

Lesson 3

Working with Controls

This Lesson introduces the standard Visual Basic toolbox controls and teaches you how to use them to build useful interface features. In this Lesson, you will learn how to:

- Name Visual Basic objects.
- Use basic controls to display text and process input.
- Use file system controls to browse the files and folders on your computer.
- Use data input controls to display lists and check boxes.

2.1: Programmer Talk

Like any specialized activity, Visual Basic programming comes with its own evolving set of technical terms. Many of these terms have their roots in object-oriented programming, a methodology closely associated with C++ programming. As you work through the lessons in this course, you'll want to be familiar with these essential Visual Basic concepts.

Control

A control is a tool you use to create objects on a Visual Basic form. You select controls from the toolbox and use the mouse to draw objects on a form. You use most controls to create user interface elements, such as command buttons, image boxes, and list boxes.

Object

An object is a type of user interface element you create on a Visual Basic form by using a toolbox control. (In fact, in Visual Basic, the form itself is also an object.) You can move, resize, and customize objects by setting object properties. Objects also have what is known as inherent functionality — they know how to operate and can respond to certain situations on their own. (A list box “knows” how to scroll, for example.) You can customize Visual Basic objects by using event procedures that are fine-tuned for different conditions in a program.

Property

A property is a value or characteristic held by a Visual Basic object, such as **Caption** or **ForeColor**. Properties can be set at design time by using the Properties window or at run time by using statements in the program code. In code, the format for setting a property is:

Object. Property = Value

Where,

Object is the name of the object you're customizing.

Property is the characteristic you want to change.

Value is the new property setting.

For example,

```
Command1.Caption = "Hello"
```

Could be used in the program code to set the **Caption** property of the **Command1** object to “Hello”.

Event Procedure

An event procedure is a block of code that runs when a program object is manipulated.

For example, clicking the first command button in a program executes the `Command1_Click` event procedure. Event procedures typically evaluate and set properties and use other program statements to perform the work of the program.

Program Statement

A program statement is a combination of keywords, identifiers, and arguments in the code that does the work of the program. Visual Basic program statements create storage space for data, open files, perform calculations, and do several other important tasks.

Method

A method is a special keyword that performs an action or a service for a particular program object. In code, the format for using a method is,

Object.Method Value

Where,

Object is the name of the object you are working with.

Method is the action you want the object to perform.

Value is an optional argument to be used by the method.

For example, this statement uses the **Add Item** method to put the word *Check* in the **List1** list box:

```
List1.AddItem "Check"
```

Variable

A variable or identifier is a special container that holds data temporarily in a program. You create variables to store calculation results, create file names, process input, and so on. Variables can store numbers, names, property values, and references to objects.

2.2: Naming Visual Basic Objects

You may be a Visual Basic beginner now, but that won't be true for long. As your programs increase in size and sophistication, the number of objects you use on your forms will multiply quickly. There is an easy way to avoid mistaking one object for another in the Properties window or in your program code: assigning a unique name to each object soon after you creates it. It's simple — when you set your other object properties, just click the (**Name**) property, and then give the object a unique name.

➤ Visual Basic Object Names

Programmers typically create names for their objects that clearly identify the purpose of the object and the toolbox control that created it. For example, you might give the name **lblInstructions** to a label that displays user-operating instructions. (In this case, **lbl** stands for the **Label** control, and **Instructions** describes the label's purpose).

➤ Object Naming Conventions

This table lists the naming conventions for objects created by the 20 standard Visual Basic toolbox controls. (**Form** and **menu** objects, which you use often but which are not in the toolbox, are given the prefixes *frm* and *mnu* respectively.) Whenever you use more than five or six objects on a form, use this table as a naming guide:

Object	Prefix	Example
combo box	cbo	cboEnglish
check box	chk	chkReadOnly
command button	cmd	Open
data	dat	datBiblio
directory list box	dir	dirSource
drive list box	drv	drvTarget
file list box	fil	filSource
frame	fra	fraLanguage
horizontal scroll bar	hsb	hsbVolume
image	img	imgEmptyBarrel
label	lbl	lblInstructions
line	lin	linUnderline
list box	lst	lstPeripherals
OLE	ole	oleObject1
option button	opt	optFrench
picture box	pic	picSmokeCloud
shape	shp	shpWireScreen
text box	txt	txtGetName
timer	tmr	tmrRunAnimation
vertical scroll bar	vsb	vsbTemperature

2.3: Basic Controls

As you learned in Toolbox Controls, controls are the design tools you use to build the user interface of a Visual Basic program. After you place controls on a form, they become objects that you can customize with property settings and program code.

In this section, you learn about the most basic user interface controls in the Visual Basic toolbox. This section includes the following topics:

➤ Label



Label, the simplest control in the Visual Basic toolbox, displays formatted text on a user interface form. Typical uses for the **Label** control include:

- Help text
- Program **splash screen** headings
- Formatted output, such as names, times, and dates
- Descriptive labels for other objects, including text boxes and list boxes.

A **label** is a control you use to display text that a **user can't edit directly**. We've seen, though, in previous examples, that the text of a label box can be changed at run-time in response to events.

- **Label Properties:**

Alignment	Aligns caption within border.
Appearance	Selects 3-D or flat appearance.
AutoSize	If "True", the label is resized to fit the text specified by the caption property. If "False", the label will remain the size defined at design time and the text may be clipped.
BorderStyle	Determines type of border.
Caption	String to be displayed in box.
Font	Sets font type, style, size.
WordWrap	Works in conjunction with "AutoSize" property. If "AutoSize = True", "WordWrap = True", then the text will wrap and label will expand vertically to fit the Caption. If "AutoSize = True", "WordWrap = False", then the text will not wrap and the label expands horizontally to fit the Caption. If AutoSize = False, the text will not wrap regardless of WordWrap value.

- **Label Events:**

Click	Event triggered when user clicks on a label.
DbClick	Event triggered when user double-clicks on a label.

Creating Labels on a Form

To create a label control on a form, we surely refer to the toolbox window to select the label icon, shown as capital letter "A". When the label control is selected, the label can be placed on a form by creating a rectangle with the mouse, which is held left button clicked. Once the left button is released, the label of size as the rectangle created is placed on the form.

Creating Labels in Code

You can also set label properties with program code, as shown in the following. The program codes below, when command1 button is clicked, will set the caption of label1 as "Welcome" and label2 as "Please enter your name below:"

```
Private Sub Command1_Click ()
    Label1.Caption = "Welcome"
    Label2.Caption = "Please enter your name below:"
End Sub
```

➤ Textbox



A **Textbox** 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.

The **Textbox** control is one of the most versatile tools in the Visual Basic toolbox. This control performs two functions:

- **Displaying output** (such as operating instructions or the contents of a file) on a form.
- **Receiving text** (such as names and phone numbers) as user input.

How a text box works depends on how you set its properties and how you reference the text box in your program code. The following popup illustration shows a text box that:

- Displays data entry instructions for the user.
- Receives input when the user types in a delivery note and clicks the **Print Order** button.

- **Text Box Properties:**

Appearance	Selects 3-D or flat appearance.
BorderStyle	Determines type of border.
Font	Sets font type, style, size.
MaxLength	Limits the length of displayed text (0 value indicates unlimited length).
MultiLine	Specifies whether text box displays single line or multiple lines.
PasswordChar	Hides text with a single character.
ScrollBars	Specifies type of displayed scroll bar(s).
SelLength	Length of selected text (run-time only).
SelStart	Starting position of selected text (run-time only).
SelText	Selected text (run-time only).
Tag	Stores a string expression.
Text	Displayed text.

- **Text Box Events:**

Change	Triggered every time the Text property changes.
LostFocus	Triggered when the user leaves the text box. This is a good place to examine the contents of a text box after editing.
KeyPress	Triggered whenever a key is pressed. Used for key trapping, as seen in last class.

- **Text Box Methods:**

SetFocus	Places the cursor in a specified text box.
-----------------	--

The sample code below shows how to control a text box, **Command 1 is clicked**,

```
Private Sub Command1_Click()
'The following line places text in a text box:
    Text1.Text = "Enter your delivery note here."
End Sub
```

Command 2 is clicked,

```
Private Sub Command2_Click()
'This line saves input from a text box in a variable:
    DeliveryNote = Text1.Text
End Sub
```

➤ **CommandButton**



As you saw in Lesson 2 you use the **CommandButton** control to create buttons with a variety of uses on a form. A command button is the most basic way to get user input while a program is running. By clicking a command button, the user requests that a specific action be taken in the program. Or, in Visual Basic terms, clicking a command button creates an event, which must be processed in your program.

Here are some command buttons that you'd typically find in a program:

OK	Accepts a list of options and indicates that the user is ready to proceed.
Cancel	Discards a list of options.
Quit	Exits an open dialog box or program.

In each case, you should use command buttons carefully so that they work as expected when they are clicked.

- **Command Button Properties:**

Appearance	Selects 3-D or flat appearance.
Cancel	Allows selection of button with Esc key (only one button on a form can have this property True).
Caption	String to be displayed on button.
Default	Allows selection of button with Enter key (only one button on a form can have this property True).
Font	Sets font type, style, size.

- **Command Button Events:**

Click	Event triggered when button is selected either by clicking on it or by pressing the access key.
--------------	---

Changing Command Button Properties

You can change command button **properties** (like those of all objects) in two ways:

- By adjusting property settings in the Properties window.
- By changing properties with program code.

2.4: Data Input Controls

Visual Basic provides several other controls that you can use to collect user input in a program. In this section, you learn to use four versatile toolbox controls.

This section includes the following topics:

➤ OptionButton

To receive user input in a Visual Basic program, you can use controls that:

- Present a range of acceptable choices.
- Receive input in an acceptable format.

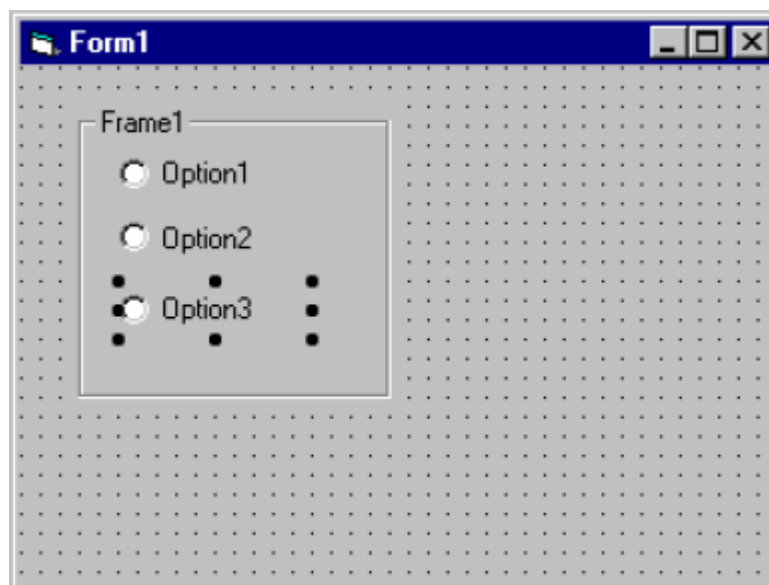
The **OptionButton** control satisfies these requirements. It displays a list of choices on a form and requires the **user to pick one (but only one) item from a list of choices**.

Creating Option Buttons

Like the other items in the Visual Basic toolbox, you create an option button on a form by clicking **OptionButton** in the toolbox and then drawing on the form.

Grouping Option Buttons

If you want to create a group of option buttons that work together, you must first create a frame for the buttons. (To do this, use **Frame**, a Visual Basic toolbox control.) Next, place your option buttons inside the frame so that they can be processed as a group in your program code and moved as a group along with the frame. See figure below.

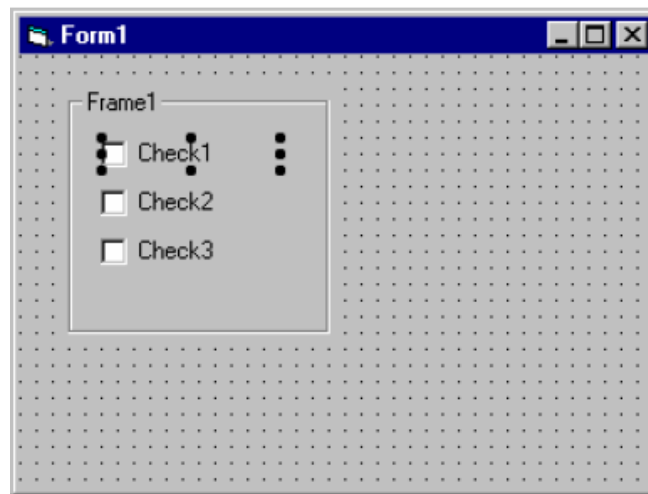


Option buttons, grouped by a frame.

➤ Checkbox

The **Checkbox** control is similar to the **OptionButton** control, except that **Checkbox** displays a list of choices and gives the user the **option to pick multiple items (or none at all) from a list of choices**.

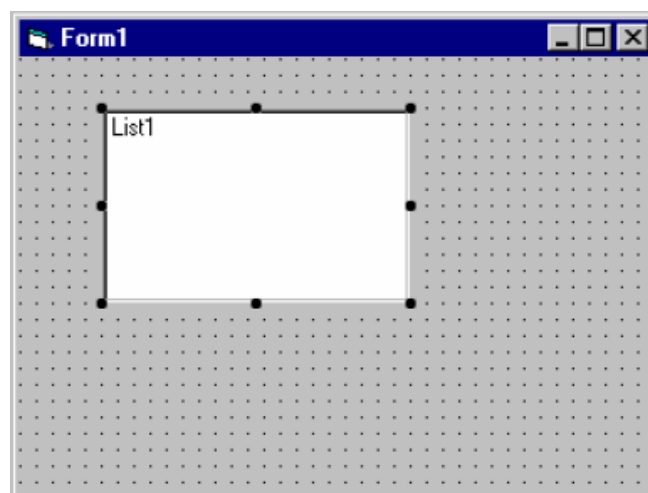
You create a check box on a form much as you would make an option button. Start by clicking the **Checkbox** control in the toolbox, and then draw on the form. If you want to create a group of check boxes that work together, create a frame for the buttons with **Frame**, a toolbox control. Then, you can place your check boxes inside the frame so that they can be processed as a group in your program code or moved as a group along with the frame. See figure below.



A Group of check boxes surrounded by a frame

➤ ListBox

When you want a user to choose a single response from a long list of choices, you use the **ListBox** control. (Scroll bars appear if the list is longer than the list box.) Unlike option buttons, however, list boxes don't require a default selection.



The figure showing a list box with room for six or seven visible items.

Creating List Boxes

Building a list box is simple, since it doesn't require creating a frame or a separate object for each list item. You just click **ListBox** in the Visual Basic toolbox and draw a suitably sized list box on your form.

Setting List Box Properties

In a Visual Basic program, you can define list box characteristics in the **Properties window** or by using program code, so that users can add, remove, or sort list box items while the program runs.

You add choices to a list box with the **AddItem** method, which is typically placed in the **Form_Load** event procedure. (**AddItem** is a method normally used with list-oriented objects).

The sample code below shows an example of the **AddItem** method.

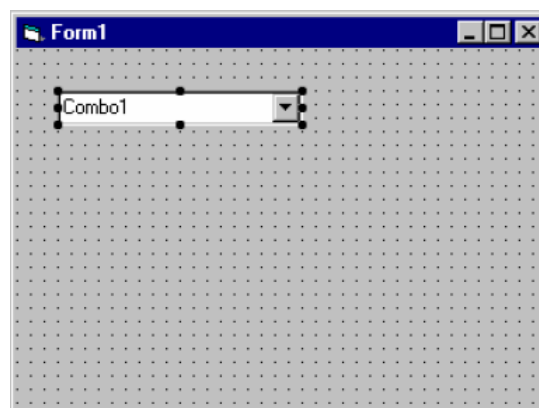
```
Private Sub Form_Load()  
    List1.AddItem "Extra hard disk"  
    List1.AddItem "Printer"  
    List1.AddItem "Satellite dish"  
End Sub
```

Adding Default Program Statements

Each time your Visual Basic program displays a form, the **Form_Load** event procedure (a block of code) runs. Although **Form_Load** doesn't contain default program statements, you can add statements to it. Just double-click the form to open the **Form_Load** event procedure in the code window.

➤ ComboBox

Objects created with the **ComboBox** control are similar to those created with the **ListBox** control. Like a list box, items can be added to, removed from, or sorted in a combo (combination) box while your program runs. However, there are differences. Combo boxes take up less space on the form, show a default setting, and in their default **Style** property setting are only one line high. But if the user clicks the combo box while your program runs, the combo box expands to reveal a full list of choices. (If the list is too long to be displayed in the combo box, scroll bars appear).



The illustration showing a combo box on a form.

Adding Combo Box Items

Typically, choices are added to a combo box with the **AddItem** method (a method normally used with list-oriented objects). You use the **AddItem** method by writing code in the `Form_Load` event procedure.

The sample code below shows an example of the `Form_Load` event procedure.

```
Private Sub Form_Load()  
    Combo1.AddItem "U.S. Dollars"  
    Combo1.AddItem "Check"  
    Combo1.AddItem "English Pounds"  
End Sub
```