Chapter Two: Controls and Properties

2.1 Controls Provide the Interface

The controls you select for your application's form are important because the controls (also called tools) provide the application interface for your users. Users interact with your application by clicking the controls and entering text in the controls. Placing and sizing controls are perhaps the two most important tasks you can master at this point.

1- Placing the controls

There are two methods for placing controls on the form:

A. Double-click any control on the Toolbox window to place that control on the Form window.

If a control appears in the center of the form already, the new control will overwrite the existing control. You can drag the new control to a different location, however. The eight sizing handles (the small boxes that appear around a selected control) indicate that the control is selected. If several controls appear on the Form window, the selected controls will display their sizing handles. (Typically, only one control will be selected at any one time but you can select multiple controls by holding the Ctrl key and clicking several controls).

B. If you click a toolbox control once, the toolbox highlights the control. If you then move the mouse cursor to the Form window, the mouse cursor turns into a crosshair indicating that you can place the selected control anywhere on the form. Whereas a control appears in the center of the Form window automatically as soon as you double-click the control, a selected control appears only when you click and drag your mouse crosshair on the Form window. The final control appears when you release the mouse.

The advantage of using the second approach to placing controls over the first approach is that you don't have to move and resize the control after you've placed it. While in the second approach you can place the control exactly where you want it and at the size you want it when you drag the control onto the form.

2-Sizing and Moving Controls

The eight sizing handles are the key to resizing the control. You can drag any of the eight sizing handles in any direction to increase or decrease the control's size.
You can move a selected control to any area of the Form window by dragging the control with your mouse. Once you click to select a control, click the control and hold down the mouse button to drag the control to another part of the Form window.
Sometimes you may want to drag several controls to a new location as a group. For example, perhaps you've placed a set of command buttons at the bottom of a form and after adjusting the Form window's size, you determine that you need to move the buttons down some. Although you can move the command buttons one at a time, you can more quickly select all the command buttons and move them as a group.

Note:-

If you select multiple controls before changing a control property, all controls in the selected range will take on that new property value. You can only change the common properties that appear in all of the selected controls.

2.2 Setting Properties

You can set some properties only at design time (such as a control's Name property), you can set some properties both at design time and at runtime inside event procedures and other module code (such as a caption), and you can set some properties only at runtime from within the program (such as a list box's entries). All of a control's properties that appear in the Properties window are settable at design time, and some of those you can set at runtime as well.

1. Setting Properties of Objects at Design Time

As you add controls to the Form window, the Properties window updates to show the properties for the currently selected control. The selected control is usually the control you last placed on the form. Each form and control has properties assigned to it by default when you start up a new project. There are two ways to display the properties of an object. The first way is to click on the object (form or control) in the form window. Then, click on the Properties Window or the Properties Window button in the tool bar. The second way is to first click on the Properties Window. Then, select the object from the Object box in the Properties Window.

Note:-

Visual Basic programmers often use the generic term object to refer to controls, forms, menus, and various other items on the screen and in the code.

Note:-

Each kind of control supports the same set of properties. Therefore, every command button you place on the form supports the same properties (and events as well) as every other command button, but option buttons and text boxes support different sets of properties than command buttons.

After you place and size a control, the first property you should modify is the Name property. Although Visual Basic assigns default names to controls when you place the controls on the Form window, the default names don't indicate the control's true purpose in your application. In addition, the default names don't contain the three-letter prefix that describes the control you learned.

Table.1. Use these prefix abbreviations before control names.

Prefix	*Control*
cbo	**Combo box**
chk	**Check box**
cmd	**Command button**
dir	**Directory list box**
drv	**Drive list box**
fil	**File list box**
fra	**Frame**
frm	**Form**
grd	**Grid**
hsb	**Horizontal scrollbar**
img	**Image**
lbl	**Label**
lin	**Line**
lst	**List box**
mnu	**Menu**
ole	**OLE client**
opt	**Option button**
pic	**Picture box**
shp	**Shape**
tmr	**Timer**
txt	**Text box**
vsb	**Vertical scrollbar**

A tooltip: - is a pop-up description box that appears when the user resets the mouse pointer over a control.

Some property values you set by typing the values directly in the Properties window. For example, to enter a value for a control's **ToolTipText** property, click once on the Properties window's **ToolTipText** property and type the tooltip text.

Note:-

If you want to change a property value, such as the Name property, you can click the Name property and enter a new name. As you type, the new name replaces the original name. If instead of clicking you double-click the property, visual Basic highlights the property value and lets you edit the existing value by pressing your cursor keys and using Insert and Delete to edit the current property value.

Note:-

As you select a property, read the text that appears at the bottom of the Properties window. The text describes the property and serves as a reminder about what some of the more obscure properties do.

Note:-

Some properties require a selection from a drop-down list box. For example, the **Visible property** can either be **True** or **False.** No other values work for the property, so Visual Basic lets you select from one of those two values when you click the property value to display the down arrow and open the drop-down list box.

Note:-

If an ellipsis **(...)** is displayed when you click the property value, such as the **Font property** when you click the current Font property's value, a dialog box opens when you click the ellipsis. A **Font property** is more than just a style name or size. The **control's Font property** can take on all kinds of values and the Font dialog box that appears from the click of the ellipsis.

2.3 Related Properties

1. Command Buttons

Command buttons appear in almost every window of every Windows application. Command buttons determine when the user wants to do something, such as exit the application or begin printing. In almost every case, you will perform these tasks to add a command button to an application:

Table.1. Common command button properties.

Property	*Description*
BackColor	**Specifies the command button's background color. Click the BackColor's palette down arrow to see a list of colors and click Categorized to see a list of common Windows control colors. Before the command button displays the background color, you must change the Style property from 0-Standard to 1-Graphical.**
Caption	**Holds the text that appears on the command button.**
Enabled	**Determines whether the command button is active. Often, you'll change the Enabled property at runtime with code when a command button is no longer needed and you want**
Font	**Produces a Font dialog box in which you can set the caption's font name, style, and size.**

Height	**Holds the height of the command button in twips.**
Left	**Holds the number of twips from the command button's left edge to the Form window's left edge.**
MousePointer	**Determines the shape of the mouse cursor when the user moves the mouse over the command button.**
Picture	**Holds the name of an icon graphic image that appears on the command button as long as the Style property is set to 1-Graphical.**
Style	**Determines whether the command button appears as a standard Windows command button (if set to 0-Standard) or a command button with a color and possible picture (if set to 1-Graphical).**
ToolTipText	**Holds the text that appears as a tooltip at runtime.**
Top	**Holds the number of twips from the command button's top edge to the Form window's top edge.**
Visible	**Determines whether the command button appears or is hidden from the user. (Invisible controls cannot receive the focus until the running code changes the Visible property to True.)**
Width	**Holds the width of the command button in twips.**

2. Labels

Labels hold the primary text that appears on a form. Often, programmers use labels to place titles around the form and to label text boxes so users know what to type into the text box.

Table 2 Common label properties.

Property	*Description*
Alignment	**Determines whether the label's caption appears left-justified, centered, or right-justified within the label's boundaries.**
AutoSize	**Enlarges the label's size properties, when True, if you assign a caption that is too large to fit in the current label's boundaries at runtime.**

BackColor	**Specifies the label's background color. Click the BackColor's palette down arrow to see a list of colors and click Categorized to see a list of common Windows control colors.**
BackStyle	**Determines whether the background shows through the label or if the label covers up its background text, graphics, and color.**
BorderStyle	**Determines whether a single-line border appears around the label.**
Caption	**Holds the text that appears on the label.**
Enabled	**Determines whether the label is active. Often, you'll change the Enabled property at runtime with code when a label is no longer needed.**
Font	**Produces a Font dialog box in which you can set the caption's font name, style, and size.**
ForeColor	**Holds the color of the label's text.**
Height	**Holds the height of the label's outline in twips.**
Left	**Holds the number of twips from the label's left edge to the Form window's left edge.**
MousePointer	**Determines the shape of the mouse cursor when the user moves the mouse over the label.**
ToolTipText	**Holds the text that appears as a tooltip at runtime.**
Top	**Holds the number of twips from the label's top edge to the Form window's top edge.**
Visible	**Determines whether the label appears or is hidden from the user.**
Width	**Holds the width of the label in twips.**
WordWrap	**Determines whether the label expands to fit whatever text appears in the caption.**

Note:-

1-Set the **AutoSize property** to **False** if you want the label to remain the same size and not resize automatically to fit the Caption property value.

2-The label automatically expands horizontally to hold the entire caption, if set **AutoSize property** to **true**

3-To expand the label downward when needed to hold entire caption in a multiline label. Set **WordWrap** to **True** before you set the **AutoSize property** to True.

If you set **AutoSize** first, the label expands horizontally before you have a chance to set the **WordWrap property**.

3. Text Boxes

Text boxes accept user input. Although several other controls accept user input, text boxes are perhaps the easiest to set up and respond to. In addition, a text box is simple for your users to use.

A text box is used to display information entered at design time, by a user at run-time, or assigned within code. The displayed text may be edited.

Table 3 Common text box properties.

Property	*Description*
Alignment	**Determines whether the text box's text appears left-justified, centered, or right-justified within the text box's boundaries.**
BackColor	**Specifies the text box's background color. Click the BackColor property's palette down arrow to see a list of colors and click Categorized to see a list of common Windows control colors.**
BorderStyle	**Determines whether a single-line border appears around the text box.**
Enabled	**Determines whether the text box is active. Often, you'll change the Enabled property at runtime with code when a text box is no longer needed.**
Font	**Produces a Font dialog box in which you can set the Text property's font name, style, and size.**

ForeColor	**Holds the color of the text box's text.**
Height	**Holds the height of the text box's outline in twips.**
Left	**Holds the number of twips from the text box's left edge to the Form window's left edge.**
MaxLength	**Specifies the number of characters the user can type into the text box.**
MousePointer	**Determines the shape of the mouse cursor when the user moves the mouse over the text box.**
MultiLine	**Let's the text box hold multiple lines of text or sets the text box to hold only a single line of text. Add scrollbars if you wish to put text in a multiline text box so your users can scroll through the text.**
PasswordChar	**Determines the character that appears in the text box when the user enters a password.**
ScrollBars	**Determines whether scrollbars appear on the edges of a multiline text box.**
Text	**Holds the value of the text inside the text box. The Text property changes at runtime as the user types text into the text box. If you set an initial Text property value, that value becomes the default value that appears in the text box when the user first sees the text box.**
ToolTipText	**Holds the text that appears as a tooltip at runtime.**
Top	**Holds the number of twips from the text box's top edge to the Form window's top edge.**
Visible	**Determines whether the text box appears or is hidden from the user.**
Width	**Holds the width of the text box in twips.**

Note:-

1. The Caption property is the most common property that displays text on a control such as a command button and a label. Text Box controls does not support the Caption property. The Text property holds text for Text Box controls.

2. If you are unsure how to use a particular control's property, click the property in the Properties window and press F1 to read the online help.

4.Form Properties

Forms have properties that you can and should set when you create an application. Being the background of your application.

Table 4 Common form properties.

Property	*Description*
BackColor	**Specifies the form's background color. Click the BackColor's palette down arrow to see a list of colors and click Categorized to see a list of common Windows control colors.**
BorderStyle	**Determines how the Form window appears. The BorderStyle property specifies whether the user can resize the form and also determines the kind of form you wish to display.**
Caption	**Displays text on the form's title bar at runtime.**
ControlBox	**Determines whether the form appears with the Control menu icon. The Control menu appears when your application's user clicks the Control menu icon.**
Enabled	**Determines whether the form is active. Often, you'll change the Enabled property at runtime with code when a form is no longer needed. Generally, only multiform applications, such as MDI applications, need to modify a form's Enabled property.**
Font	**Produces a Font dialog box in which you can set the text's font name, style, and size.**
ForeColor	**Holds the color of the form's text.**
Height	**Holds the height of the form's outline in twips.**

Icon	**Describes the icon graphic image displayed on the taskbar when the user minimizes the form.**
Left	**Holds the number of twips from the form's left edge to the screen's left edge.**
MaxButton	**Specifies whether a maximize window button appears on the form.**
MinButton	**Specifies whether a minimize window button appears on the form.**
MousePointer	**Determines the shape of the mouse cursor when the user moves the mouse over the form.**
Moveable	**Specifies whether the user can move the form at runtime.**
Picture	**Determines a graphic image that appears on the form's background at runtime.**
StartUpPosition	**Determines the state (centered or default) of the form at application startup.**
Top	**Holds the number of twips from the form's top edge to the Form window's top edge.**
Visible	**Determines whether the form appears or is hidden from the user.**
Width	**Holds the width of the form in twips.**

The event: - is something that happens, usually but not always due to the user at the keyboard, during a program's operation.

Before talk about setting property at run time we must talk about:-

- **Control Events**

  Every control you place on a form supports one or more events. For example, if you place a text box in the center of the Form window and run the program, you can click the text box, enter text in the text box. If you've written an event procedure for that text box's event, your code's instructions will execute automatically as soon as the event occurs.

❖ **Event Procedures**

Visual Basic makes it easy to locate event procedure code for controls on forms. Double-click any control to see one of its event procedures. For example, if you double-click the command button labeled cmdUp, Visual Basic opens the Code window and places statements of code, see example 1.

Example1:-The cmdUp command buttons Click event procedure

```
Private Sub cmdUp_Click()
  statements
End Sub
```

Event procedure

Wrapper lines: - are the first and last lines of a procedure.

Most event procedures begin with the statement **Private Sub** (first line) and end with **End Sub** (last line). The Private-End block (a block is a section of code that goes together as a single unit) illustrates the first and last lines of the event procedure. The lines between these wrapper lines comprise the **body** of the event procedure.

All controls have unique names, as you saw earlier. All event procedures also have unique names. An event procedure name always takes this form:

Private Sub **ControlName_EventName** (Optional Arguments)

.

.

End Sub

2. Setting Properties at Run Time

You can also set or modify properties while your application is running. To do this, you must write some code in Event procedure. The code format is:

ObjectName.Property = NewValue

Example:- To change the BackColor property of a form name frmStart, we'd type: frmStart.BackColor = BLUE.

```
Private Sub form_Click()
 frmStart.BackColor =vbBLUE
End Sub
```

Example:- Change the caption property of a label1 name lblbold to **welcome** and then change the font of label to **bold**.

```
Private Sub lblbold_Click()
Lblbold.caption=”welcome”
Lblbold . fontbold= true
End sub
```

Example:- **Set the properties of the form and each object.**

Form1:

BorderStyle **1-Fixed Single**

Caption Savings Account

Name frmSavings

Label1:

Caption Monthly Deposit

Label2:

Caption Yearly Interest

Label3:

Caption Number of Months

Text1:

Text [Blank]

Name txtDeposit

Text2:

Text [Blank]

Name txtInterest

Text3:

Text [Blank]

Name txtMonth