

# **VISUAL PROGRAMMING**

## **The concept of Visual Programming**

Visual Programming: Is a sort of programming language that allows users to illustrate processes using visual elements.

Visual programming is an important tool in modern computing, as it helps people quickly and easily visualize complex computer processes. Visual programming allows users to create visual diagrams and models of their programs.

## **INTRODUCTION TO VISUAL BASIC**

**Visual Basic (VB):** Is an event-driven programming language designed by Microsoft that provide a graphical user interface (GUI) which allows programmer to modify code by simply dragging and dropping objects and defining their behavior and appearance.

### **Visual Basic features and characteristics**

VB is a GUI-based development tool that offers a faster RAD than most other programming languages. VB also features [syntax](#) that is more straightforward than other languages, a visual environment that is easy to understand and high database connectivity.

The visual environment is characterized by a drag-and-drop feature which allows programmers to build a user interface that is easy to use, even for developers with minimum experience.

## **CUSTOMIZE FORM**

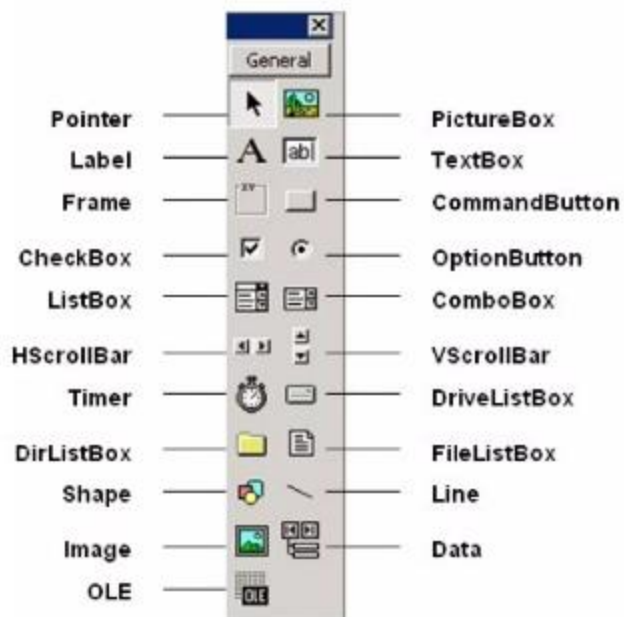
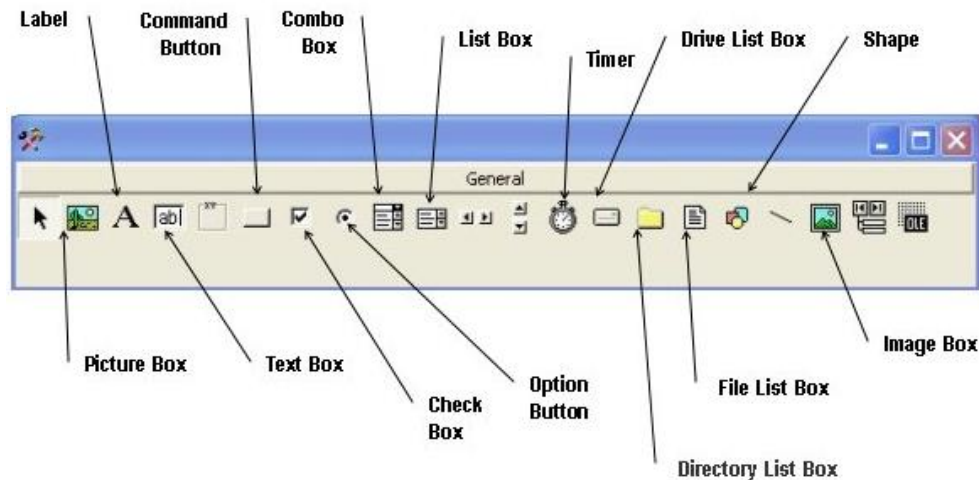
**Visual Basic Form** is the container for all the controls that make up the user interface. The forms are windows which holds control like button, checkbox etc which make user application. The large area of form is called client area for working with different controls.

The Show method both creates and displays the message box. Visual Basic

Unlike the Show method which cares of both loading and displaying the Form, the load statement doesn't show the Form. You have to call the Form's Show method to display it on the desktop.

Handling some of the common Controls

below is the VB6 toolbox that shows the basic controls.



## CONTROL STRUCTURES

### The Control Properties

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

### 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.

## The TextBox

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

## The 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

### Syntax

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

The following example illustrates a simple program that processes the input from the user.

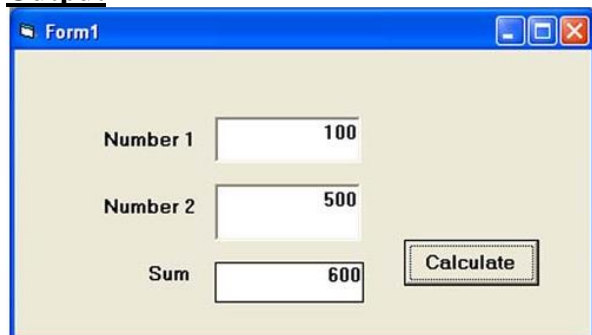
### Example

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

### Source Code

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

### Output



The screenshot shows a Windows application window titled "Form1". Inside the window, there is a light beige background. On the left side, there are three labels: "Number 1", "Number 2", and "Sum". To the right of each label is a text box. The text boxes contain the values "100", "500", and "600" respectively. To the right of the "Sum" text box is a button labeled "Calculate".

## The PictureBox

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:\VBprogram\Images\grape.gif")
```

```
Private Sub cmd_LoadPic_Click()  
MyPicture.Picture = LoadPicture("C:\Users\admin.DESKTOP-  
G1G4HEK\Documents\My Websites\vbtutor\vb6\images\uranus.jpg")  
End Sub
```

## The Image Control

The Image Control is another control that handles images and pictures. It functions almost identically to the pictureBox. 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 PictureBox. 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.

```
Private Sub cmd_LoadImg_Click()  
Image1.Picture=LoadPicture ("C:\VBprogram\Images\grape.gif")  
End Sub
```

## The ListBox

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

## **Example**

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

## **Output**



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 third item has a ListIndex 2 and so on

## **The ComboBox**

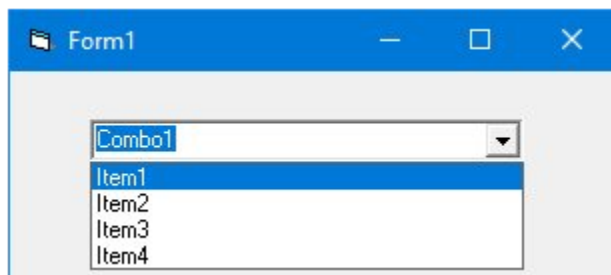
The function of the Combo Box is also to present a list of items where the user can click and select the items from the list. However, the user needs to click on the small arrowhead on the right of the combo box to see the items which are presented in a drop-down list. In order to add items to the list, 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**

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

### *The Output*



The Visual Basic 6 Integrated Programming Environment is shown in Figure 1.2. It consists of the toolbox, the form, the project explorer and the properties window.

## **The CheckBox**

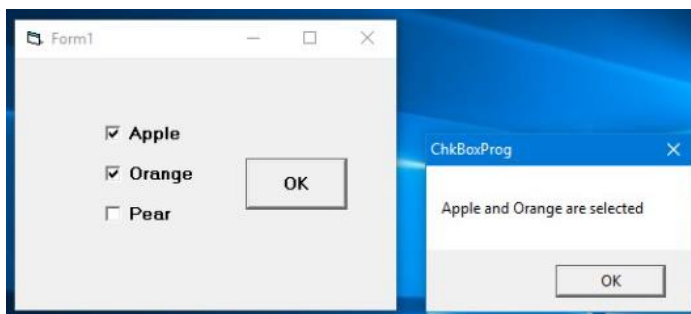
The Check Box control lets the user selects or unselects an option. When the Check Box is checked, its value is set to 1 and when it is unchecked, the value is set to 0. You can include the statements `Check1.Value=1` to mark the Check Box and `Check1.Value=0` to unmark the Check Box, as well as use them to initiate certain actions.

For example, the program below shows which items are selected in a message box.

## Example

```
Private Sub Cmd_OK_Click()  
    If Check1.Value = 1 And Check2.Value = 0 And Check3.Value = 0  
Then  
        MsgBox "Apple is selected"  
    ElseIf Check2.Value = 1 And Check1.Value = 0 And Check3.Value  
= 0 Then  
        MsgBox "Orange is selected"  
    ElseIf Check3.Value = 1 And Check1.Value = 0 And  
Check2.Value = 0 Then  
        MsgBox "Orange is selected"  
    ElseIf Check2.Value = 1 And Check1.Value = 1 And Check3.Value  
= 0 Then  
        MsgBox "Apple and Orange are selected"  
    ElseIf Check3.Value = 1 And Check1.Value = 1 And Check2.Value  
= 0 Then  
        MsgBox "Apple and Pear are selected"  
    ElseIf Check2.Value = 1 And Check3.Value = 1 And Check1.Value  
= 0 Then  
        MsgBox "Orange and Pear are selected"  
    Else  
        MsgBox "All are selected"  
    End If  
End Sub
```

### *The Output*



## The OptionButton

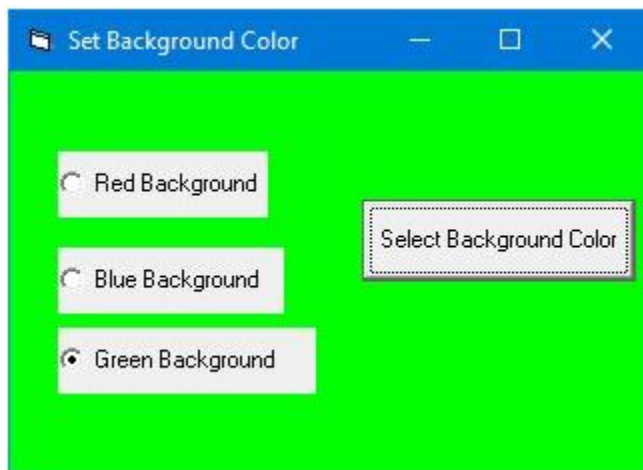
The OptionButton control also lets the user select one of the choices. However, two or more Option buttons must work together because as one of the option buttons is selected, the other Option button will be unselected. In fact, only one Option Box can be selected at one time. When an option box is selected,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

## Example

In this example, we want to change the background color of the form according to the selected option. We insert three option buttons and change their captions to "Red Background", "Blue Background" and "Green Background" respectively. Next, insert a command button and change its name to cmd\_SetColor and its caption to "Set Background Color". Now, click on the command button and enter the following code in the code window:

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

Run the program, select an option and click the "Set Background Color" produces the output, as shown in Figure 3.10.



## The Shape Control

In the following example, the shape control is placed in the form together with six OptionButtons. To determine the shape of the shape control, we use the shape property. The property 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 circle, a rounded rectangle and a rounded square respectively.

## Example

In this example, we insert six option buttons. It is better to make the option buttons into a control array as they perform similar action, i.e to change shape. In order to create a control array, click on the first option button, rename it as MyOption. Next, click on the option button and select copy then paste. After clicking the paste button, a popup dialog will appear asking you whether you wish to create a control array, select yes. The control array can be accessed via its index value, MyOption(Index). In addition, we also insert a shape control.

Now, enter the code in the code window. We use the If Then Else program structure to determine which option button is selected by the user.

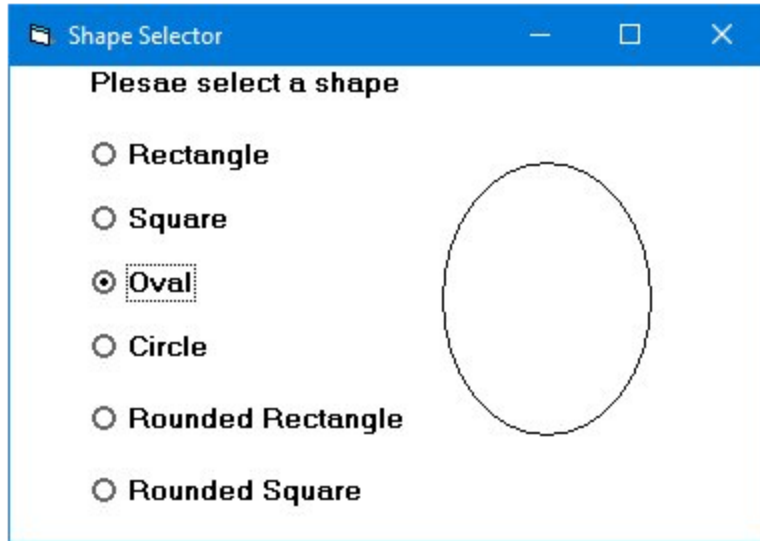
```
Private Sub MyOption_Click(Index As Integer)
If Index = 0 Then
MyShape.Shape = 0
ElseIf Index = 1 Then
MyShape.Shape = 1
ElseIf Index = 2 Then
MyShape.Shape = 2
ElseIf Index = 3 Then
MyShape.Shape = 3
ElseIf Index = 4 Then
MyShape.Shape = 4
ElseIf Index = 5 Then
MyShape.Shape = 5

End If

End Sub
```

Run the program and you can change the shape of the shape control by clicking one of the option buttons. The output shown below





## **The DriveListBox**

The DriveListBox 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.

## **The DirListBox**

The DirListBox means the Directory List Box. It is for displaying a 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

## **Object Linking and Embedding (OLE)**

The OLE control is used to incorporate data into a Visual Basic application either by linking or embedding.

## **Writing the Code**

The source code window: Is where user type the code which VB execute. It has two parts

- i. The control box
- ii. Event box

The heading of the window indicates which event the code is associated. It is a window where user write most of the code.

# **Creating Your First Application**

## **Steps used in creating a VB application**

When you write a Visual Basic application, you follow a three-step process for planning the project and then repeat the process for creating the project.

The three steps involve

- i. Setting up /Design the user interface
- ii. Defining/Set the properties of the controls (object)
- iii. Creating the code / Write the events' procedures

## **Opening, Saving and Running**

### **Opening Project**

Start-----→Program-----→Microsoft Visual Basic 6

### **Saving a Project**

After creating a project, user need to save it. For this click File menu and select save project. Then system ask:-

1. Provide a name for Form File (.frm)
2. After that provide a name for Project file (.vbp)

### **Running a Project**

After saving the project, the next stage is to execute the program. Following are different ways to execute a project

1. Press F5 Key
2. On Menu Bar Click Run-→Start
3. On VB Toolbar, Click Run Icon

## **SUB AND FUNCTION PROCEDURE**

In this lesson, we shall learn some basic techniques in writing the Visual Basic program code. First of all, we should understand that each control or object in VB are able to run numerous kinds of events.

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. Among the events are loading a form, clicking on 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 code for an event procedure, you need to double-click an object to enter the VB code window. For example, if you want to write code for the event of clicking a command button, you double-click the command button and enter the codes in the event procedure that appears in the code window.

The structure of an event procedure is as follows:

```
Private Sub Command1_Click
```

```
VB Statements
```

```
End Sub
```

You enter the codes in the space between **Private Sub Command1\_Click..... End Sub**.

The keyword **Sub** actually stands for a sub procedure that made up a part of all the procedures in a program or a module.

The program code is made up of a number of VB statements that set certain properties or trigger some actions.

The syntax of the Visual Basic's program code is almost like the normal English language, though not exactly the same, so it is fairly easy to learn.

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

**Object.Property**

**Where:**

**Object** and **Property** are separated by a period (or dot).

For example, the statement

**Form1.Show** means to show the form with the name Form1,

**Label1.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 1

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

### Example 2

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

### Example 3

```
Private Sub Command1_click()  
Picture1.Show=true  
Timer1.Enabled=True  
Label1.Caption="Start Counting"  
End sub
```

### Example 4 A Counter

This is a counter which start counting after the user click on a command button. In this program, we insert a label, two command buttons and a Timer control. The label acts as a counter, one of the command buttons is to start the counter and the other one is to stop the counter. The Timer control is a control that is only used by the developer, it is invisible during runtime and it does not allow the user to interact with it.

The Timer's Interval property determine how frequent the timer changes. A value of 1 is 1 milliseconds which means a value of 1000 represents 1 second. In this example, we set the interval to 100, which represents 0.1 second interval. In addition, the Timer's Enabled property is set to false at design time as we do not want the program to start counting immediately, the program only start counting after the the user clicks on te "Start Counting" button. You can also reset the counter using another command button.

## *The Code*

```
Dim n As Integer
Private Sub cmd_StartCount_Click()
Timer1.Enabled = True
End Sub

Private Sub cmd_Stop_Click()
Timer1.Enabled = False
End Sub

Private Sub Command1_Click()
Lbl_Display.Caption = 0
End Sub

Private Sub Timer1_Timer()
n = n + 1
Lbl_Display.Caption = n
End Sub
```

\* We declare the variable `n` in the general area. After the `Timer1` is enabled, it will add 1 to the preceding number using `n=n+1` after every interval until the user click on the "Stop Counting" button.

## *The Output*



## Example 5 Click and Double Click

This program display a message whether the label is being click once or click twice. In this program, insert a label and rename it as MyLabel and change its caption to "CLICK ME". Next, key in the following codes:

```
Private Sub MyLabel_Click()  
    MyLabel.Caption = "You Click Me Once"  
End Sub  
  
Private Sub MyLabel_DblClick()  
    MyLabel.Caption = "You Click Me Twice!"  
End Sub
```

### *The Output*



Running the program and click the label once, the "CLICK ME" caption will change to "You Click Me Once". If you click the label twice, the "CLICK ME" caption will change to "You Click Me Twice!".

In Visual Basic, most of the syntaxes resemble the English language. Among the syntaxes 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.

Program code that involves calculations is fairly easy to write, just like what 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, 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 to a numerical value and **Str\$** is to convert numerical to a string (text). While the function **Str\$** is as important as VB can display a numeric value as string implicitly, failure to use **Val** will result in the 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 example 4.5 will give you the correct result, i.e., 15