                Dim data As String
                Dim data1 As String

                MsgBox "Transfer Completed"
                 Do
                        data1 = Inet1.GetChunk(1024, icString)
                        data = data & data1

                        Loop While Len(data1) <> 0
                        Form1.Text6.Text = data
    End Select
    End Sub
    Private Sub CancelButton_Click()
        Text1.Text = ""
        Text2.Text = ""
        Text3.Text = ""
    End Sub
```

retrieve

The statement data1 = Inet1.GetChunk (1024, icString) is to use the getChunk method to grab information of the remote directory and then display the files of the directory in Textbox6.

After logging in, the main form will be presented as shown in Figure 30.3



Lesson 35: Errors Handling in Visual Basic

35.1 Introduction

Error handling is an essential procedure in Visual Basic programming because it can help make the program error-free. An error-free program can run smoothly and efficiently, and the user does not have to face all sorts of problems such as program crash or system hang.

Errors often occur due to incorrect input from the user. For example, the user might make the mistake of attempting to ask the computer to divide a number by zero which will definitely cause system error. Another example is the user might enter a text (string) to a box that is designed to handle only numeric values such as the weight of a person, the computer will not be able to perform arithmetic calculation for text therefore will create an error. These errors are known as synchronous errors.

Therefore a good programmer should be more alert to the parts of program that could trigger errors and should write errors handling code to help the user in managing the errors.

Writing errors handling code should be considered a good practice for Visual Basic programmers, so do try to finish a program fast by omitting the errors handling code. However, there should not be too many errors handling code in the program as it create problems for the programmer to maintain and troubleshoot the program later.

35.2 Writing the Errors Handling Code

We shall now learn how to write errors handling code in Visual Basic. The syntax for errors handling is

On Error GoTo *program_label*

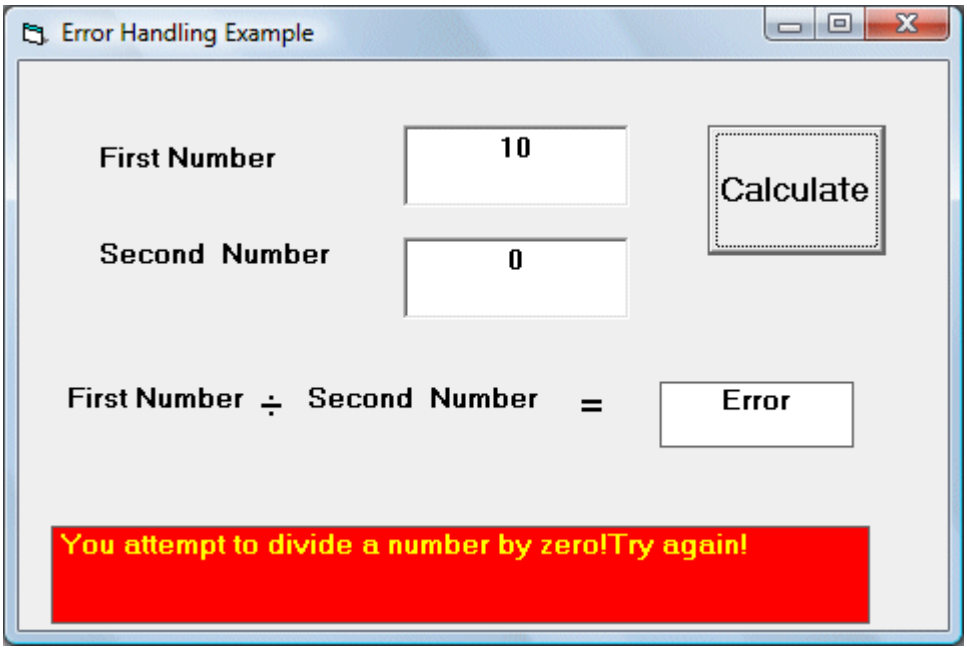
where ***program_label*** is the section of code that is designed by the programmer to handle the error committed by the user. Once an error is detected, the program will jump to the ***program_label*** section for error handling.

Example 35.1: Division by Zero

```
Private Sub CmdCalculate_Click()

Dim firstNum, secondNum As Double
firstNum = Txt_FirstNumber.Text
secondNum = Txt_SecondNumber.Text
On Error GoTo error_handler
Lbl_Answer.Caption = firstNum / secondNum
Exit Sub        'To prevent error handling even the inputs are valid

error_handler:
Lbl_Answer.Caption = "Error"
Lbl_ErrorMsg.Visible = True
Lbl_ErrorMsg.Caption = " You attempt to divide a number by zero!Try again!"

End Sub

Private Sub Txt_FirstNumber_GotFocus()

Lbl_ErrorMsg.Visible = False
```

```
End Sub

Private Sub Txt_SecondNumber_GotFocus()

Lbl_ErrorMsg.Visible = False

End Sub
```

The Output Window



Explanation:

In this example, you design the interface as above. Name the first textbox as Txt_FirstNumber and the second textbox as Txt_SecondNumber. Insert one command button as label it as Calculate. Insert one label and name it as Lbl_Answer to display the answers. If the user enter 0 in the second textbox as shown above, the program will jump to the label **error_handler**, and the procedure that is executed. It will show an error in the Txt_Answer label and an error message in the Lbl_ErrorMsg label.

Notice that Exit sub after the division. It is important because it can prevent the program to execute the error_handler code even though the user does not enter zero in the second textbox.

Lastly, after the error message appeared, the user will click on the textboxes again. When this occur, the error message will disappea both from the answer label and error message label. This is achieved by using he event procedure GotFocu, as shown in the code.

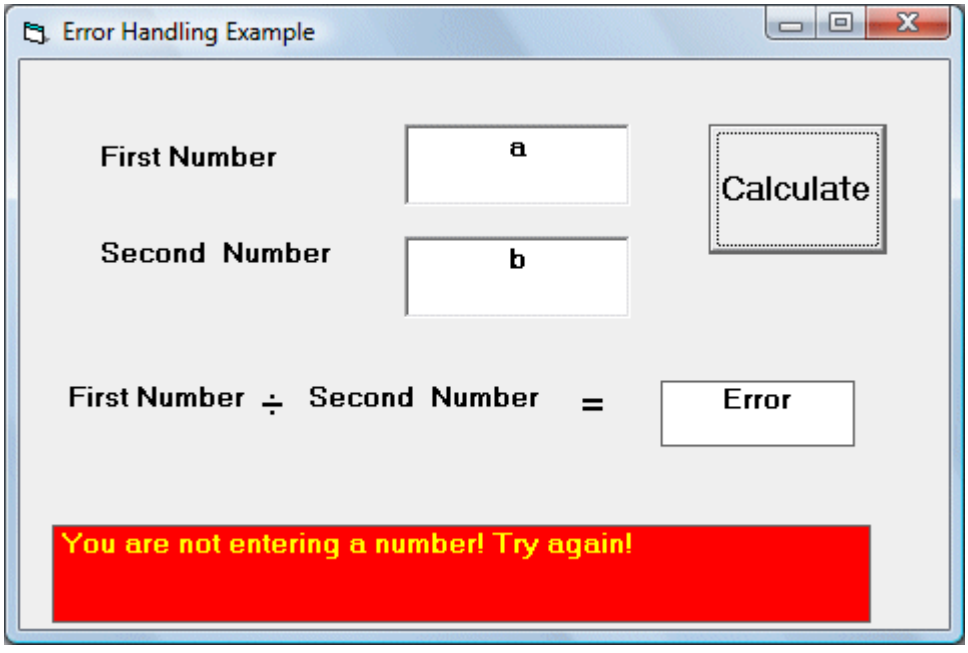
Example 35.2: **Nested Error Handling Procedure**

By referring to Example 28.1, we need to consider other errors probably will be made by the user, such as entering non-numeric inputs like letters. Therefore, we need to write error handling code for this error too. It should be put in the first place as soon as the user input something in the textboxes. And the error handler label ***error_handler1*** for this errror should be put after the ***error_handler2*** label. This means the second error handling procedure is **nested** within the first error handling procedure. Notice that you have to put an Exit Sub for the second error handling procedure to prevent to execute the first error handling procedure again. The code is as follow:

```
Private Sub CmdCalculate_Click()

Dim firstNum, secondNum As Double
On Error GoTo error_handler1
firstNum = Txt_FirstNumber.Text
secondNum = Txt_SecondNumber.Text
On Error GoTo error_handler2
Lbl_Answer.Caption = firstNum / secondNum
Exit Sub 'To prevent errror handling even the inputs are valid

error_handler2:
Lbl_Answer.Caption = "Error"
Lbl_ErrorMsg.Visible = True
Lbl_ErrorMsg.Caption = " You attempt to divide a number by zero!Try again!"
Exit Sub
error_handler1:
```

```
Lbl_Answer.Caption = "Error"
Lbl_ErrorMsg.Visible = True
Lbl_ErrorMsg.Caption = " You are not entering a number! Try again!"

End Sub

Private Sub Txt_FirstNumber_GotFocus()

Lbl_ErrorMsg.Visible = False

End Sub


Private Sub Txt_SecondNumber_GotFocus()

Lbl_ErrorMsg.Visible = False

End Sub
```

The Output window



Lesson 36: Compiling and Distributing Your Programs

36.1 Compiling your Visual Basic Program

Once your have completed a VB program, you can compile the program to run as a standalone windows application, without having to launch the Visual Basic IDE. However, before you compile your program, you have to debug your program to make sure it is errors free. Once the program is compiled into an EXE file (executable file), you can not debug it anymore. If you wish to do so, you have to correct the errors and recompile it.

To start compiling your program, click on the menu File and select Make Project1.exe, as show in Figure 36.1. When you click on Make Project1.exe , the Make Project dialog box will appear, as shown in Figure 36.2. In this dialog box, you can select the project you wish to compile. In this example, the project I chose to compile is reversi. The option button in this dialog box let you customize the program you are going to compile. For example, you can enter the title of the program , the program's version and your company name. Clicking on the compile tab will let you decide the kind of code you wish to compile. The default option is native code and it is the best option because it normally runs faster. It requires fewer files to run, particular the VB DLL files. Once you have done that, you can click the OK button to compile the program. Now you program can run as a standalone application. You can start your program without launching the Visual Basic IDE.

Figure 36.1

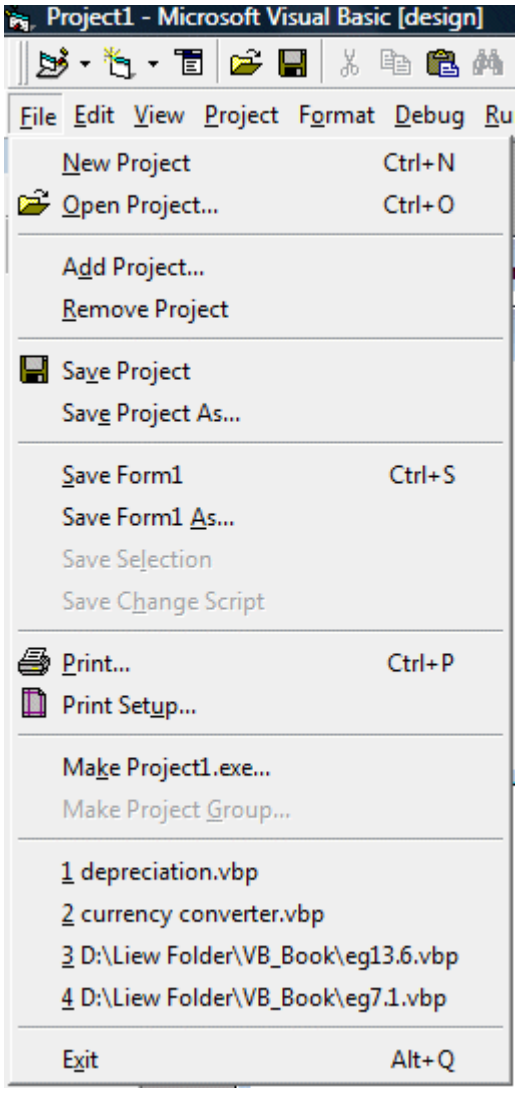

Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Ru
New Project Ctrl+N
Open Project... Ctrl+O
Add Project...
Remove Project
Save Project
Save Project As...
Save Form1 Ctrl+S
Save Form1 As...
Save Selection
Save Change Script
Print... Ctrl+P
Print Setup...
Make Project1.exe...
Make Project Group...
1 depreciation.vbp
2 currency converter.vbp
3 D:\Liew Folder\VB_Book\eg13.6.vbp
4 D:\Liew Folder\VB_Book\eg7.1.vbp
Exit Alt+Q

Figure 36.2

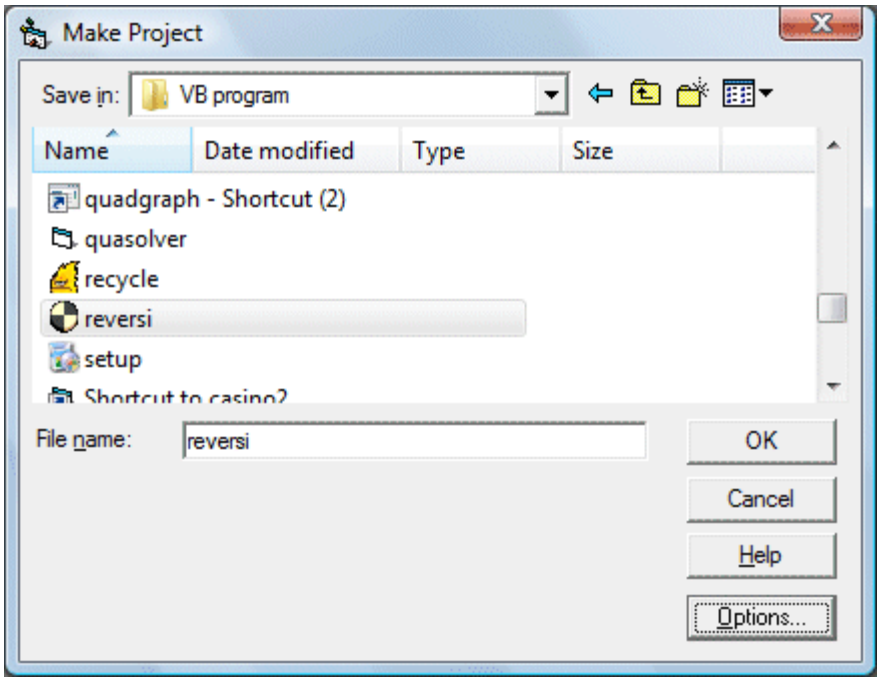
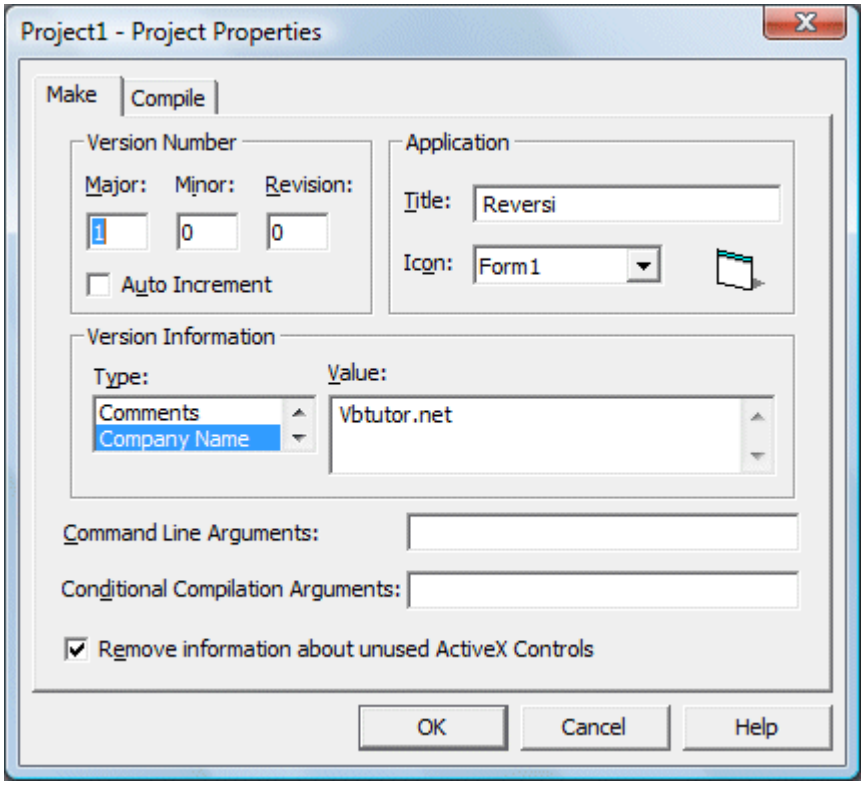
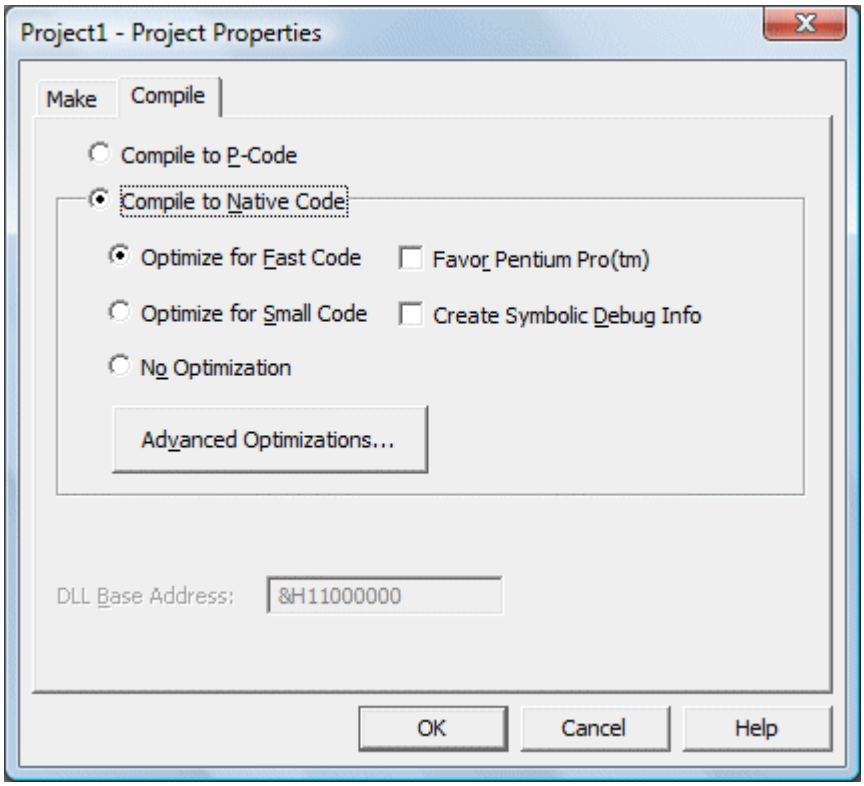

Make Project
Save in:
VB program
Name
Date modified
Type
Size
quadgraph - Shortcut (2)
quasolver
recycle
reversi
setup
File name:
reversi
OK
Cancel
Help
Options...

Figure 36.3


Project1 - Project Properties
Make
Compile
Version Number
Major:
Minor:
Revision:
1
0
0
Auto Increment
Application
Title:
Reversi
Icon:
Form1
Version Information
Type:
Value:
Comments
Company Name
Vbtutor.net
Command Line Arguments:
Conditional Compilation Arguments:
Remove information about unused ActiveX Controls
OK
Cancel
Help

36.2 Distributing Your Programs

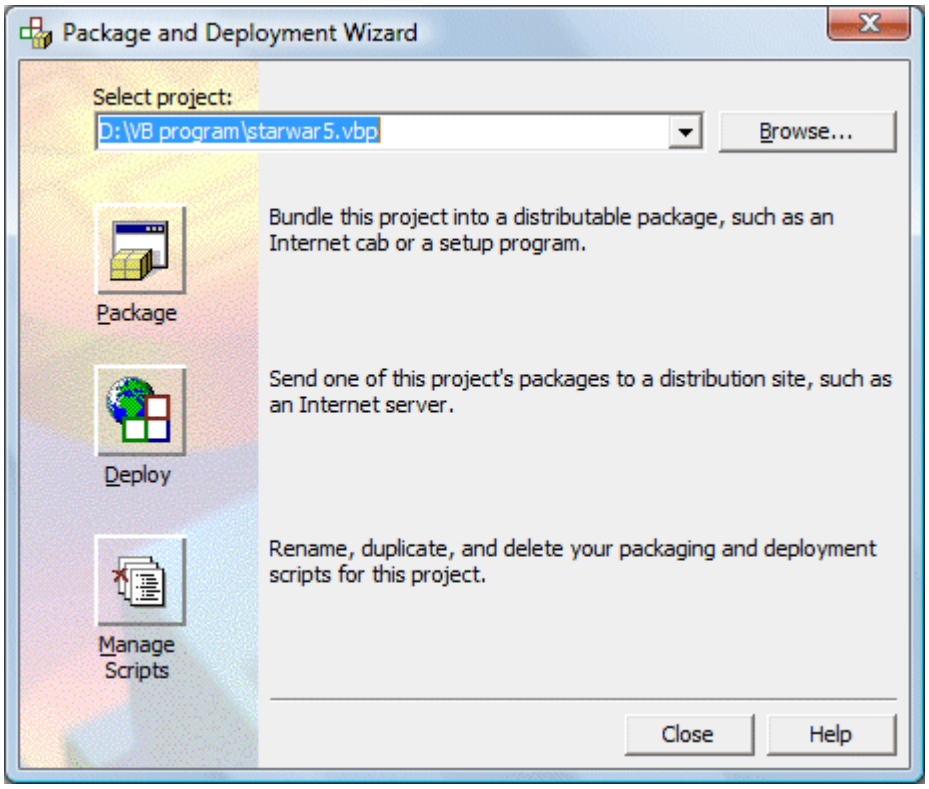
Figure 36.4



After successfully created a VB program, you might want to market your product, either online or offline. This means that you need to create a package that can be distributed to your potential customers. The package created can be distributed using CD ROM, diskette or the Internet. The package will allow the user to install the program to install in the computer with the standard setup routine.

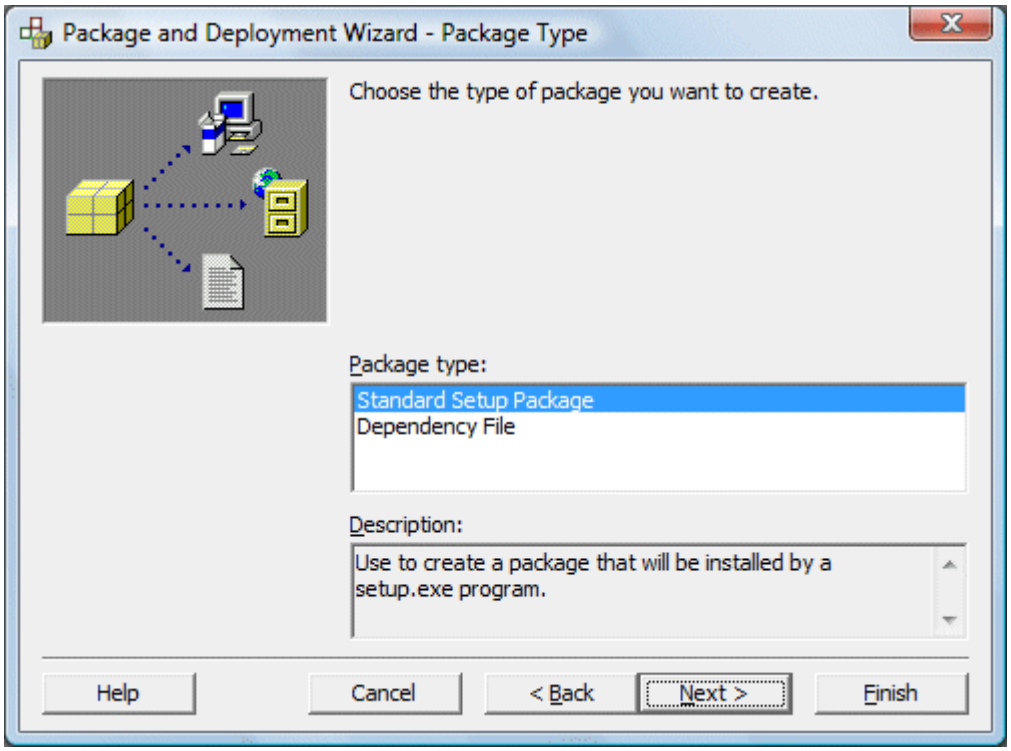
To create the distributable package, you can use the Package and Development Wizard that came with Visual Basic 6. The main purpose of this wizard is to create a setup program that can be used to install the application. Off course, it also does many other jobs like compiling your application and compresses the files for easy distribution.

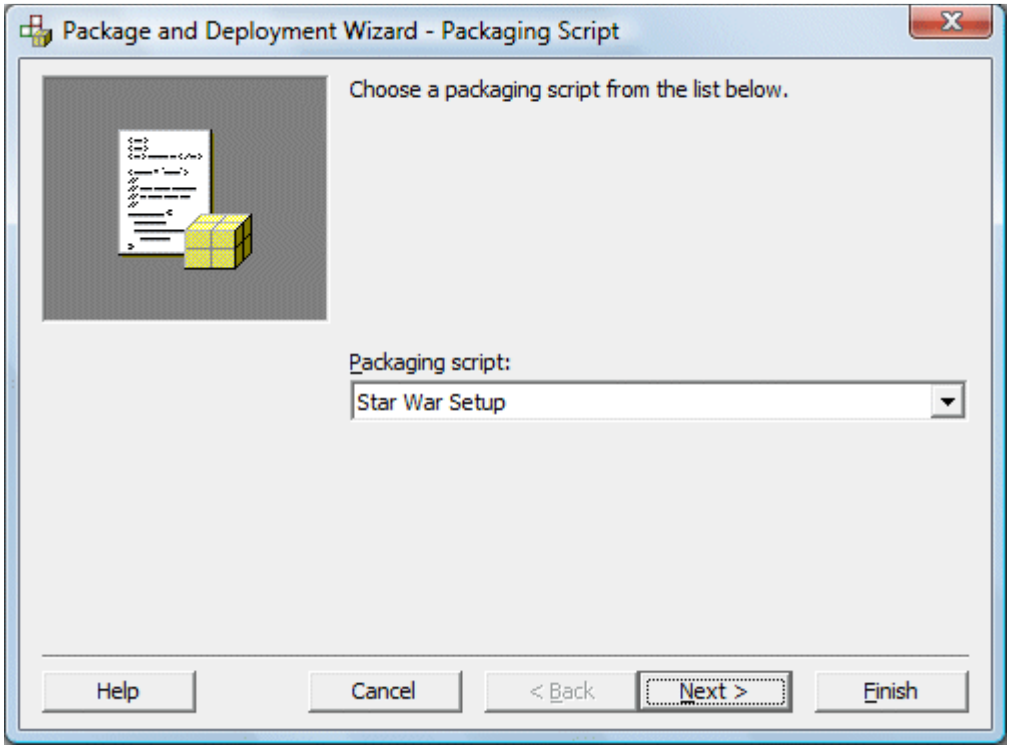
When you start the Package and Development Wizard, you will be presented with the following dialog box:



First of all, you need to select the project you want to package. Here I have selected the starwar.vbp project. Next, you need to select one of the three options. Here, I suggest you select the first option to let the wizard create the installation package for you to distribute it using CD ROM or the Internet.

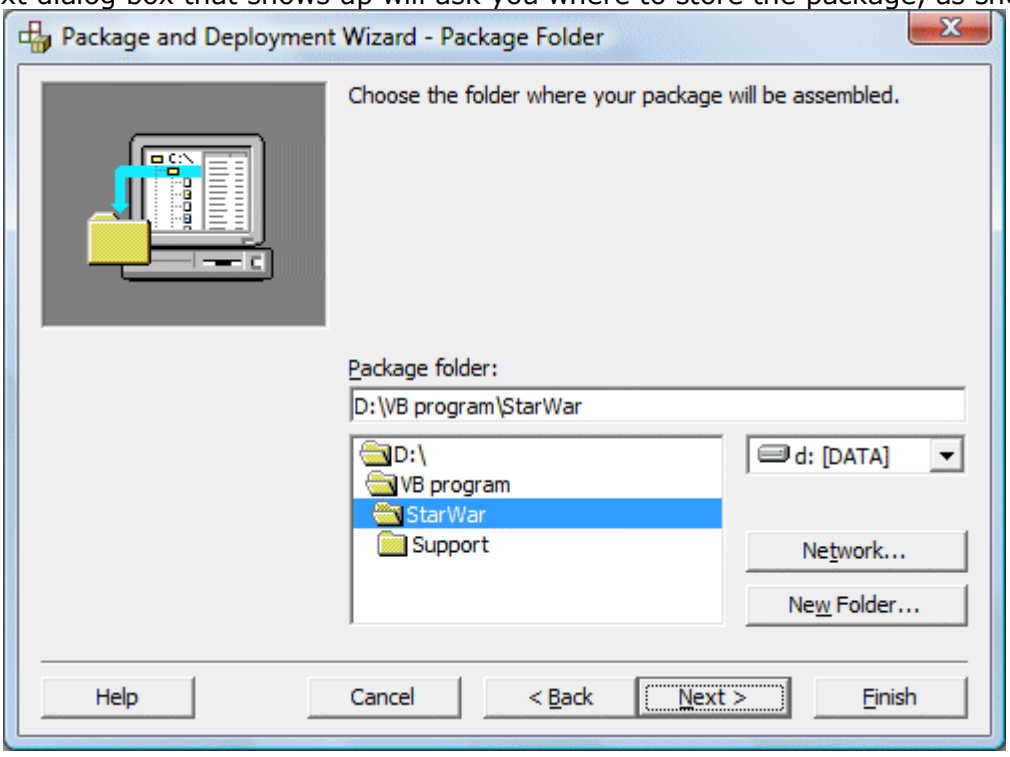
Once you click the package option, you will see the following dialog box where you are asked to choose a packaging script:





After you click next, you will see the following dialog box where you will be asked to choose a packaging type. Normally we choose the Standard Setup Package.

The next dialog box that shows up will ask you where to store the package, as shown below:



The next dialog box will show you the files that will be included in the package.

When you click the Finish button, the package will be created and ready for distribution. Here is the packaged files for the starwar program for download at

Setup.exe

SETUP.LST

starwar5.CAB

You must download all the three files into a folder and then run the setup program.

Lesson 37: Creating Menus for Your Applications

Menu bar is the standard feature of most windows applications. The main purpose of the menus is for easy navigation and control of an application. Some of the most common menu items are File, Edit, View, Tools, Help and more. Each item on the main menu bar also provide a list of options or in the form of a pull-down menu. When you create a Visual Basic 6 program, you need not include as many menu items as a full fledge Windows application

such as Microsoft Words. What you need is to include those menu items that can improve the ease of using your program by the user, and not to confuse the user with unnecessary items. Adding menu bar is relatively easy to accomplish in Visual Basic. There are two ways to add menus to your application, one way is to use the Visual Basic's Application Wizard and the other way is to use the menu editor.

37.1 Adding Menu Bar Using Visual Basic's Application Wizard

The easiest way to add menu bar to your application is by using Visual Basic's Application Wizard. This wizard allows the user to insert fully customized standard windows menus into his or her application. To start using Visual Basic's Application Wizard, you click on the Application Wizard icon at the Visual Basic new project dialog box, as shown below:

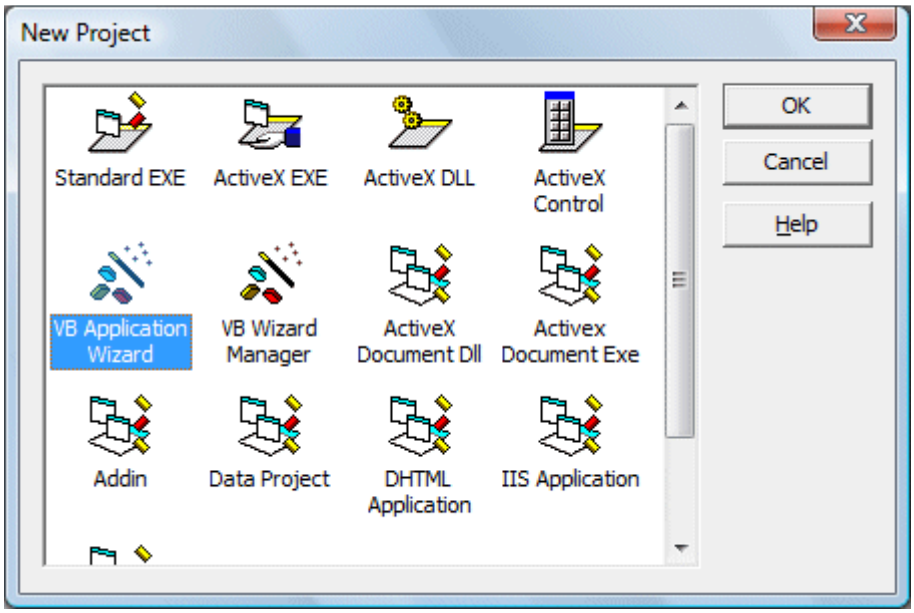


Figure 37.1: New Project Window

When you click on the VB Application wizard, the introduction dialog box will appear, as shown in Figure 37.1. As you are not loading any default setting, just click on the Next button. After clicking the Next button, the interface type dialog box will be displayed, as shown in Figure 37.3. There are three choices of interface for your project, as we currently not creating a Multiple Document Interface (MDI), we choose Single Document Interface (SDI). You can also type the project name in the textbox below, here I am using MyFirstMenu. After clicking the Next button, you will be presented with a list of menus and submenus that you would like to add them to your application. Check to select a menu item

and uncheck to unselect a menu item. Let say we choose all the menus and click next, then you will get an interface will File, Edit, View and Help menus. such as that shown in Figure 37.5

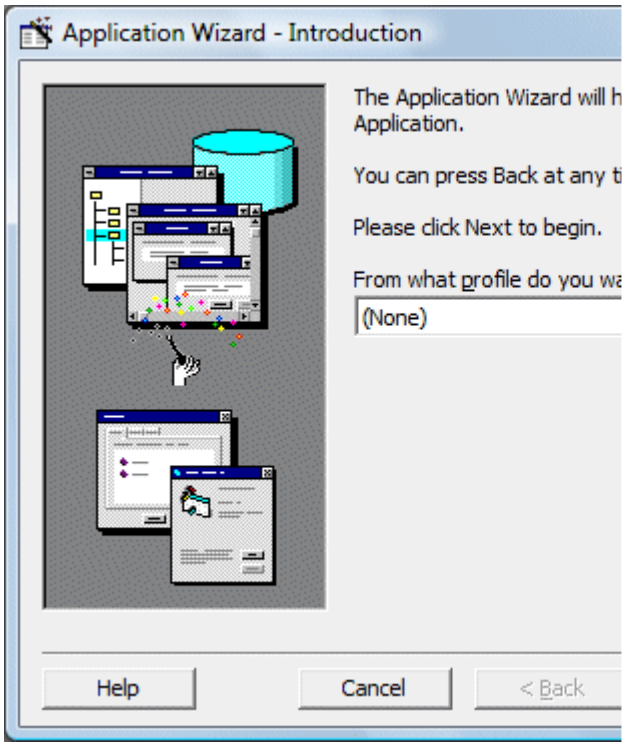


Figure 37.2

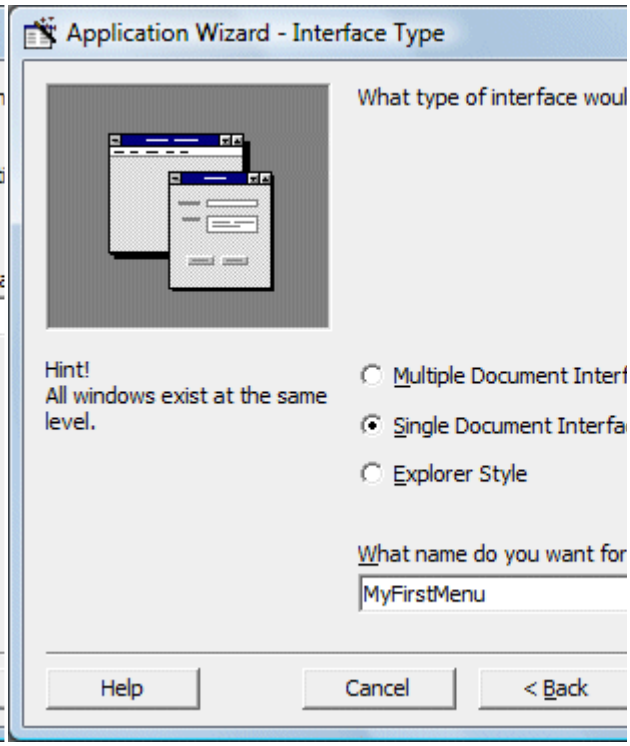


Figure 37.3

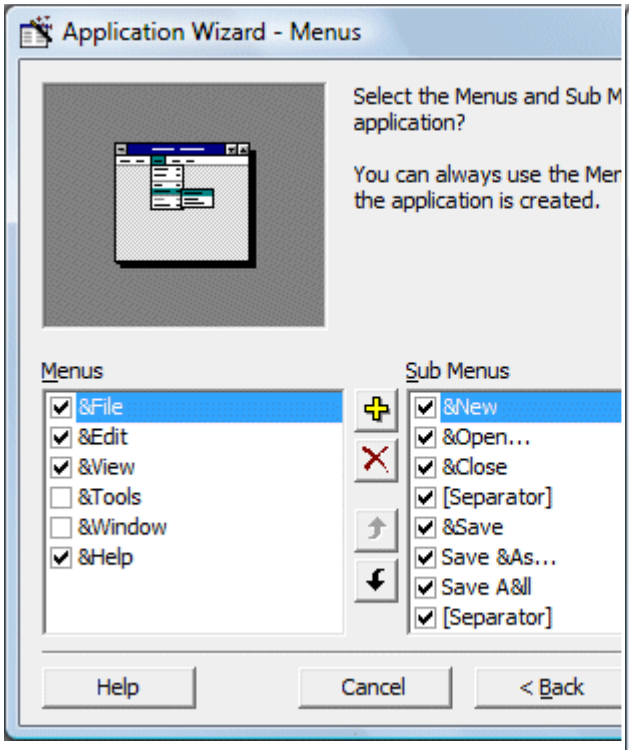


Figure 37.4

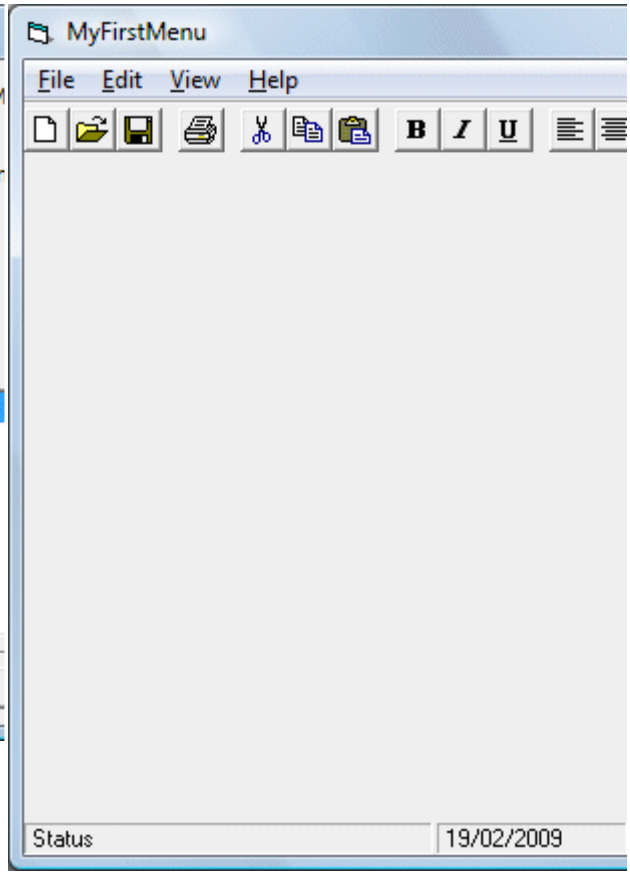


Figure 37.5

When you click on any menu item, a list of drop-down submenu items will be displayed. For example, if you click on the File menu, the list of submenu items such as New, Open, Save, Save As and more will be displayed, as shown in Figure 37.6

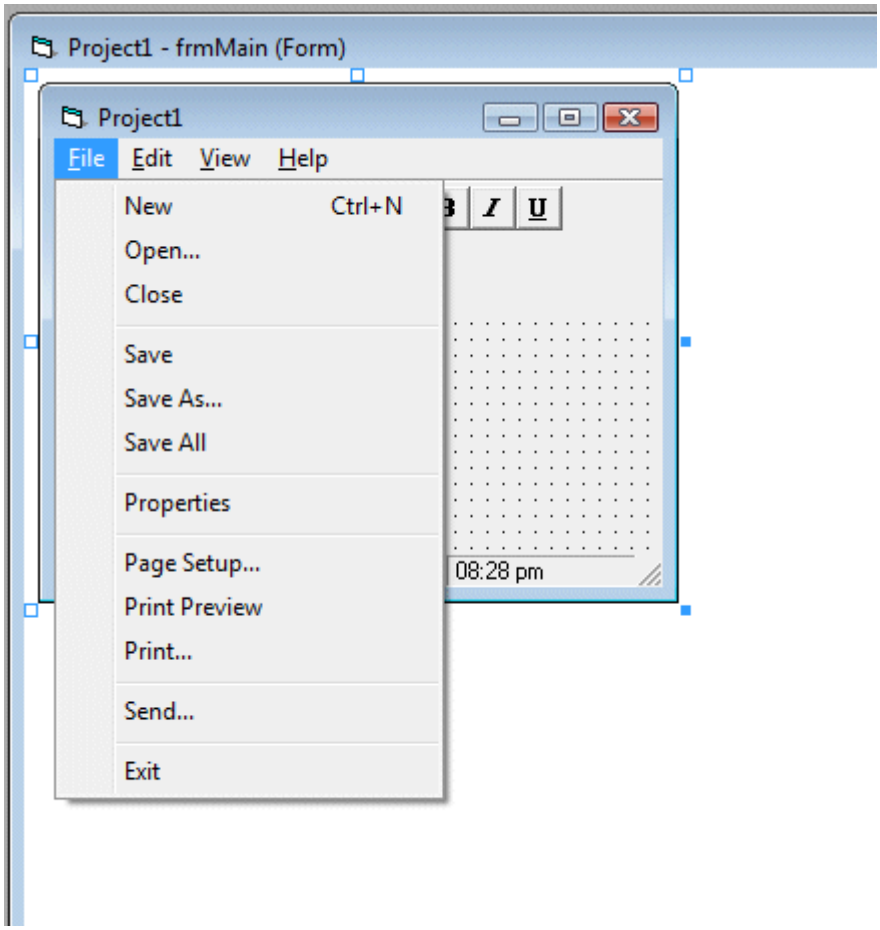


Figure 37.6

Clicking on any of the dropped down menu item will show the code associated with it, and this is where you can modify the code to suit your programming needs. For example, clicking on the item Open will reveal the following code:

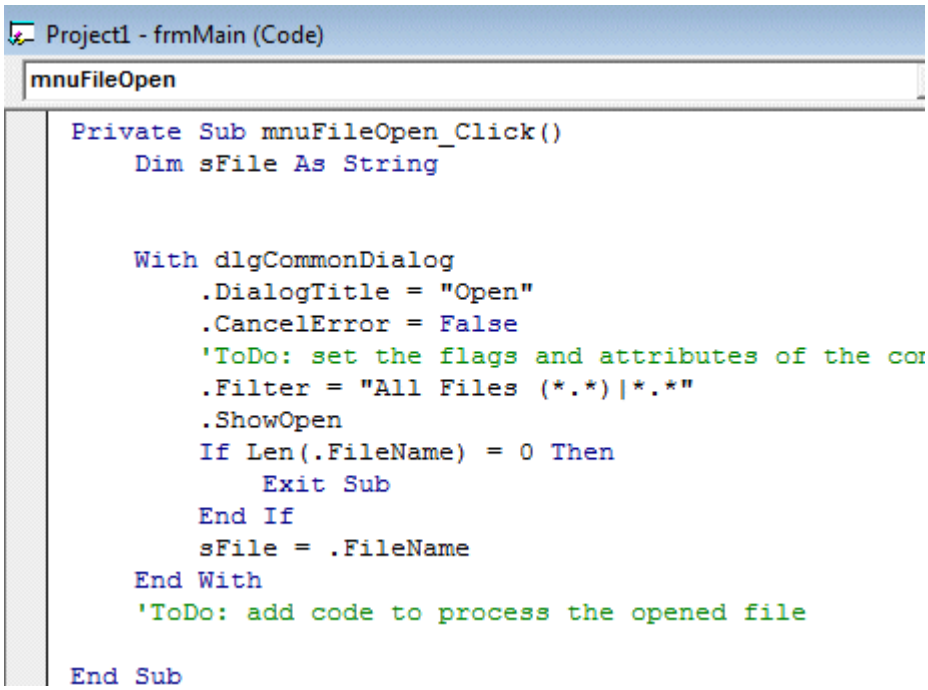


Figure 37.7

Now, I will show you how to modify the code in order to open a graphic file and display it in an image box. For this program, you have to insert a Image box into the form. Next add the following lines so that the user can open graphic files of different formats.

.Filter = "Bitmaps(*.BMP)|*.BMP|Metafiles(*.WMF)|*.WMF|Jpeg Files(*.jpg)|*.jpg|GIF Files(*.gif)|*.gif|Icon Files(*.ico)|*.ico|All Files(*.*)|*.*".

Then, you need to load the image into the Image box with the following code:

Image1.Picture = LoadPicture(.FileName)

Also set the Stretch property of the Image box to true so that the image loaded can resize by itself. Please note that each menu item is a special control, so it has a name too. The name for the menu File in this example is mnuFileOpen.

The full code is as follows:

Private Sub mnuFileOpen_Click()
Dim sFile As String

With dlgCommonDialog
.DialogTitle = "Open"
.CancelError = False
'ToDo: set the flags and attributes of the common dialog control
.Filter = "Bitmaps(*.BMP)|*.BMP|Metafiles(*.WMF)|*.WMF|Jpeg Files(*.jpg)|*.jpg|GIF Files(*.gif)|*.gif|Icon Files(*.ico)|*.ico|All Files(*.*)|*.*"
.ShowOpen

```
Image1.Picture = LoadPicture(.FileName)

If Len(.FileName) = 0 Then
Exit Sub
End If
sFile = .FileName
End With
'ToDo: add code to process the opened file

End Sub
```

When you run the program and click on the File menu and then the submenu Open, the following Open dialog box will be displayed, where you can look for graphic files of various formats to load it into the image box.

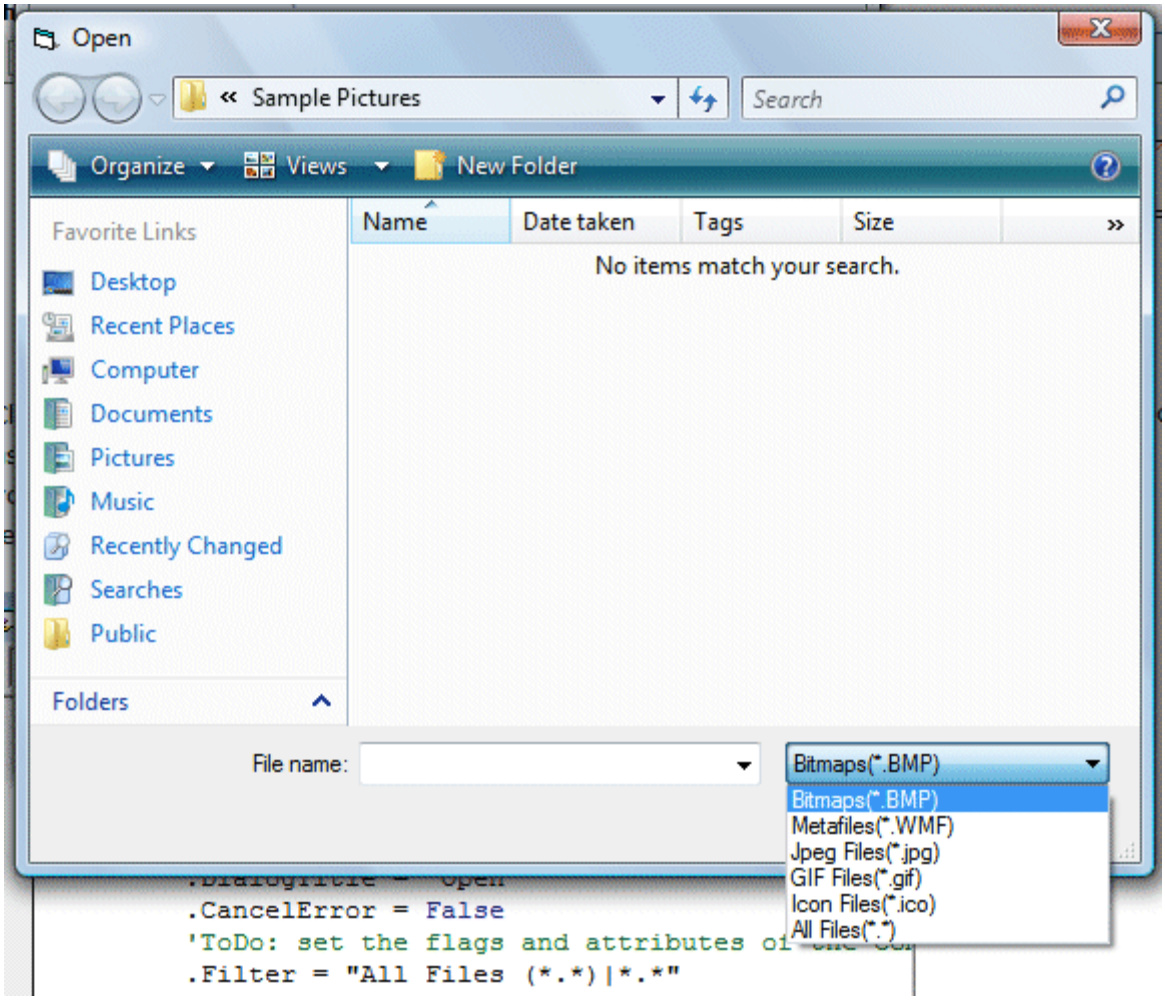


Figure 37.8

For example, selecting the jpeg file will allow you to choose the images of jpeg format.

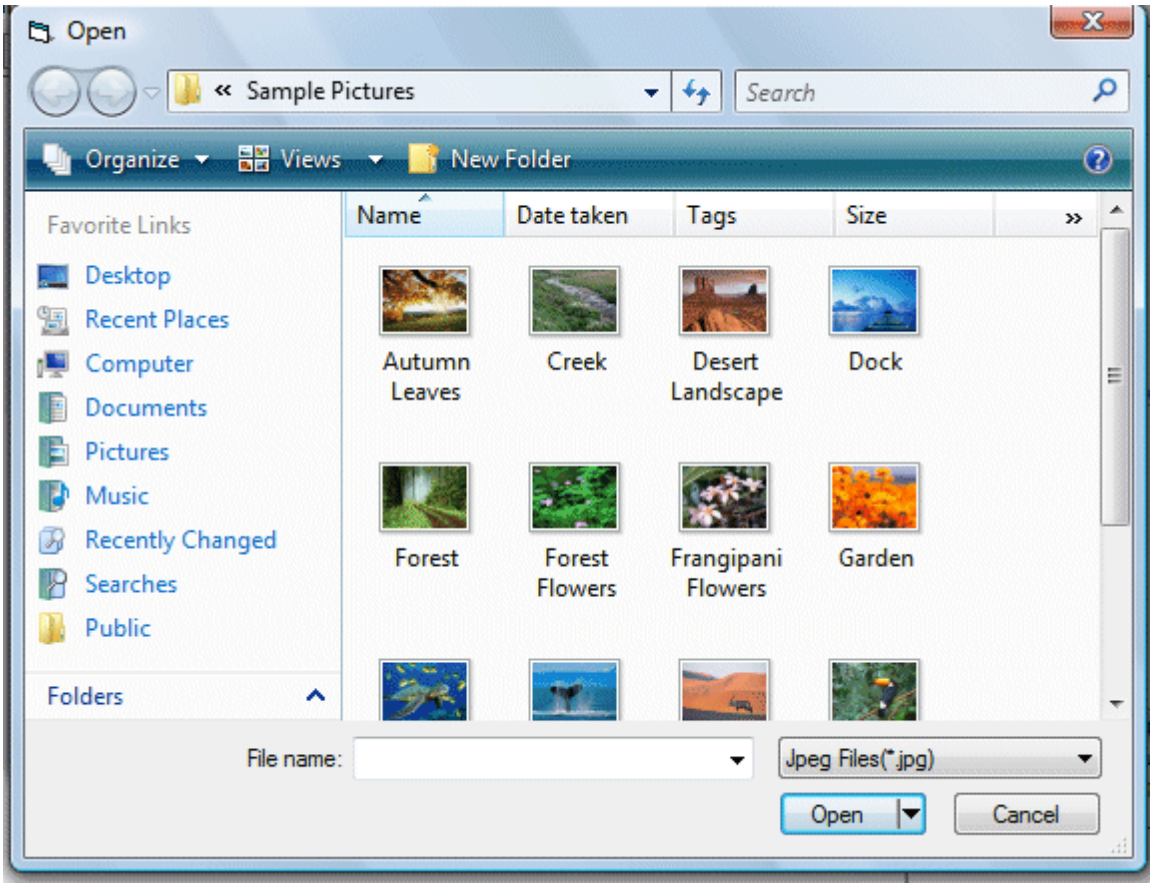
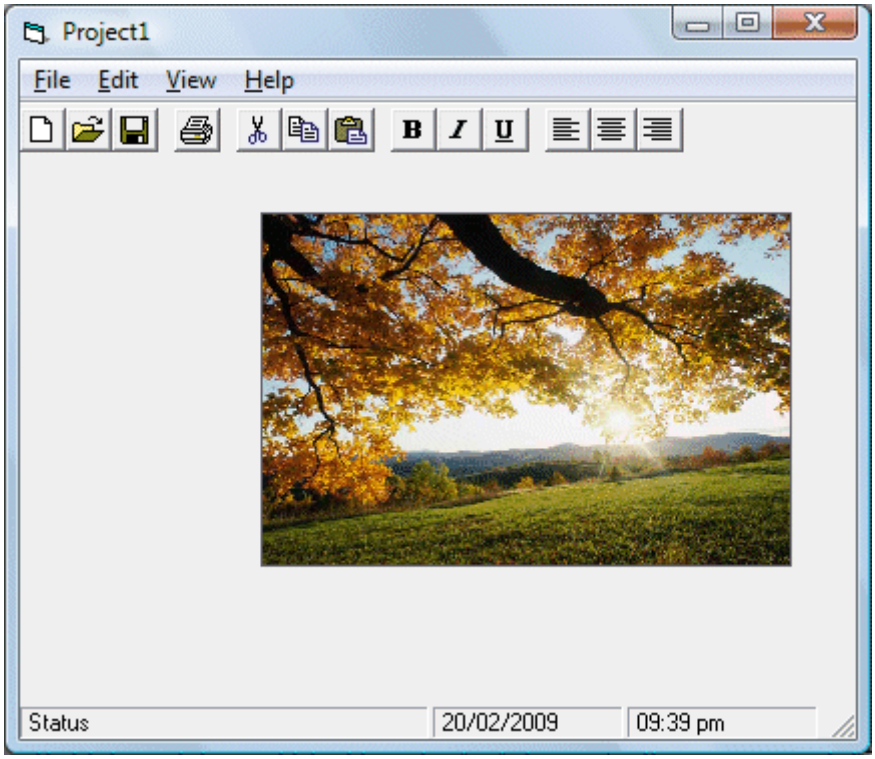


Figure 37.9

Clicking on the particular picture will load it into the image box, as shown below.



37.2: Adding Menu Bar Using Menu Editor

To start adding menu items to your application, open an existing project or start a new project, then click on Tools in the menu bar of the Visual Basic IDE and select Menu Editor. When you click on the Menu Editor, the Menu Editor dialog will appear. In the Menu Editor dialog , key in the first item File in the caption text box. You can use the ampersand (&) sign in front of F so that F will be underlined when it appears in the menu, and F will become the hot key to initiate the action under this item by pressing the Alt key and the letter F. After typing &File in the Caption text box, move to the name textbox to enter the name for this menu item, you can type in mnuFile here. Now, click the Next button and the menu item &File will move into the empty space below, as shown in the following diagram:

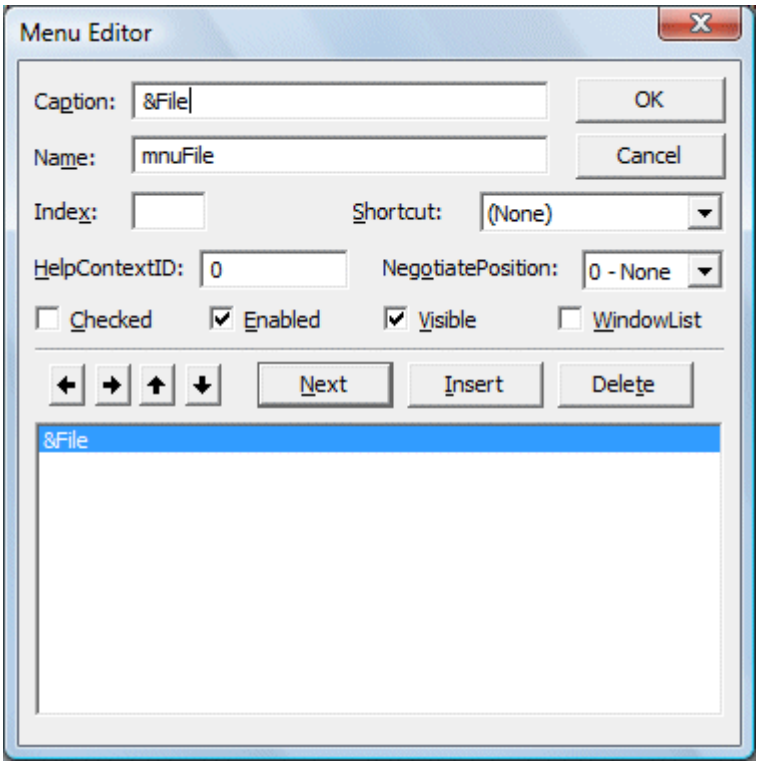
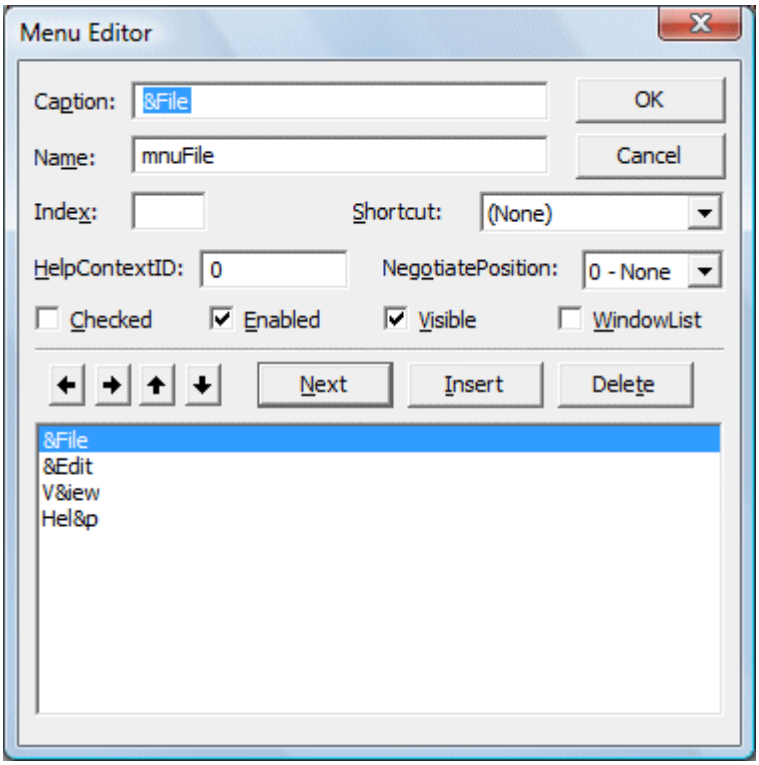


Figure 37.11

You can then add in other menu items on the menu bar by following the same procedure, as shown in the diagram below:



when you click Ok, the menu items will be shown on the menu bar of the form.

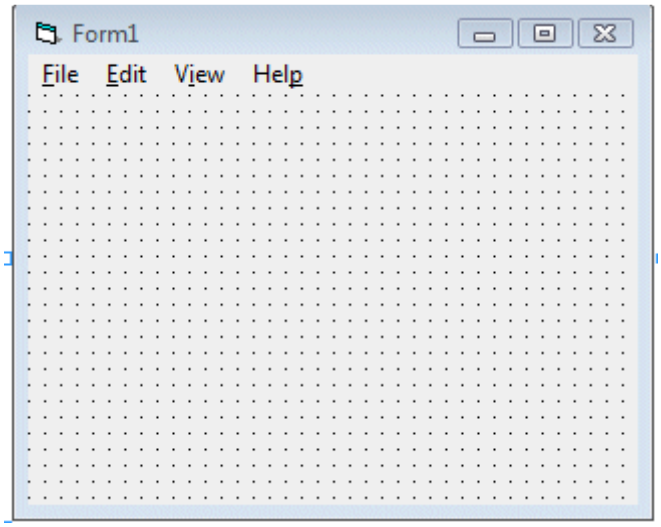


Figure 37.13

Now, you may proceed to add the sub menus. In the Menu Editor, click on the Insert button between File and Exit and then click the right arrow key, and the dotted line will appear. This shows the second level of the menu, or the submenu. Now key in the caption and the name. Repeat the same procedure to add other submenu items. Here, we are adding New, Open, Save, Save As and Exit.

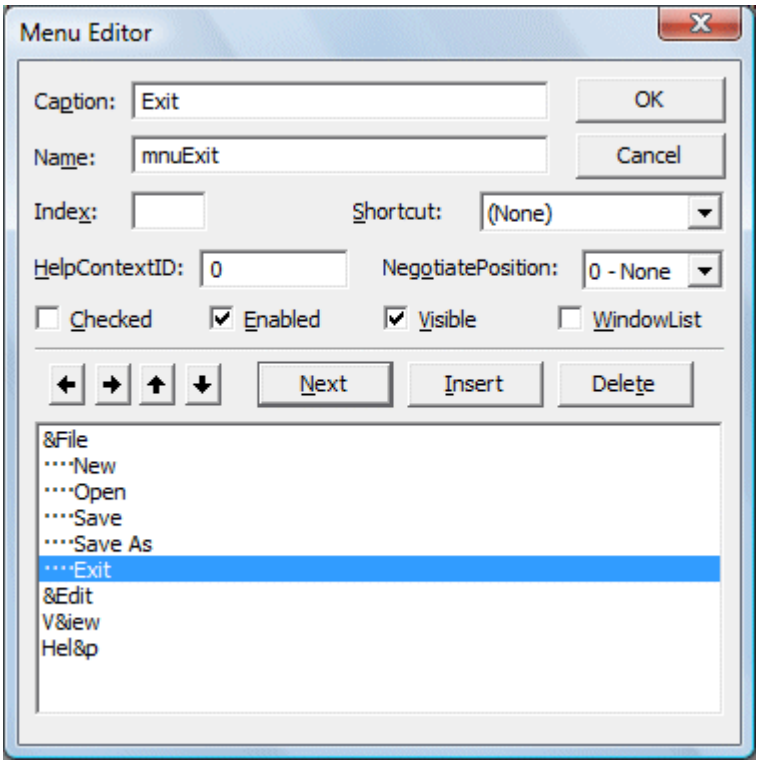


Figure 37.14

Now click the OK button and go back to your form. You can see the dropped down submenus when you click on the item File, as shown.

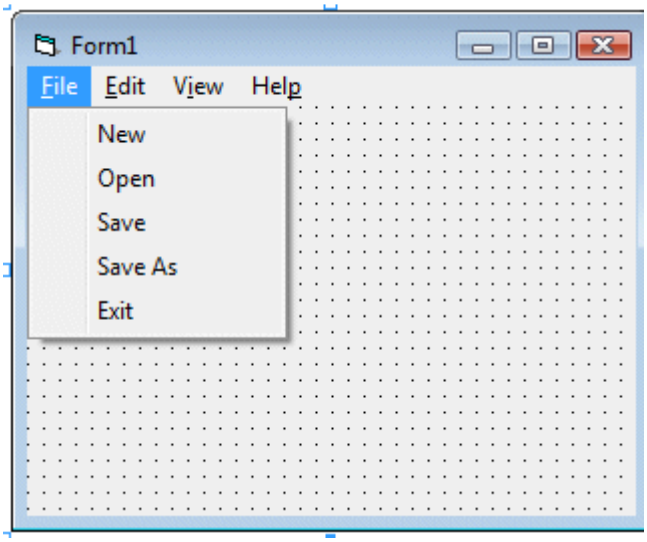


Figure 37.15

Finally, you can enter the code by clicking on any of the submenu items. You can enter code such as that shown in section 37.1

Lesson 38: Keyboard Handling

In previous lessons, we have only learned how to trigger events or control program flow by clicking the mouse. In this chapter, you will learn how to use the keyboard to trigger an event using the keyboard beside using the mouse. When the user press a key on the keyboard, it will trigger an event or a series of events. These events are called the keyboard events. In Visual Basic, the three basic event procedure to handle the key events are KeyPress, Keydown and KeyUp

38.1 ASCII

The key event occurs when the user presses any key that corresponds to a certain alphanumeric value or an action such as Enter, spacing, backspace or so on. Each of those values or actions are represented by a set of codes known as the ASCII . ASCII stands for American Standard Code for Information Interchange. ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort. ASCII was developed a long time ago and now the non-printing characters are rarely used for their original purpose.In order to write code for the Key events , we need to know the ASCII and the corresponding values. Some of the commond ASCII values are shown in Table 38.1.

ASCII	Chr	ASCII	Chr	ASCII	Chr
8	Backspace	61	=	98	b
13	Carriage Return or Enter key	62	>	99	c
32	Space	63	?	100	d
33	!	64	@	101	e
34	"	65	A	102	f
35	#	66	B	103	g
36	$	67	C	104	h
37	%	68	D	105	i
38	&	69	E	106	j
39	'	70	F	107	k
40	(	71	G	108	l
41	)	72	H	109	m
42	*	73	I	110	n
43	+	74	J	111	o
44	,	75	K	112	p
45	-	76	L	113	q
46	.	77	M	114	r
47	/	78	N	115	s
48	0	79	O	116	t
49	1	80	P	117	u
50	2	81	Q	118	v
51	3	82	R	119	w
52	4	83	S	120	x
53	5	84	T	121	y
54	6	85	U	122	z
55	7	86	V	123	{
56	8	87	W	124	|
57	9	88	X	125	}
58	:	89	Y	126	~

59	;	90	Z	127	DEL
60	<	97	a		

Table 38.1: ASCII Values

For more detail table, please refer to http://www.asciitable.com/

38.2 Common Key Events Constants.

In Visual Basic 6, it employs a set of constants that correspond to the ASCII values. We can use the constants instead of the ASCII. The following tablle shows the constants and the corresponding ASCII values.

Event Constant	ASCII	Chr	Event Constant	ASCII	Chr
vbKey0	48	0	vbKeyR	82	R
vbKey1	49	1	vbKeyS	83	S
vbKey2	50	2	vbKeyT	84	T
vbKey3	51	3	vbKeyU	85	U
vbKey4	52	4	vbKeyV	86	V
vbKey5	53	5	vbKeyW	87	W
vbKey6	54	6	vbKeyX	88	X
vbKey7	55	7	vbKeyY	89	Y
vbKey8	56	8	vbKeyZ	90	Z
vbKey9	57	9	vbKeyDecimal	110	Decima point
vbKeyA	65	A	vbkeyBack	8	Backspace key
vbKeyB	66	B	vbKeyTab	9	Tab key
vbKeyC	67	C	vbkeyReturn	13	Return key(Enter key)
vbKeyD	68	D	vbKeyShift	16	Shift key
vbKeyE	69	E	vbKeyControl	17	Ctrl key
vbKeyF	70	F	vbKeyCapital	20	Caps Lock key
vbKeyG	71	G	vbKeyEscape	27	Esc key

vbKeyH	72	H	vbKeySpace	32	Space bar
vbKeyI	73	I	vbKeyInsert	45	Insert key
vbKeyJ	74	J	vbKeyDelete	46	Delete key
vbKeyK	75	K			
vbKeyL	76	L			
vbKeyM	77	M			
vbKeyN	78	N			
vbKeyO	79	O			
vbKeyP	80	P			
vbKeyQ	81	Q			

38.3 Writing code for the key events

We can write code for the three key events i.e. keyPress, KeyDown and KeyUp.

Example 38.1

```
Private Sub Form_KeyPress(KeyAscii As Integer)
If KeyAscii = 13 Then  ' 13 is the ASCII value for the Enter key
Print "You have pressed the Enter key"
Else
Print "You have pressed other key"

End If

End Sub
```

In this example, the program can detect the pressing of Enter key and the keys other than the Enter key.

Example 38.2

If you wish to detect and display the key pressed by the user, simply type the following code:

```
Private Sub Form_KeyPress(KeyAscii As Integer)

Print Chr(KeyAscii)

End Sub
```

The function Chr will convert the ASCII values to the corresponding characters as shown in the ASCII table.

Example 38.3

```
Private Sub Form_KeyPress(KeyAscii As Integer)

For i = 65 To 90
Print Chr(KeyAscii)
Next

End Sub
```

In this example, we use the **For ...Next** loop to display the alphabet A to Z by pressing any key on the keyboard.

Example 38.4

```
Private Sub Form_KeyPress(KeyAscii As Integer)

If KeyAscii = 13 Then
For i = 97 To 122
Print Chr(i)
Next
End If

End Sub
```

Lesson 39: Using the Printer-Part 1

39.1 Printing using the Printer object

In previous lessons, we have only written programs that send output to the screen and not the printer. In this lesson, we will learn how to send an output to the printer and get it printed. Sending output to the printer is a simple task in Visual Basic, it involves the use of the Printer object and the print method. The standard code of sending an output to the printer and get it printed out is as follows:

```
Private Sub Form_Load()
   Printer.Print "Welcome to Visual Basic"
End Sub
```

However, the code above only send the output to the printer without actually printing it. It will only print the output when you terminate the application. It can a very inconvenience if you need to close the program every time you want to print the output. To solve this little problem, we need to add the newpage or EndDoc method. So, add one extra line to the above code as follows:

```
Private Sub Form_Load()
   Printer.Print "Welcome to Visual Basic"

   Printer.EndDoc

End Sub
```

Beside printing messages in string form, you can actually print out other varibales including numeric values. Below is an example:

```
Private Sub Command1_Click()
Dim x, y As String, z As Variant
x = InputBox("Enter the first Number")
y = InputBox("Enter the second Number")
z = Val(x) + Val(y)
```

```
Printer.Print "The answer is" & z

Printer.EndDoc

End Sub
```

If x=3 and y=4, the printing output is "The answer is 7'

You can also use loops to send output to the printer. In the follow example, I used the For.....Next loop to print out the multiplication table.

```
Private Sub Command1_Click()

Dim i, j As Integer
For i = 2 To 9
For j = 2 To 9
Printer.Print i & "x" & j & "=" & i * j,
Next j
Printer.Print Chr(13)

Next i
Printer.EndDoc

End Sub
```

The command Printer.Print Chr(13) is equivalent to pressing the Enter and print the output on the next line. The output is as follows:

2x2=4	2x3=6	2x4=8	2x5=10	2x6=12	2x7=14	2x8=16	2x9=18
3x2=6	3x3=9	3x4=12	3x5=15	3x6=18	3x7=21	3x8=24	3x9=27
4x2=8	4x3=12	4x4=16	4x5=20	4x6=24	4x7=28	4x8=32	4x9=36
5x2=10	5x3=15	5x4=20	5x5=25	5x6=30	5x7=35	5x8=40	5x9=45
6x2=12	6x3=18	6x4=24	6x5=30	6x6=36	6x7=42	6x8=48	6x9=54
7x2=14	7x3=21	7x4=28	7x5=35	7x6=42	7x7=49	7x8=56	7x9=63
8x2=16	8x3=24	8x4=32	8x5=40	8x6=48	8x7=56	8x8=64	8x9=72
9x2=18	9x3=27	9x4=36	9x5=45	9x6=54	9x7=63	9x8=72	9x9=81

Now you might want to know whether it is possible to print the content of text file created in Visual Basic. The answer is a big "YES". Let me use Example 17.3.2 of Lesson 17. We shall add a command button to the form and rename it as cmdPrint and change the label to Print, and then double click the button to insert the follow code:

```
Private Sub CmdPrint_Click()
Printer.Print Text1.Text
Printer.EndDoc
End Sub
```

By clicking the Print button you should be able to print the content of the text box.

The full code of the program as follows:

```
Dim linetext As String

Private Sub CmdPrint_Click()
Printer.Print Text1.Text
Printer.EndDoc

End Sub

Private Sub open_Click()
CommonDialog1.Filter = "Text files{*.txt)|*.txt"
CommonDialog1.ShowOpen
If CommonDialog1.FileName <> "" Then
Open CommonDialog1.FileName For Input As #1
Do
Input #1, linetext
Text1.Text = Text1.Text & linetext
Loop Until EOF(1)
End If

Close #1
```

```
End Sub

Private Sub save_Click()
CommonDialog1.Filter = "Text files{*.txt)|*.txt"
CommonDialog1.ShowSave
If CommonDialog1.FileName <> "" Then
Open CommonDialog1.FileName For Output As #1
Print #1, Text1.Text
Close #1

End If

End Sub
```

39.2 Formatting the Output using Printer Object properties

You can format your output before sending it to the printer using a number of font related Printer object properties. Some of these properties are listed below:

FontBold, FontItalic, FontSize, FontName and FontUnderline

The code to format your printed output is illustrated in the example below:

```
Private Sub CmdPrint_Click()

Printer.FontName="Verdana"

Printer.FontSize=16

Printer.FontBold=True

Pinter.FontItalic=True

Printer.FontUndeline=True

        Printer.Print Text1.Text
        Printer.EndDoc
```

```
End Sub
```

Creating Reports in Visual Basic 6

40.1 A brief introduction to reporting tool in Visual basic 6

You have learned how to build a database in Visual Basic 6 in previous chapters, however you have not learned how to display the saved data in a report. Reports are important and useful in many respects because they provide useful and meaningful information concerning a set of data. In this chapter, we will show you how to create a report in Visual Basic 6.

In previous versions of Visual Basic 6, there is no primary reporting . Previous versions of Visual basic 6 uses Crystal Reports tool, a software from Seagate. Fortunately, Microsoft has integrated a good report writer into Visual Basic 6, so you no longer need to use Crystal Report.

40.2 Steps in building your report in Visual Basic 6

Visual Basic 6 provides you with a data report designer to create your report, it is somewhat similar to data report designer in Microsoft Access. The data report designer has its own set of controls which allow you to customize your report seamlessly. The steps in creating the report in VB6 are listed below:

Step 1: Adding Data Report

Start Visual Basic as a Standard EXE project. From the Project menu in the VBE, select Add Data Report in the dropdown menu. Now, you will be presented with the data report environment, as shown in Figure 40.1. The data report environment contains 6 controls, they are RptTextBox, RptLine, RptFunction, RptLabel, RptImage and RptShape.

You can customize your report here by adding a title to the page header using the report label RptLabel. Simply drag and draw the RptLabel control on the data report designer window and use the Caption property to change the text that should be displayed. You can also add graphics to the report using the RptImage control.

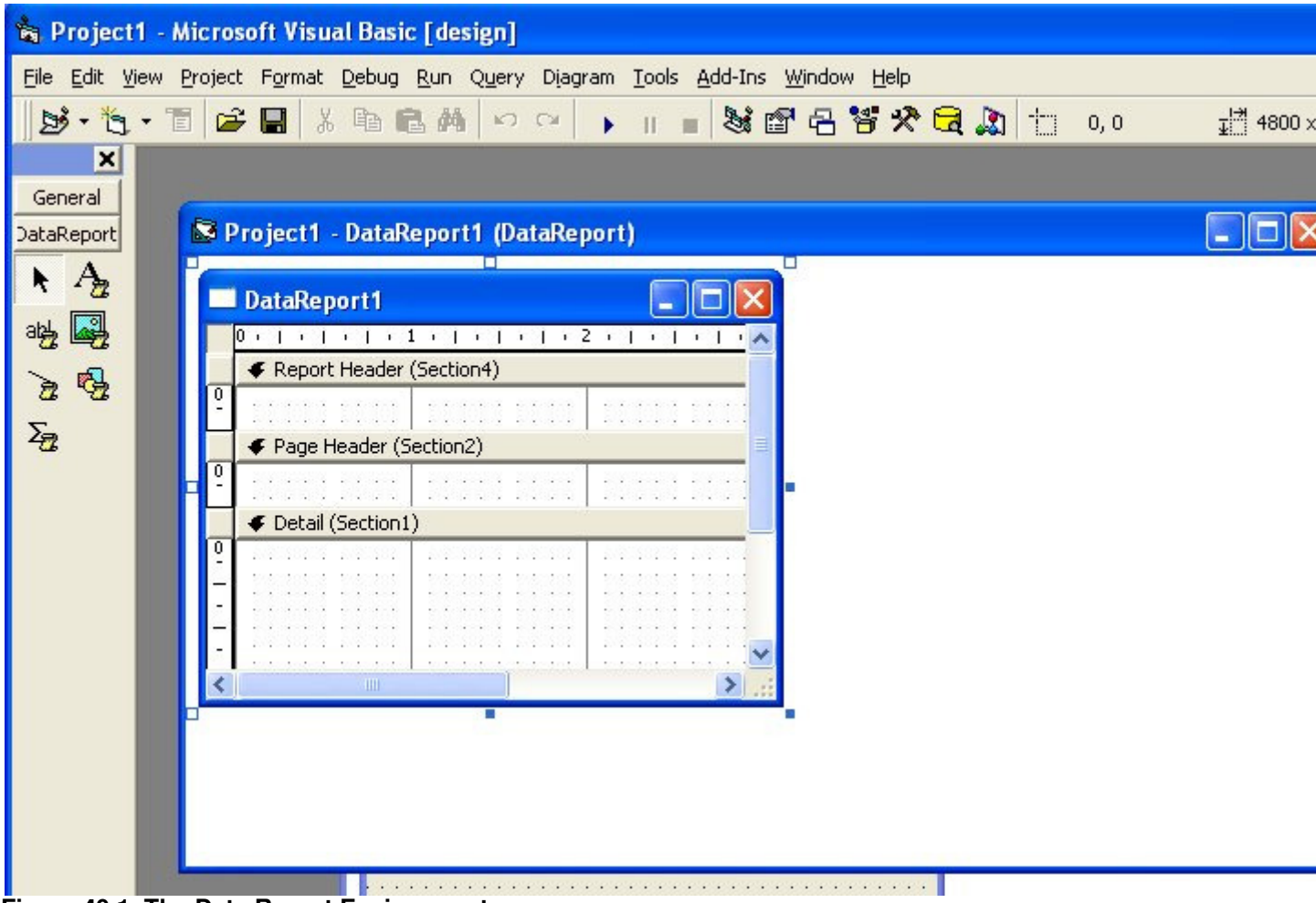


Figure 40.1: The Data Report Environment

Step 2: Connecting the report to database using Data Environment Designer

Click the Project menu, then select Data Environment. from the drop-down menu. The default data environment will appear, as shown in figure 40.2

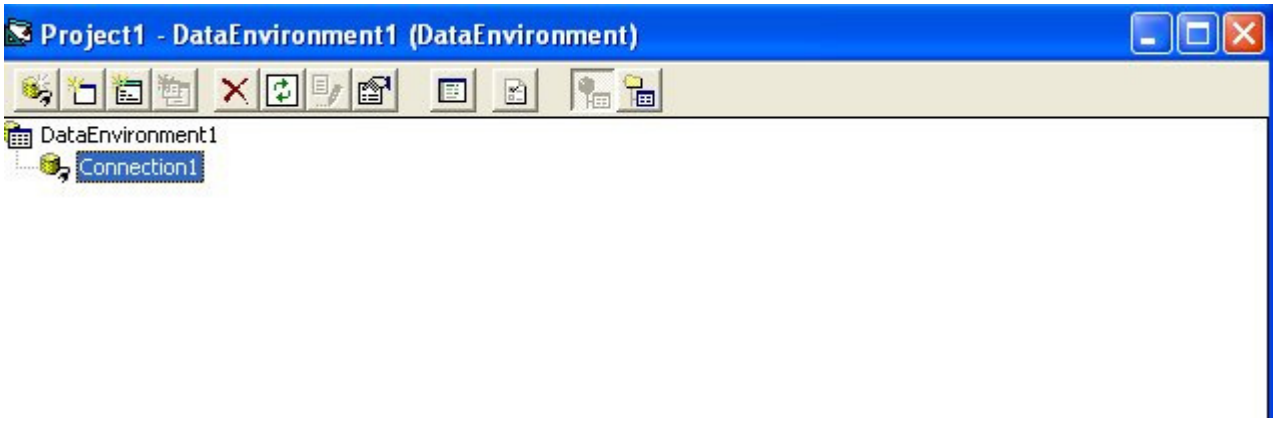


Figure 40.2: Data Environment

Now, to connect to the database, right-click connection1 and select Microsoft Jet 3.51 OLE DB Provider (as we are using MS Access database) from the Data Link Properties dialog (as shown in Figure 40.3), then click next.

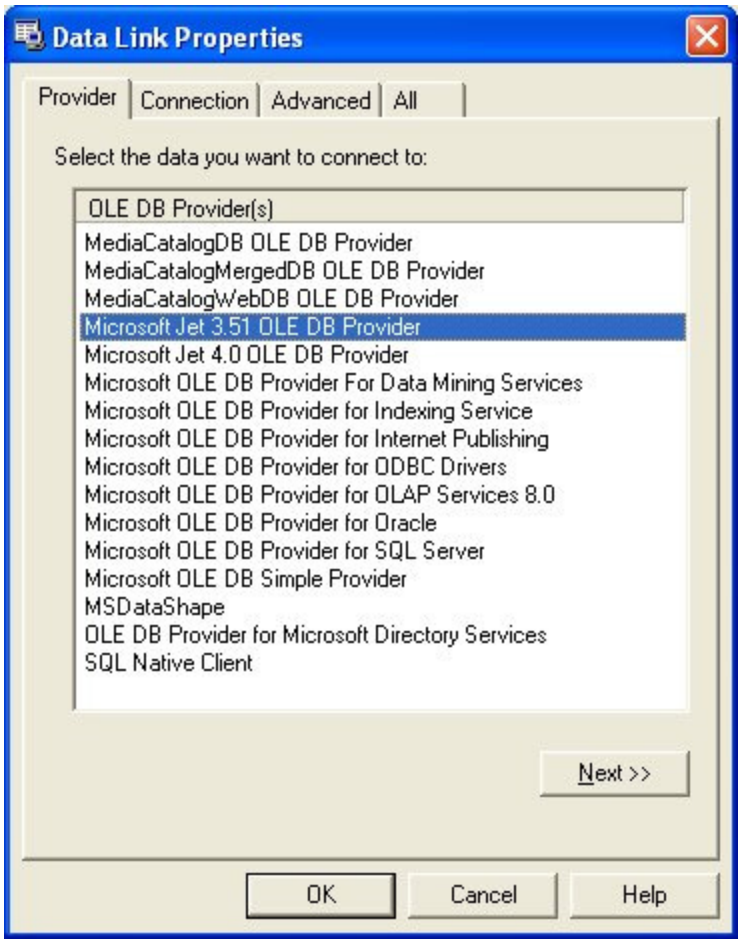


Figure 40.3

Now, you need to connect to the database by selecting a database file from your hard disk. For demonstration purpose, we will use the database BIBLIO.MDB that comes with Visual Basic, as shown in Figure 40.4. The path to this database file is C:\Program Files\Microsoft Visual Studio\VB98\BIBLIO.MDB. This path varies from computers to computers, depending on where you install the file. After selecting the file, you need to test the connection by clicking the Test Connection button at the right bottom of the Data Link Properties dialog. If the connection is successful, a message that says 'Test Connection Succeeded' will appear. Click the OK button on the message box to return to the data environment. Now you can rename connection1 to any name you like by right-clicking it. For example, you can change it to MyConnection. You may also change the name of DataEnvironment1 to MyDataEnvironment using the Properties window.

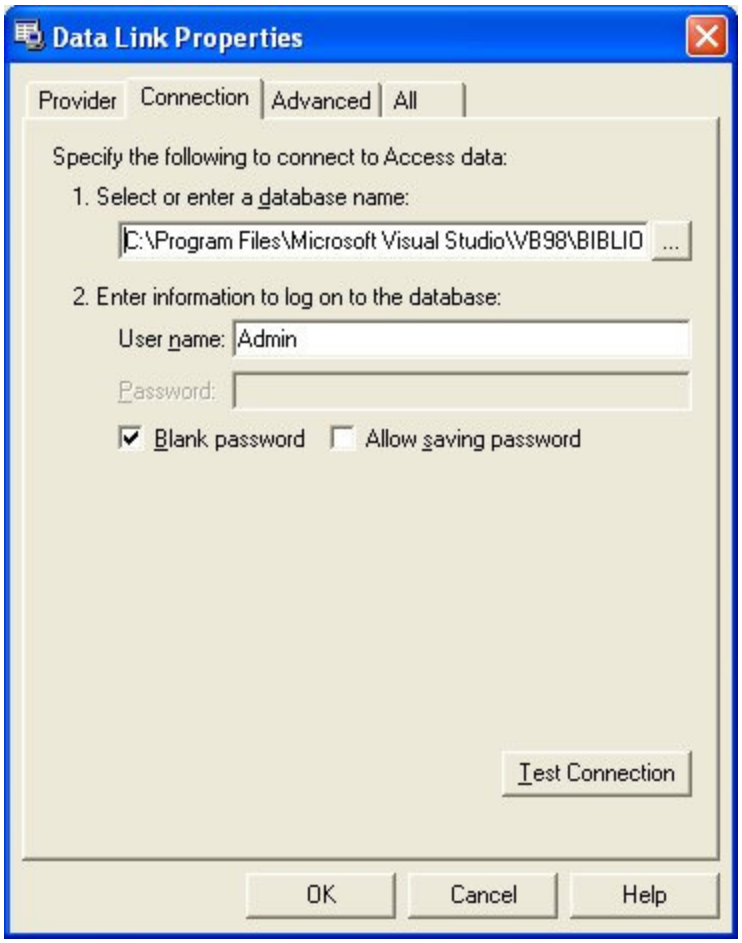


Figure 40.4

Step 3: Retrieving Information from the Database

In order to use the database in your report, you need to create query to retrieve the information from the database. Here , we will use SQL command to create the query. First of all, right click on MyConnection to add a command to the data environment. The default command is Command1, you can rename it as MyCommand, as shown in Figure 40.5.

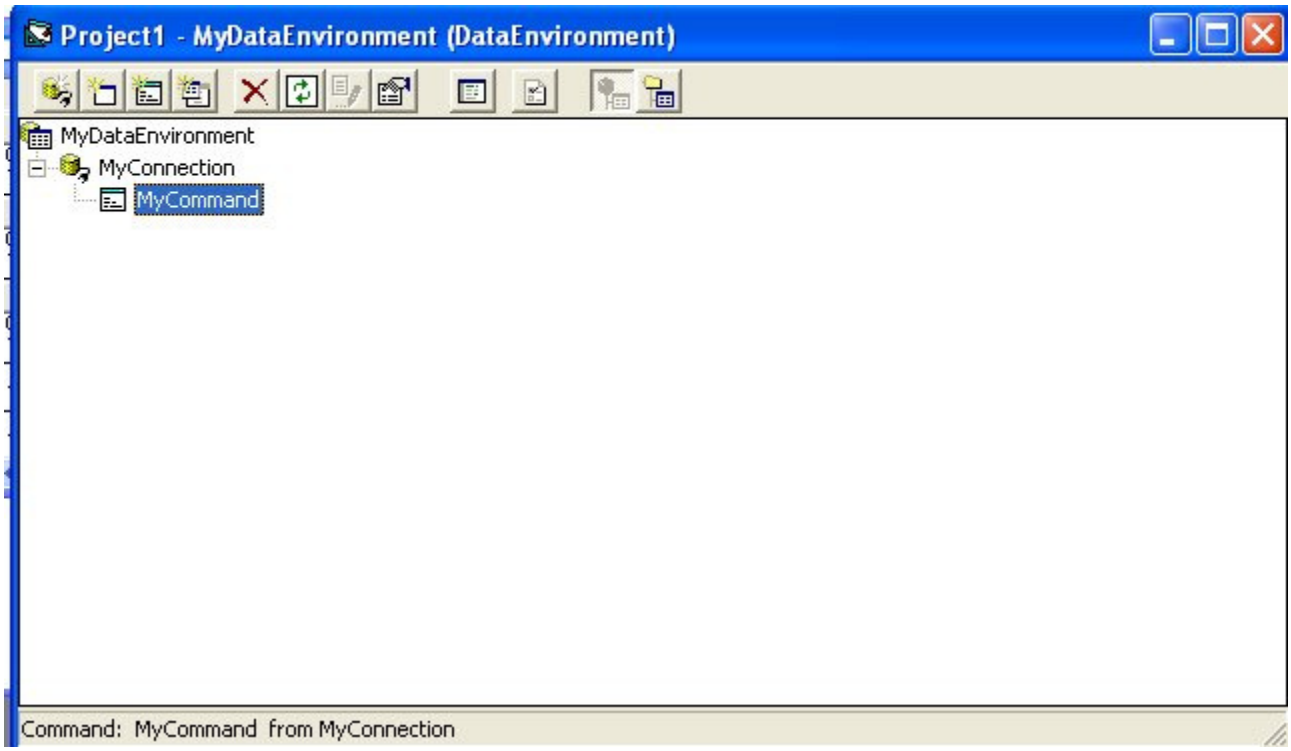


Figure 40.5: MyCommand

In order to use SQL command, right-click MyCommand and you can see its properties dialog. At the General tab, select SQL statement and key in the following SQL statement:

SELECT Au_ID, Author
FROM Authors ORDER BY Author

This command is to select all the fields from the Authors table in the Biblio.Mdb database. The command ORDER BY Author is to arrange the list in ascending order according to the Authors' Names.

Now, you need to customize a few properties of your data report so that it can connect to the database. The first property to set is the DataSource, set it to MyDataEnvironment. Next, you need to set the DataMember property to MyCommand,as shown in Figure 40.6

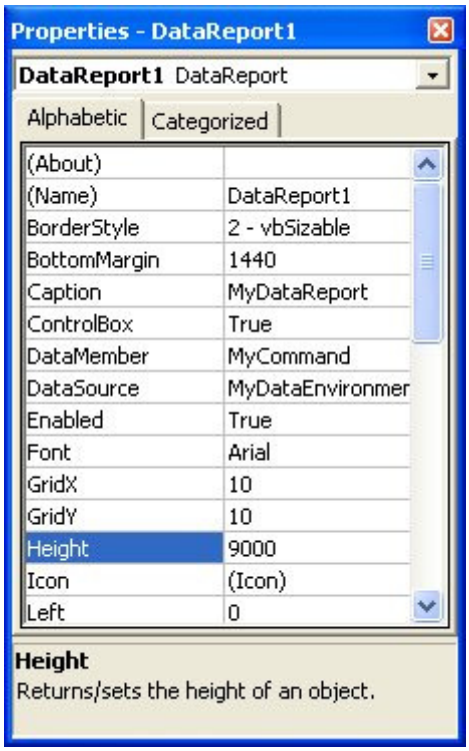


Figure 40.6: Properties of

To add data to your report, you need to drag the fields from MyCommand in MyDataEnvironment into MyDataReport, as shown in Figure 40.7.Visual Basic 6 will automatically draw a RptTextBox, along with a RptLabel control for each field on the report. You can customize the look of the labels as well as the TextBoxes from the properties window of MyDataReport.

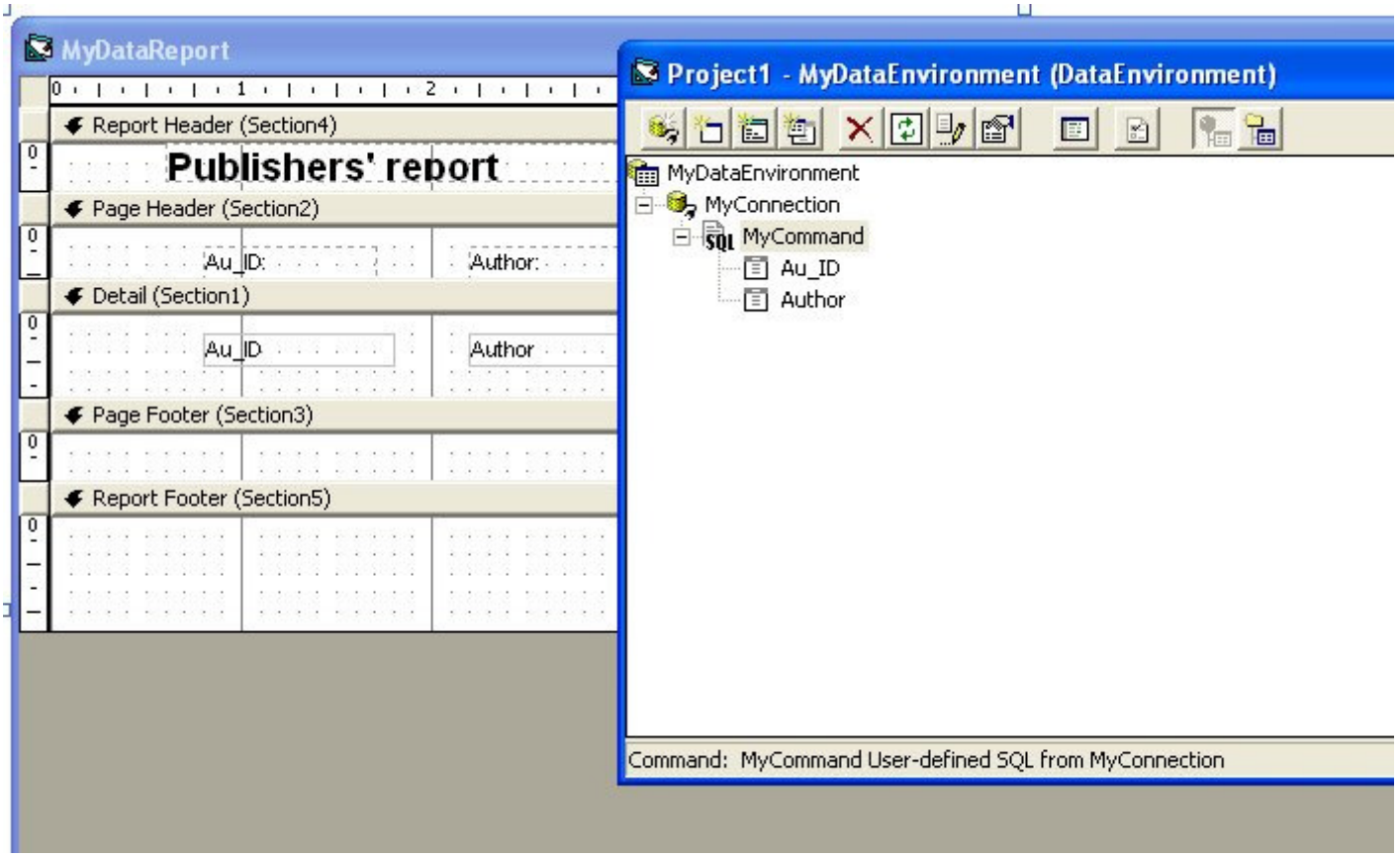


Figure 40.7

The Final step is to set MydataReport as the Startup form from the Project menu, then run the program. You will see your report as shown in Figure 40.8. You can print out your report.

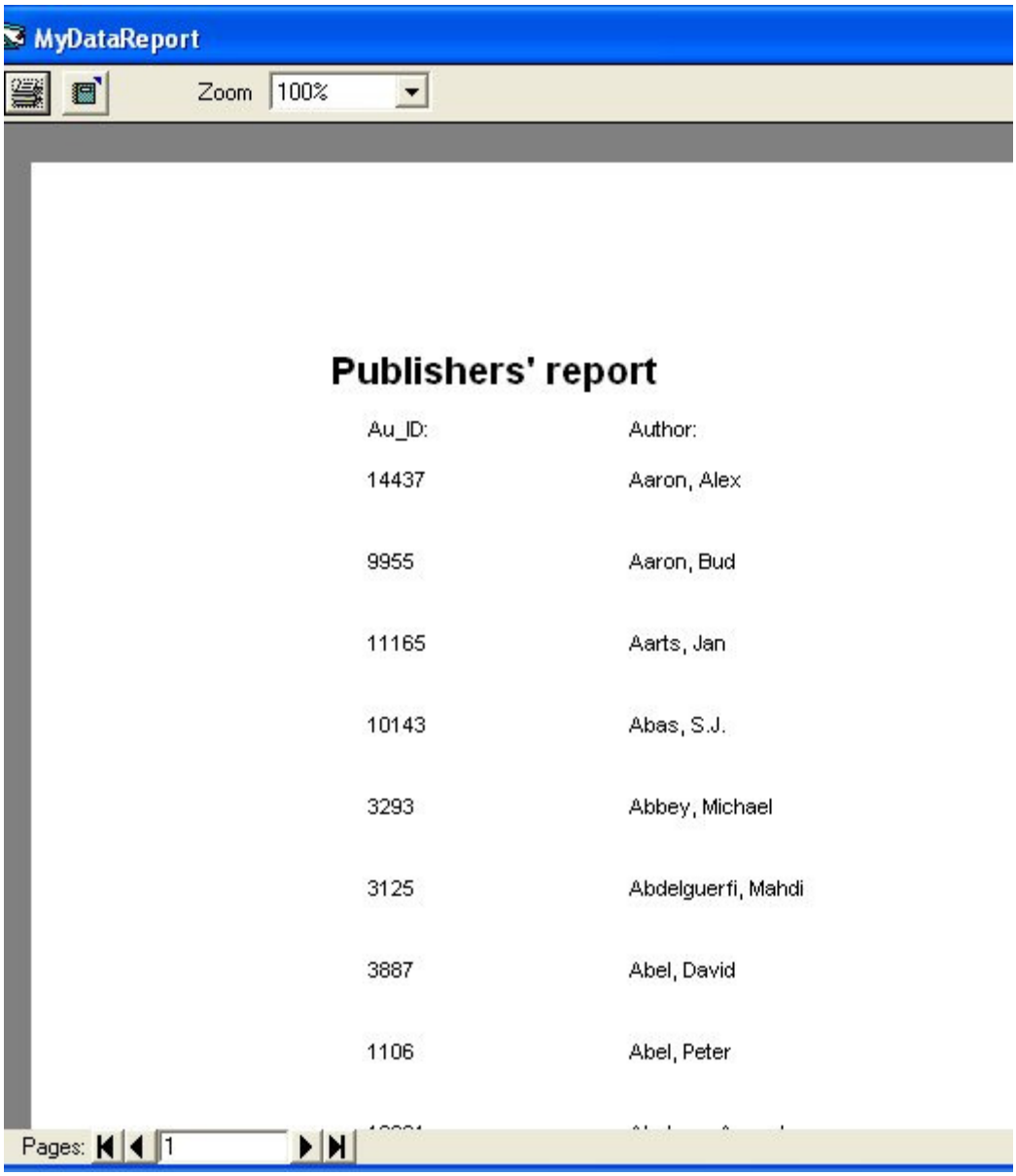
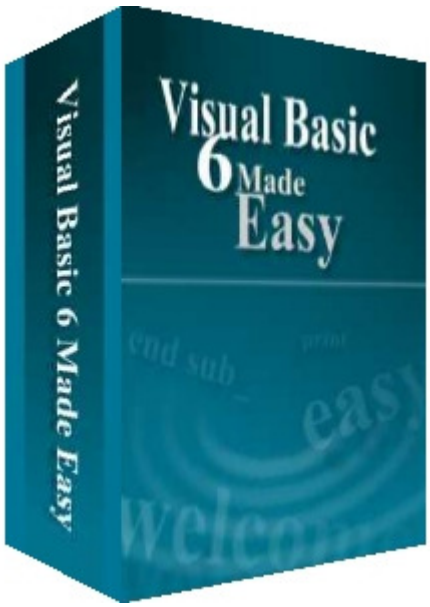


Figure 40.8: The Final Report.

Congratulation! You have finish reading all the 39 lessons, and now you can consider yourself a VB programmer. You should consider buying the TEXTBOOK for this tutorial for easy referencing in the future. Buy this book by clicking the picture below:


Visual Basic 6 Made Easy
Visual Basic 6 Made Easy
end sub
print
easy
welcome