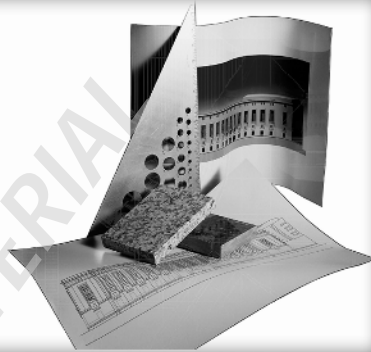


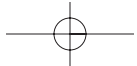
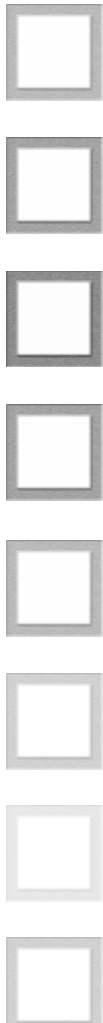
Part 1

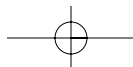
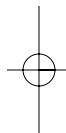
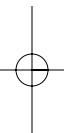
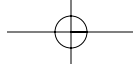
VBA Macros and the Visual Basic Editor

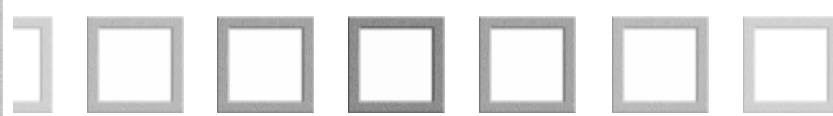


In This Part

<input type="checkbox"/>	Chapter 1: Developing a Simple VBA Application	3
<input type="checkbox"/>	Chapter 2: Creating VBA Macros	29
<input type="checkbox"/>	Chapter 3: Quick Tour of the IDE	53
<input type="checkbox"/>	Chapter 4: VBA Programming Concepts	79








Developing a Simple VBA Application

AUTOCAD VBA

Chapter 1



In this chapter, you will learn how to create a simple VBA application through a series of tutorials. You will learn how to employ visual tools to create the graphical user interface, and write code to execute simple commands. You will also explore the basics of the Visual Basic environment by using its features to create practical code that you can reuse for repetitive tasks.

If you consider yourself an intermediate user of AutoCAD VBA and have already written some successful VBA code, you may want to skip this chapter—or even just skim over the entire Part I, depending on your experience. The material covered in Chapter 4, “VBA Programming Concepts,” will always be useful to you as a reference.

This chapter covers the following topics:

- Advantages of using VBA with AutoCAD
- The AutoCAD VBA environment
- Developing your first application

Advantages of Using VBA with AutoCAD

Visual Basic for Applications (VBA) is a programming environment created by Microsoft that is built into applications to automate operations. It provides tools that you can drag and drop to build a graphical user interface (GUI), and a programming language that you can use to interact with AutoCAD objects. Using VBA with AutoCAD allows you to customize your AutoCAD application in seemingly unlimited ways. In this book, you'll see how simple it is to automate repetitive tasks. With the time saved, you'll be free to concentrate on applying your artistic talents and engineering skills to make your drawings more intricate.

Once you start writing VBA code, you'll quickly realize just how easy it is to access objects from the AutoCAD and VBA object libraries. You'll soon be able to call on the power of these objects while gaining a deeper insight into AutoCAD's features. As you start to see the benefits that VBA macros and applications can provide, you'll want to spend your extra time customizing even more tasks. Before you know it, you'll have a whole library of reusable macros and applications at your fingertips!



A macro is a group of code statements, usually not very long or complicated, that is useful in automating a repetitive task. In your work at the computer, most likely you've used them already without realizing it.



An application is a program that has been created to perform a specific task. This can be something very simple, such as prompting the user for their name and password, or something very substantial and complex such as AutoCAD itself.

Another not-so-obvious advantage of learning AutoCAD VBA is that your skills are transferable to a growing number of other applications that have VBA capability. These applications include all those in the Microsoft Office family of applications, such as Access, Word, and Excel, in addition to Microsoft Visual Basic itself and about two hundred other licensees.

VBA interfaces with applications by communicating and controlling objects through the application's object libraries, rather than by having any special connection to the application's inner workings. All you need to know are the names of the objects involved, and you can access their functionality with VBA code. You'll find the object names meaningful and easy to remember, such as `ThisDrawing` (an object in AutoCAD), `ThisDocument` (an object in Word), and `ThisWorkbook` (an object in Excel). As you enter the names of objects in your code, the editing features of VBA's Integrated

Development Environment (IDE) offer you drop-down lists of elements particular to the type of object you've entered. You'll see how this works in Chapter 2.

You can use a macro to automate just about anything, but it is probably most productive to start with the tasks you have to do time and time again. Why not think about those boring, uninspiring jobs you hate doing the most, and start with them! These are probably tasks that are time consuming and error prone, so implementing them in macros or applications will not only increase your productivity but lift your spirits by removing some of the tedious tasks that make you yawn just by thinking about them. (Such as adding all the required bits and pieces of information for starting a new drawing; or perhaps you've found that you regularly draw the same items over and over again.)

After you've successfully developed VBA code that works in the ways you want it to, it's guaranteed to perform correctly and reliably each time you run it. Now *there's* an incentive to learn VBA! So let's begin by examining the interface where you will write your code—the Visual Basic Editor.

The AutoCAD VBA Environment

The *Visual Basic Editor* is the integrated environment in which you develop all your VBA code. As you can see from Figure 1.1, the IDE has its own graphical user interface. Its windows provide all the tools required for creating, editing, debugging, and running your macros and applications. With this much functionality, the IDE is almost a stand-alone application, except that it can only be opened from the AutoCAD window and does not remain open after AutoCAD has been closed.



The term VBA Editor is often used synonymously with the IDE even though the IDE provides more than just editing features.

The VBA IDE can be opened from the AutoCAD window in a variety of ways:

- Type **vbaide** next to the command prompt in the command line.
- Choose Tools → Macro → Visual Basic Editor.
- Use the key-combination (shortcut) Alt+F11.

In the examples throughout this book, I've used menu commands in instructions. Most commands can also be accessed through a toolbar button or a key-combination, so occasionally I give an alternative access method as a reminder that there are other ways to invoke commands. The access method you use is up to your personal preference.

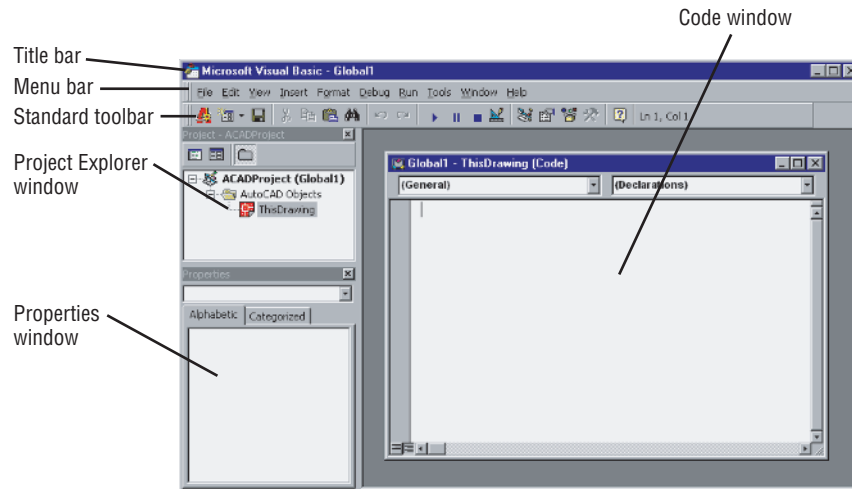


Figure 1.1 AutoCAD's VBA environment

Chapter 3 takes you on a guided tour of the IDE and discusses some of its features in more detail.

Creating UserForm Modules

A *module* is just a container for VBA code, and a *UserForm module* is a window (or dialog box) that can be considered a powerful extension to the AutoCAD GUI. You decide what controls to place on your UserForm in order to perform the tasks required by your application. You can place labels and text boxes in which users can enter the information that your program needs in order to run. You can use option buttons and check boxes to give users the chance to select the items they require. In fact, just about any control you've seen in the applications running under Windows is available for use in the VBA IDE.

Adding a UserForm Module to Your Application

To add a UserForm module to your application:

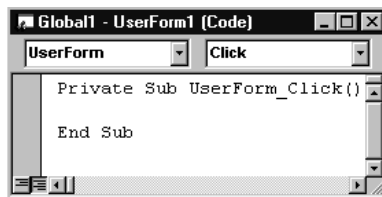
1. Start AutoCAD and choose Tools → Macro → Visual Basic Editor. The VBA IDE opens.
2. Choose Insert → UserForm. UserForm1 appears, containing a title bar with a Close button, as shown in Figure 1.2.



Figure 1.2 A brand-new UserForm

A UserForm is the only module that can be viewed from the IDE in two different ways:

- Graphically, by choosing View → Object to open up the UserForm window (see Figure 1.2).
- With code, by choosing View → Code to open up the Code window as shown here:



Opening the Code window displays the *event procedure* that will run if the UserForm is clicked. There are many events to choose from, but double-clicking any control automatically displays the primary event that you'll most likely want to respond to. You'll see how to access the other events later in this chapter, in the section "Coding Event Procedures."

Puttering Around in the Toolbox

VBA provides a set of common controls in a Toolbox window (see Figure 1.3). You can simply drag and drop individual controls onto a UserForm to develop GUI features for your application.

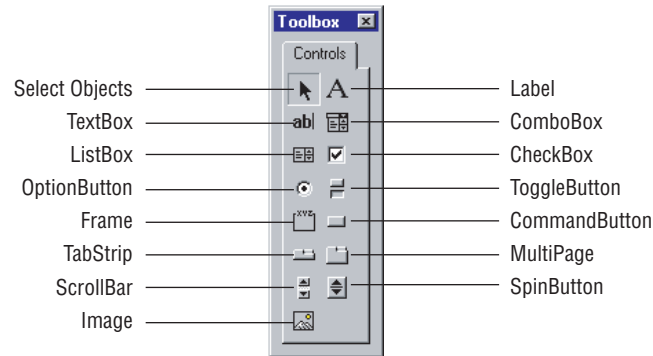


Figure 1.3 Controls in the standard Toolbox



If the Toolbox isn't on display, select the Toolbox command from the View menu or click the Toolbox icon (shown in the margin) at the end of the Standard toolbar. Both of these actions will be available for selection only when a UserForm has been added to a project and it is being viewed in its graphical form rather than as a Code window. To view a UserForm graphically, simply choose View → Object.

These controls are all available for you to drag and drop onto your UserForm, making AutoCAD VBA a powerful visual programming system that's extremely efficient at quickly developing the GUI for a project—you can virtually design your whole GUI using only the mouse!

You will recognize most of the controls in the Toolbox because you have already experienced using them in other applications. Following are brief descriptions of what each control does:

Select Objects The Select Objects control is used to resize and move a control after it has been drawn on a UserForm. If you've double-clicked another control to insert multiple instances of it on the UserForm without returning to the Toolbox, you can click the Select Objects control to stop.

Label The Label control displays text that you've assigned to its Caption property at design time or in code. A Label control is often used to display information or indicate to a user what data should be entered into a TextBox control. The text displayed by the Label control is strictly read-only and cannot be changed by the user during run time.

TextBox The TextBox control allows users to input textual information during run time. It is common practice to assign the empty string (" ") to this control's Text property to initialize it or to clear out any data that's no longer required.

ComboBox The ComboBox control combines the functionality of the TextBox control with that of the ListBox control to provide the user with the option of entering text into the TextBox, or clicking the down-arrow button and selecting an item from a drop-down list. When an item is selected from the list, it is automatically displayed in the text box at the top of the ComboBox. You can set the Style property of this control in order to restrict the user to selecting an item from the ListBox. This eliminates the need to test whether the user has entered a valid value.

ListBox The ListBox control enables the user to select one or more items from a list. If there are too many items to be displayed at once, a scroll bar will automatically appear. Even so, this control generally takes up more space than a ComboBox control, which needs only the same amount of space on the UserForm as a TextBox control.

CheckBox The CheckBox control enables the user to “check” (select) zero or more boxes from a group of CheckBoxes. This control is often used to indicate whether items are true or false, or to answer yes or no to questions. CheckBoxes are similar in function to OptionButtons, except that they allow more than one item to be selected at the same time.

OptionButton OptionButton controls are used in groups of options to allow selection of one option from the group. When only one group of OptionButtons is required, it can be placed directly onto the UserForm. When there is more than one group, Frame controls (described just below) are used to separate groups, to enable one OptionButton per group to be selected. You determine the OptionButton that will be the default selection. If the user selects another OptionButton from the same group, the old one is automatically deselected.

ToggleButton The ToggleButton control toggles between on or off, true or false, and yes or no. When clicked, its appearance changes to match its value; it toggles between a raised button appearance (off) to a pushed-in appearance (on). The Picture property of this control allows you to display a selection of options to the user in a graphical way.

Frame The Frame control allows a UserForm to be divided into areas where you can group other controls, such as OptionButtons and CheckBoxes. The Frame control and the controls inside it then behave collectively as a single entity. When you move the Frame, all the controls inside it move, too; when you disable the Frame, all its controls become disabled. Frames are typically used to create groups of OptionButtons on a UserForm, so that an option from each group can be selected rather than just a single option.

CommandButton The CommandButton control is used when you want something to happen as soon as the button is clicked. It is often used in dialog boxes with the caption “OK” for users to click when they’ve finished reading the message displayed to them. This control is useful for running macros by calling the macro in response to the control being clicked.

TabStrip The TabStrip control contains several pages, each displaying an identical set of controls. Adding a control to any page of the TabStrip control makes it appear on all pages. This control is used to enable viewing of several sets of data containing the same kind of information, one set per page. You provide the code to update the information in the controls according to the tab selected by the user. For example, you could display the name, address, and telephone number of your clients, one client per page.

MultiPage The MultiPage control is similar to the TabStrip control in that it has several pages that can be accessed by clicking their tabs. Each page from the MultiPage control has its own individual set of controls to enable the display of different kinds of information on each page, although the information would typically be related in some way. For example, you could display the name and address of a client on the first page, and information about the client’s orders on other pages. In such a scheme, you would need to create a MultiPage control for each client.

ScrollBar The ScrollBar control allows you to place a scroll bar on a UserForm. The scroll bar can be either vertical or horizontal depending on its width and height. This control provides scrolling for controls that do not have scroll bars added automatically. Appropriate values are set for the control’s Min and Max properties to define the limits for the Value property of the ScrollBar. The control’s Value property will be set to a value dependent on the position of the slider bar in relation to each end of the scroll bar.

SpinButton The SpinButton control functions similarly to a ScrollBar control but doesn’t have a slider bar. Up- or down-arrow buttons are displayed for the user to click in order to increment or decrement the number assigned to the control’s Value property. The Value is typically displayed in another control, such as a TextBox or a Label.

Image The Image control is used to display graphics stored in the major graphical formats, such as bitmaps (.bmp), GIFs (.gif), JPEGs (.jpg), metafiles (.wmf), and icons (.ico). The image can be cropped, sized, or zoomed as required to fit the control’s size. Image controls can be used as fancy command buttons or as toolbar buttons and can even display simple animations.

CHAPTER ONE • DEVELOPING A SIMPLE VBA APPLICATION

You can place as many instances of each control as you like onto the same UserForm. Visual Basic gives each instance a default name based on the name of the class it belongs to, appended to a sequential number—this enables each object to be uniquely identified. Visual Basic also assigns the properties of the control's position and dimensions on the UserForm based on where you've placed it and what size you've made it. All other properties are assigned a default setting.

Placing a Toolbox Control in a UserForm

This tutorial shows you how to drag and drop a TextBox control from the Toolbox onto a UserForm. The same technique is used to place any of the other controls onto a UserForm.

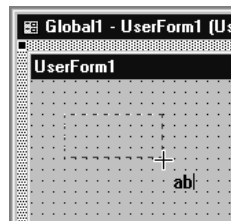
1. Click the TextBox icon (shown here) in the Toolbox and, without pressing the mouse button, move the mouse cursor over the UserForm.

After you click the TextBox icon, the Toolbar button changes its appearance to look pressed in, as shown here:

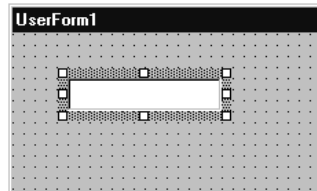


And the mouse cursor changes, too, as it moves over the boundary of the UserForm. It becomes a cross alongside the TextBox control's icon.

2. Move the mouse cursor to the position where you want one of the corners of your text box to be, and press the left mouse button. While holding the mouse button down, move the cursor around and watch a rectangle with dashed lines follow the cursor, with one corner anchored at the position you've just selected. This rectangle indicates the size and position for the new text box that will be created when you release the mouse button.



3. Still holding down the mouse button, move the cursor to the corner diagonal to the initial (anchored) one and release the mouse button. The new text box object replaces the dashed rectangle on the screen:

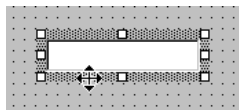


This becomes the new active control, as indicated by the handles and thick border that now appear around the rectangle. These can be used to make further adjustments to the size and position of the text box.

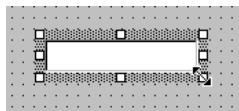
Changing the Dimensions and Position of a Toolbox Control

Click any control on a UserForm to make it the active control. Once active, the control is enclosed by a thick border with eight handles placed around it. You can click and drag these handles to change the position and dimensions of the control, as follows:

- If you move the cursor directly over the thick border but not on top of a handle, the cursor changes to a cross with two double-headed arrows (shown here) to allow you to reposition the whole control without changing its size. Simply click and hold down the mouse button and drag the control to its new position.



- If you move the cursor directly over any handle, the cursor changes to a double-headed arrow (shown here). This allows you to drag and drop the handle to change the size of the control while maintaining its position.



Now that you have an understanding of how to place components on your GUI, it's time to use these skills to develop your first simple application—I'm sure you're revved up ready to go.

Developing Your First Application

For your first application, we'll work with the Metric-Imperial Converter application, which converts metric measurements to imperial, and vice versa. This application uses two text boxes as shown in Figure 1.4. The interface allows the user to enter imperial measurements in yards into the first text box, then converts the measurement to metric and displays the results in the second text box. Alternatively, the user can enter metric measurements in meters into the second text box, and the application converts the value to imperial and displays it in the first text box.

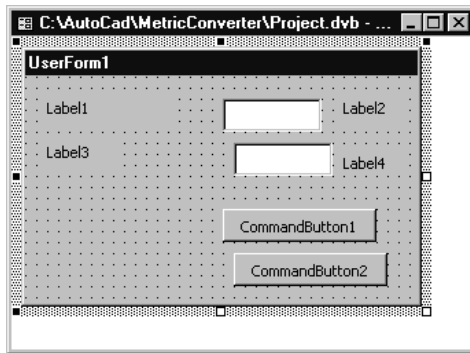


Figure 1.4 GUI for the Metric-Imperial Converter application before any properties have been updated

One command button is needed for the user to click in order to tell the application that the measurement has been entered. The second command button allows the user to stop the application.

Creating the GUI

Creating the GUI for the Metric-Imperial Converter application can be achieved using a single UserForm. Exercise 1.1 takes you through the steps required.

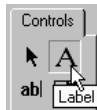


EXERCISE 1.1: METRIC-IMPERIAL CONVERTER APPLICATION

1. Choose Tools → Macro → Visual Basic Editor to open the IDE.
2. Choose Insert → UserForm to create a UserForm. The graphical representation of the UserForm is displayed, along with the Toolbox.
3. Drag and drop a TextBox control from the Toolbox onto the UserForm and place it near the top right. In Figure 1.4, the two empty boxes are TextBox controls.

The Name property of this text box is assigned as TextBox1 by default. The Name property of a control is the first item listed in the control's Properties window, which you'll see in the upcoming section "Setting Captions in the Properties Window." Default names all start with the type of control followed by a number to denote its place in the sequence of controls of the same type. So the next text box you add will be named TextBox2.

4. Drag and drop a Label control (shown here) and place it to the left of the TextBox control.



The Name property of this Label control is Label1 by default, which is also the initial value assigned to its Caption property. The Label1 control is used to label the TextBox so that the user knows what its contents represent and can enter appropriate values.

5. Drag and drop a second label and position it to the right of the TextBox control. The Name property of this Label control is Label2 by default.
6. Drag and drop a second text box with two accompanying labels, and place them immediately below the first text box and labels. The default Name properties of these controls are TextBox2, Label3, and Label4.
7. Drag and drop a command button (shown here) from the Toolbox onto the UserForm, placing it below the second text box. The default Name and Caption properties of this command button are both CommandButton1.

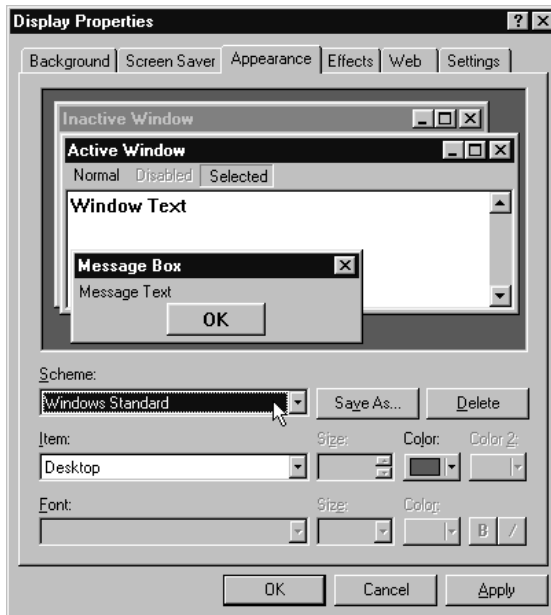


8. Drag and drop a second command button and place it directly below the first. The Name and Caption properties of this command button are—you guessed it—CommandButton2.
9. Now that all the controls required by the GUI have been added to the UserForm, adjust the UserForm's dimensions by dragging and dropping its borders until it is a snug fit for all the controls you've placed inside it. The layout of controls should look similar to the one shown earlier in Figure 1.4.

How Default Settings Are Initiated

A lot of the default object settings, such as the color of the title bar and background, are retrieved from the *display scheme* set in the Microsoft Windows environment of the PC on which the application is running. In the examples throughout this book, the Windows Standard display scheme is used. If your PC is set to a different scheme, your GUI will have a similar appearance to some of the other Windows applications that run on your PC. Some default settings such as those specifying position and dimensions are determined as you drag and drop controls onto UserForms.

You can find out the display scheme setting for your PC by clicking the Display icon in Control Panel and selecting the Appearance tab in the Display Properties dialog box—the current setting is displayed in the Scheme list box, as shown here:



Setting Captions in the Properties Window

The following tutorial shows you how to set the Caption property of UserForm1 in the Metric-Imperial Converter application:

1. If the Properties window is not already on display, choose View → Properties Window.
2. The Properties window appears, with property names listed in the left column and their settings listed in the right column. Select the appropriate tab to list the properties in this window alphabetically, as shown in Figure 1.5, or categorically, as shown in Figure 1.6. The list of properties belongs to the object named in the list box at the top (the active object), which is followed by the class the object belongs to.

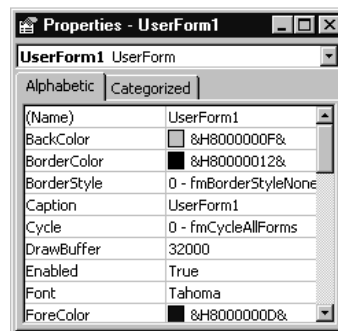


Figure 1.5 Properties window Alphabetic tab

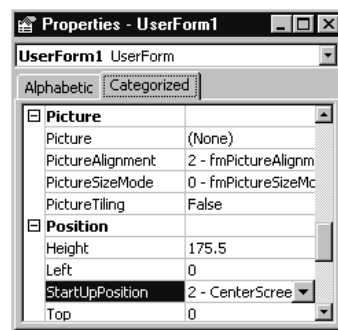


Figure 1.6 Properties window Categorized tab

CHAPTER ONE • DEVELOPING A SIMPLE VBA APPLICATION

3. If the object named at the top of the Properties window is not already UserForm1, click the down-arrow button at the right end of the Object box and select UserForm1 from the drop-down list.

This drop-down list contains all the objects associated with the active UserForm. When UserForm1 is selected, all the properties listed in the window now belong to UserForm1. The graphical representation of UserForm1 becomes the active object and appears with a thick border and eight sizing handles.

4. Scroll the list of properties until the Caption property becomes visible in the left column and select it by double-clicking it. Both the Caption property and the setting “UserForm1” are highlighted, and the insertion point (I-beam mouse pointer) blinks at the end of the highlighted text in the settings column on the right.
5. Type **Metric-Imperial Converter**. This replaces the highlighted setting for the Caption property, and the text in the title bar of UserForm1 is updated as you enter each keystroke.



Although the Name and Caption properties start off with the same default setting, they are still two distinct properties, so the Name property still remains set to UserForm1 even after you change the Caption.

6. Repeat steps 2 through 5 to change the Caption properties of the labels and command buttons to those shown in Table 1.1.

Table 1.1 Controls and Their Caption Properties

CONTROL	CAPTION
Label1	Imperial Units
Label2	Yards
Label3	Metric Units
Label4	Meters
CommandButton1	Convert
CommandButton2	Close

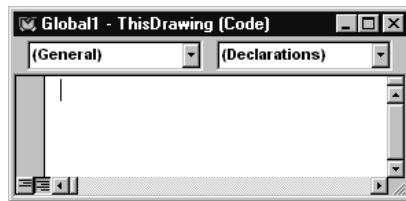


The Properties window can be opened by selecting any graphical object and choosing View → Properties Window, or by right-clicking and selecting Properties from the shortcut menu, or by pressing the F4 function key.

The VBA Code Window

The *Code window* is where you enter the code that will interact with the user or manipulate AutoCAD objects. When you start a new AutoCAD project, a *drawing object* is automatically created and made the active document. You can refer to this object in code by its Name property, ThisDrawing. When you open the IDE and look in the Project Explorer window, the ThisDrawing object will be the only object listed (shown previously in Figure 1.1).

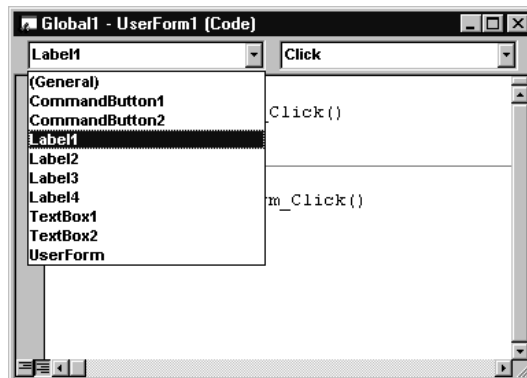
Every object listed in the Project Explorer window has its own Code window. One way to open the Code window for a particular object is to double-click it from the Project Explorer window. Here is the Code window for ThisDrawing:



Displaying the Code Window

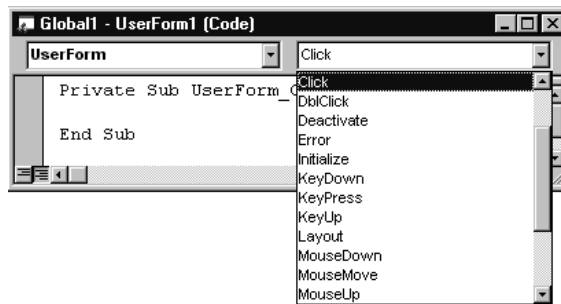
When double-clicked in the Project Explorer window, all the modules except UserForms will immediately display the Code window. When a UserForm is double-clicked, its graphical representation is displayed in the UserForm window. You must then double-click anywhere inside this UserForm window to display its Code window.

Every Code window has a drop-down list box of objects on the left, and a drop-down list box of procedures on the right. The Object list contains all the objects and controls attached to that module. In the example shown here, all the controls placed on UserForm1 are displayed.



CHAPTER ONE • DEVELOPING A SIMPLE VBA APPLICATION

You can tell which UserForm the Code window belongs to by looking at the Caption on the title bar. Notice in the preceding example that there's an object named UserForm in the drop-down list. Because a UserForm has its own Code window, there is no need to append a distinguishing number after UserForm when you're referring to it in code in its own Code window. The Procedure list box on the right contains a list of all the procedures (including event procedures) associated with the object named in the Object box, as shown here:



Coding Event Procedures

The following tutorial shows how to write the code to stop an application in response to the user's clicking the Close button:

1. If the Code window is displayed, choose View → Object to display the graphical representation of UserForm1. Double-click CommandButton2. The Code window appears, containing the skeleton code for the CommandButton2_Click event procedure, with the I-beam cursor blinking in the blank line.

2. As shown in Figure 1.7, type

```
Unload Me
```



Figure 1.7 The Code Window showing the CommandButton2_Click event procedure

Now you're ready to start coding the response you want given to the user who's entering data into one of the text boxes. When the Metric-Imperial Converter application is run for the first time, both text boxes will be empty. The user enters a measurement into one text box and clicks CommandButton1 to convert the measurement and display the result in the other text box. When CommandButton1 is clicked, the application checks which text box contains the measurement so that it knows which conversion is required. If the user requires another, subsequent conversion and enters another measurement into a text box, the application must clear the other text box of data so that the correct conversion is used. The following steps show you how these actions are coded:

1. Open the Code window and select TextBox1 from the Object list.
2. Select KeyDown from the Procedure list, as shown in Figure 1.8.

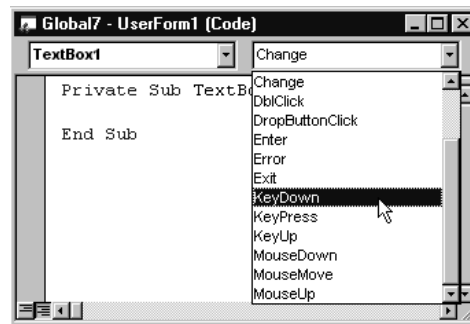


Figure 1.8 Drop-down list of procedures (events) available for the TextBox class of object

3. In the skeleton TextBox1_KeyDown event procedure, where the Entry cursor is blinking, type


```
TextBox2.Text = ""
```
4. Select TextBox2 from the Object list, and select KeyDown from the Procedure list.
5. In the skeleton TextBox2_KeyDown event procedure, type


```
TextBox1.Text = ""
```
6. Select CommandButton1 from the Object list, and the skeleton (first and last lines) of its Click event procedure will appear.

7. Type the following code inside the skeleton:

```
If TextBox1.Text = "" Then
    TextBox1.Text = TextBox2.Text * 1.0936
Else
    TextBox2.Text = TextBox1.Text * 0.9144
End If
```

That's all the coding necessary for this conversion application! Listing 1.1 provides a numbered listing of all the code, followed by an Analysis section that describes what each statement actually does. This application is available for use on the CD-ROM as Listing 1.1. You'll need to follow the instructions in the "Loading VBA Project Files" section in Chapter 2.



In the listings throughout the book, the numbers preceding each line of code are not part of the actual code. They are provided to help you follow the discussions in the Analysis sections.



LISTING 1.1: METRIC-IMPERIAL CONVERSION MACRO

```
1 Private Sub CommandButton1_Click()
2   If TextBox1.Text = "" Then
3     TextBox1.Text = TextBox2.Text * 1.0936
4   Else
5     TextBox2.Text = TextBox1.Text * 0.9144
6   End If
7 End Sub
8
9 Private Sub CommandButton2_Click()
10 End
11 End Sub
12
13 Private Sub TextBox1_KeyDown(↵
    ByVal KeyCode As MSForms.ReturnInteger, ↵
    ByVal Shift As Integer)
14   TextBox2.Text = ""
15 End Sub
16
17 Private Sub TextBox2_KeyDown(↵
    ByVal KeyCode As MSForms.ReturnInteger, ↵
    ByVal Shift As Integer)
18   TextBox1.Text = ""
19 End Sub
```

ANALYSIS

You'll find the code shown in Listing 1.1 quite straightforward. This Analysis section has been included to take you through it line by line.

- Line 1 is the opening statement and declares the `CommandButton1_Click` event procedure. The code in this event procedure will be executed when the user clicks the Convert command button. This statement, along with the `End Sub` statement in Line 7, is provided as a skeleton procedure by the IDE.
- Line 2 checks to see if the `Text` property of `TextBox1` is empty, meaning that the user has entered meters into `TextBox2`; if so, the program executes the statement at Line 3. If `TextBox1` isn't empty, it contains yards, and the statement in Line 5 is executed. The section "Using Conditions to Control Code Execution" in Chapter 4 gives more information on `If` statements.
- Line 3 converts the meters entered into `TextBox2` to yards and assigns the results to the `Text` property of `TextBox1`. The value of the `Text` property is displayed in the text box.



If conditions and Then...Else...End If... conditional structures can be thought of as two separate parts—the If code block and the Else code block.

- Line 4 starts the `Else` part of the `If` statement. Execution jumps to this point if the condition in the `If` statement (Line 2) is `False`. Line 4 also serves as an end marker for the statements in the `If` code block when the condition is `True`, in which case execution jumps to the `End If` statement at Line 6.
- Line 5 converts the yards entered into `TextBox1` into meters and displays the results in `TextBox2`.
- Line 6 signifies the end of the `If` statement. Execution jumps to this point from the `Else` statement (Line 4) if the condition in Line 2 is `True`.
- Line 7 is the `End Sub` statement, which marks the end of the `CommandButton1_Click` event procedure.
- Line 8 is a blank line inserted to make it easier to see where one event procedure ends and another starts.
- Line 9 starts the `CommandButton2_Click` event procedure. This is the command button that has its `Caption` property set to `Close` (see step 6 of "Setting Captions in the Properties Window").

CHAPTER ONE • DEVELOPING A SIMPLE VBA APPLICATION

- Line 10 contains the End statement that stops executing your application by closing any open files and freeing any memory used by your application during run time.
- Line 13 is the opening statement for the TextBox1_KeyDown event procedure. Unlike the Click event procedures found in lines 1 and 9, the KeyDown event procedure gets passed the integer value representing the ASCII code for the character just entered. This is useful if you want to validate the character input, which is handled in the section “Validating Input” in Chapter 9.
- Line 14 assigns an empty string to the Text property of TextBox2. The empty string is denoted by two double quote characters (" "). If a new conversion is starting, it doesn’t make sense to have any values from the last conversion still displayed in the other text box.
- Line 18 clears the Text property of TextBox1 inside the TextBox2_KeyDown event procedure.

Running Your Application

To run your Metric-Imperial Converter application:

1. Ensure that UserForm1’s window is open and selected.
2. Choose the Run → Sub/UserForm menu command.
3. Test that your application is working by entering a value into one of the text boxes and clicking the Convert button. Verify that the results are what you expect. Repeat this verification for the other text box.
4. You may want to line up the controls on your UserForm to improve its appearance, as in Figure 1.9.

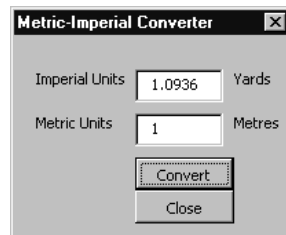


Figure 1.9 The finished version of the Metric-Imperial Converter application’s GUI with the captions set

Saving Your Application

The IDE has one Save command on the File menu. The exact wording of this command depends on whether you have previously saved your project. Figure 1.10 shows the Save option on the File menu *before* the project has been saved to a file. Figure 1.11 shows how it appears afterward—with the full pathname of the project file.

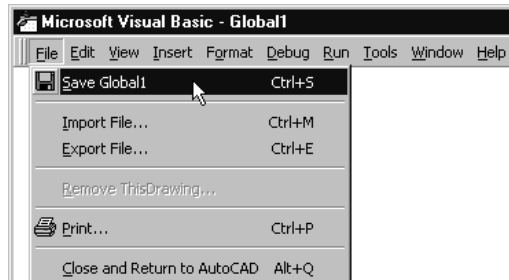


Figure 1.10 The File menu option for saving a VBA project that has never before been saved

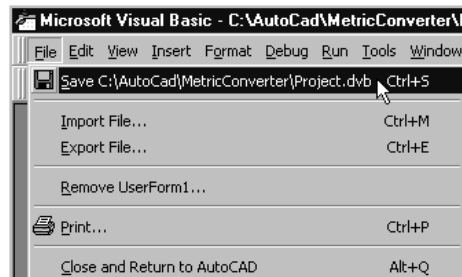


Figure 1.11 The File menu option for saving a VBA project that's been saved previously

To save your Metric-Imperial Converter project for the first time:

1. Choose File → Save Global1. The Save As dialog box shown in Figure 1.12 appears. The controls in this dialog box will be familiar to you; they are much the same as those found in the Save As dialog boxes of other applications, including AutoCAD itself.

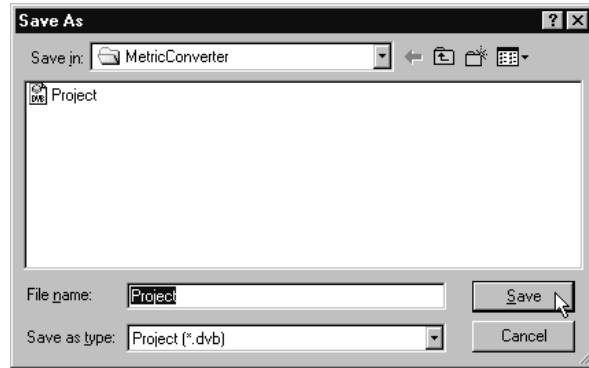


Figure 1.12 The Save As dialog box, ready to save your project to the .DVB file of your choice

2. Use the controls to create a new directory and click Save. Everything associated with the current Project will be saved in a single file with the extension .dwb.
3. Open the File menu. The Save command will now be followed by the full path of the .DVB file in which you've just saved your project.

Returning to AutoCAD

There are two menu commands that you can use to display the AutoCAD window, but they serve different purposes:

- Choose File → Close and Return to AutoCAD, or use the shortcut Alt+Q. This command unloads the IDE completely before returning to AutoCAD.
- Click the View AutoCAD button.



at the left of the Toolbar. This hides but does not close the IDE window and makes it accessible from the Taskbar. As you know from other Windows applications, you can much more quickly display a window that's already open by clicking its icon on the Taskbar rather than loading it in again from scratch.

Summary

After working through this chapter, you'll know how to

- Develop a simple VBA application.
- Open the VBA IDE.
- Drag and drop controls from the Toolbox onto a UserForm.
- Assign new values to Caption properties in the Properties window.
- Code event procedures to respond to the user's actions.
- Run your application.
- Save your application.
- Return to the AutoCAD window from the IDE.

