Training Manual FOR Microsoft® Visual Basic® Programming

Compiled by **Technology Ed**

April 2012

CHAPTER-2: BUILDING AN APPLICATION

This section describes in detail the Visual Basic approach on building an application.

2.1 Setting Object Properties

Visual Basic programming objects are loaded with properties. A property is a named attribute of a programming object. Properties define the characteristics of an object, such as Size, Color, and so on or sometimes the way in which it behaves. For example, a **TextBox** accepts properties such as **Enabled**, **Font**, **MultiLine**, **Text**, **Visible**, **Width**, and so on.

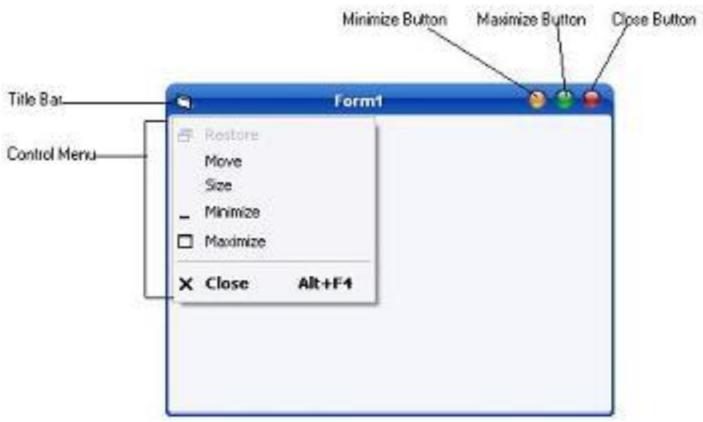
- Enables property allows the TextBox to be enabled or disabled at run time depending on the condition set to True or False
- Font property sets a particular font in the TextBox
- MultiLine property allows the TextBox to accept and display multiple lines at run time
- Text property of the TextBox control sets a particular text in the control
- Visible property is used to hide the object at run time
- Width property sets the TextBox to the desired width at design time

The properties listed above are design-time properties that can be set at the design time by selecting the Properties Window. However certain properties cannot be set at design time. For example, the CurrentX and CurrentY properties of a Form cannot be set at the design time.

2.2 Forms

The main characteristic of a Form is the title bar on which the Form's caption is displayed. On the left end of the title bar is the Control Menu icon. Clicking this icon opens the Control Menu. Maximize, Minimize and Close buttons can be found on the right side of the Form. Clicking on these buttons performs the associated function.

The following figure illustrates the appearance of a Form.

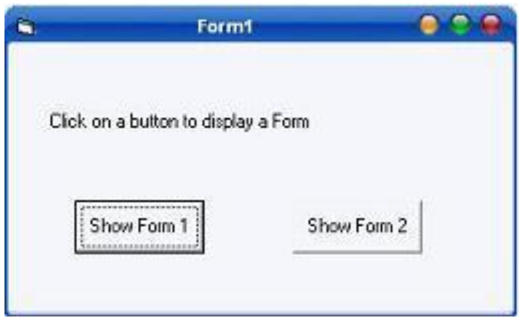


The control menu contains the following commands:

- **Restore**: Restores a maximized Form to the size it was before it was maximized; available only if the Form has been maximized.
- **Move**: Lets the user moves the Form around with the mouse
- **Size**: Lets the user resizes the control with the mouse
- **Minimize**: Minimizes the Form
- **Maximize**: Maximizes the Form
- **Close**: Closes the Form

Setting the Start-Up Form

A typical application has more than a single Form. When an application runs the main Form is loaded. By setting the Project properties you can control which Form is to be displayed in the Start-Up of the application. Following figure illustrates the Project property window.



By default, Visual Basic suggests the name of the first Form created when the project started.

Loading and Unloading Forms

In order to load and unload the forms, Load and Unload statements are used. The Load statement has the following syntax:

```
Load FormName
```

And the Unload statement has the following syntax:

```
Unload FormName
```

The `FormName` variable is the name of the Form to be loaded or unloaded. 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.

Showing and Hiding Forms

Show method is used to Show a Form. If the Form is loaded but invisible, the Show method is used to bring the Form on Top every other window. If the Form is not loaded, the Show method loads it and then displays it. The following is the syntax of the Show method of the Form:

```
FormName.Show mode
```

The `FormName` variable is the Form's name, and the optional argument mode determines whether the Form will be Modal or not. It can have one of the following syntax:

```
* 0-Modeless (default)

* 1-Modal
```

Modeless Forms are the normal Forms. Modeless Forms interact with the user and the user allowed switching to any other Form of the application. If you do not specify the optional mode argument, by default the mode is set to modeless. The Modal Forms takes the total control of the application where user cannot switch to any other Forms in the application unless the Form is closed. A modal Form, therefore, must have a Close button or some means to close the Form in order to return to the Form where the Modal Form was loaded.

Hiding Forms

The Hide method is used to hide a Form. The following is the syntax of the Hide Method.

```
FormName.Hide
```

To hide a Form from within its own code, the following code can be used.

```
Me.Hide
```

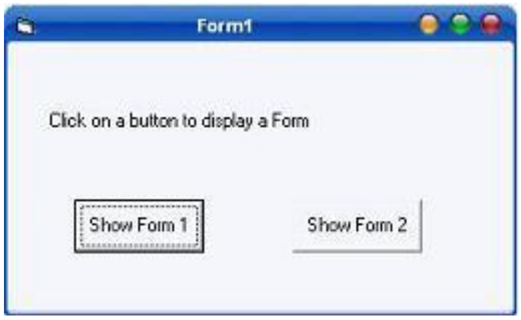
Note: It is highly recommended for you to understand that the Forms that are hidden are not unloaded; they remains in the memory and can be displayed instantly with the Show Method. When a Form is hidden, you can still access its properties and code. For instance, you can change the settings of its Control Properties or call any Public functions in the Form.

The following is an example illustrates the Show method and Mode statement:

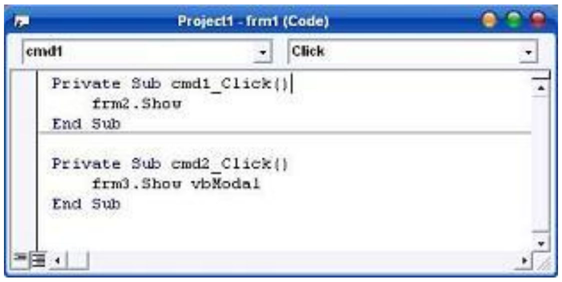
* Open a new Project and save the Project

Design the application as shown below

Object	**Property**	**Setting**
Form	Caption Name	Form1 frm1
Form	Caption Name	Form2 frm2
Form	Caption Name	Form3 frm3
Label	Caption Name	Click on a button to display a Form Label1



The following code is typed in the Click event of the command buttons:

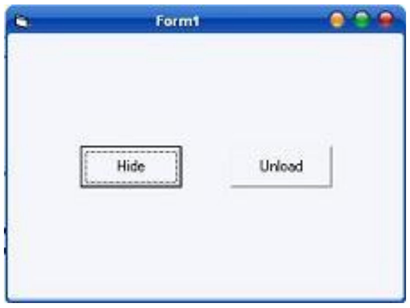


```
Private Sub cmd1_Click()
    frm2.Show
End Sub

Private Sub cmd2_Click()
    frm3.Show vbModal
End Sub
```

Once you complete typing the code, as appropriate, you can run the application. Clicking on the buttons will display the Forms respectively. However, you can see that in the cmd2_Click( ) event additionally VbModal argument has been added. You can see the difference after you display the forms by clicking on the command buttons. You can notice that you cannot switch to any other Forms in the application unless you close the Form3. In order to find the difference between Unload and Hide method, you can refer the following example:

Open a new project and save the project. Draw two buttons on the form and name those as shown above.



In the click event of the Hide button following code is entered.

```
Me.Hide
```

In the click event of the Unload button following code is entered.

```
Unload Me
```

Save the project and run the application. Once you click on Hide button you can note that the Form is invisible but the application is still running. But when you click on Unload button you can see that the application is terminated.

2.3 Introduction to Controls

This section describes the Visual Basic controls and the ways of creating and implementing the controls. It also helps us to understand the concept of Control Arrays. Controls are used to receive user input and display output and have its own set of properties, methods and events.

2.3.1 Creating and Using Controls

A control is an object that can be drawn on a Form object to enable or enhance user interaction with an application. Controls have properties that define aspects of their appearance, such as position, size and color, and aspects of their behavior, such as their response to the user input. They can respond to events initiated by the user or set off by the system. For instance, a code could be written in a CommandButton control's click event procedure that would load a file or display a result.

In addition to properties and events, methods can be used to manipulate controls from code. For instance, the move method can be used with some controls to modify their location and size.

Most of the controls provide choices to users that can be in the form of OptionButton or CheckBox controls, ListBox entries or ScrollBars to select a value.

2.3.2 Classification of Controls

Visual Basic controls are broadly classified as standard controls, ActiveX controls and insertable objects. Standard controls, such as CommandButton, Label and Frame controls are contained inside .EXE file and are always included in the ToolBox which cannot be removed. ActiveX controls exist as separate files with either .VBX or .OCX extension. They include the following specialized controls:

- MSChart control
- The Communications control
- The Animation control
- The ListView control
- An ImageList control
- The Multimedia control
- The Internet Transfer control
- The WinSock control
- The TreeView control
- The SysInfo control

- The Picture Clip control

Some of these objects support OLE Automation, which allow programming another application's object from within Visual Basic application.

Note: It is highly recommended to stress that knowing how and when to set the objects' properties is very important. This is because it can help you to write a good program or you may fail to write a good program. Hence it is recommended to spend a lot of time playing with the objects' properties.

The following are the important pointers about setting up the properties:

You must 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 must have no problem in manipulating the buttons.

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.

TabIndex Property of Controls

Visual Basic uses the TabIndex property to determine the control that would receive the focus next when a tab key is pressed. Every time a tab key is pressed, Visual Basic looks at the value of the TabIndex for the control that has focus and then it scans through the controls searching for the next highest TabIndex number. When there are no more controls with higher TabIndex value, Visual Basic starts all over again with 0 and looks for the first control with TabIndex of 0 or higher that can accept keyboard input.

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

2.4 Event Driven Programming (Object Based)

Visual Basic programs are built around events. Events are various things that can happen in a program. In procedural languages, an application is written is executed by checking for the program logically through the program statements, one after another. For a temporary phase, the control may be transferred to some other point in a program. While in an event driven application, the program statements are executed only when a particular event calls a specific part of the code that is assigned to the event.

For example, you can consider a TextBox control and a few of its associated events to understand the concept of event driven programming. The TextBox control supports various events, such as Change, Click, MouseMove and many more that will be listed in the Properties dropdown list in the code window for the TextBox control.

- The code entered in the Change event fires when there is a change in the contents of the TextBox
- The Click event fires when the TextBox control is clicked.
- The MouseMove event fires when the mouse is moved over the TextBox

As explained above, several events are associated with different controls and forms, some of the events being common to most of them and few being specific to each control.