

Chapter 34

Visual Basic for Application

Learning Objectives

After completing this chapter you will be able to:

- Load and run sample VBA projects.
- Utilize the Visual Basic Editor.
- Understand and use AutoCAD objects.
- Use object properties.
- Apply and use AutoCAD methods.

ABOUT VISUAL BASIC

The original BASIC was developed in 1963. BASIC (Beginners All-purpose Symbolic Instruction Code) was intended to be an accessible, user-friendly programming language. BASIC was the language supplied with many of the early microcomputers starting in the late 1970s. It continued development until the advent of Windows. In 1991 the first version of **Visual Basic (VB)** appeared, followed by later versions spaced about a year apart. The current version 6, like its predecessors, lets even a beginning programmer create a user-friendly graphical user interface (GUI) with a small fraction of the effort required previously. It used to take a team of programmers working with a language like C, which came with a stack of reference materials nearly a foot thick, to do the same thing.

Another advantage inherent to Visual Basic is that engineers may have old BASIC programs used in sizing components. This code, minus old input/output, can be used as the core of new Visual Basic modules to parametrically design and draft the component.

VISUAL BASIC FOR APPLICATION

Autodesk first licensed **Visual Basic® for Applications (VBA)** from Microsoft for use in R14. Dozens of other large software companies have similarly adopted VBA. The VBA language tools are either identical or very similar to those in the stand-alone Visual Basic. Programmers who know how to write macros for Microsoft Office only have to learn AutoCAD-specific functionality to program VBA in AutoCAD.

One of the features of VBA is its ability to communicate with other applications such as Microsoft Excel, Microsoft Word, Microsoft Access, and Visual Basic using **ActiveX Automation**. Automation lets you access and manipulate the objects and functionality of AutoCAD from Excel or another application supporting ActiveX. Alternately the objects and functionality of Excel, for example, can be used inside AutoCAD VBA programming. This cross-application macro programming capability does not exist in AutoLISP. Of course, before you can use AutoCAD's objects in some other software supporting ActiveX, you must make the application aware that AutoCAD is installed along with its **Object Library** on the computer.

There are millions of programmers using Visual Basic. A great many more people have programmed in BASIC but not VB. This chapter assumes you have some familiarity with BASIC or programming concepts such as those covered in the AutoLISP chapter of this book and are comfortable looking up commands and syntax in the help system.

In this version of AutoCAD you can now reference a macro from another project and have more than one macro loaded. You can also create libraries of common functions and macros.

USING VBA IN AutoCAD

VB enables you to write a code that is compiled into an *.exe* file, which is an independently executable file (*.exe*). Whereas, VBA is related to the application and the document in which you write its code. VBA programs are written according to the need. For example, if your company manufactures nuts and bolts, you can use VBA to write a program that will show a

dialog box when run. You can use the dialog box to specify the values and dimensions of the nuts and bolts. This enables you to reduce the designing time considerably because whenever there is a change in the dimensions of these components, you do not have to invoke the different commands to create the components.

You can save VBA programs as projects containing the information required to compile and run. VBA in AutoCAD has a separate environment in which you can do all the work related to a project. This environment is called Integrated Development Environment (IDE). This environment is discussed next.

Invoking VB Editor

The VB editor of AutoCAD allows the user to write programming codes known as modules. Collectively these modules form a project. To invoke the VB editor, choose **Tools > Macros > Visual Basic Editor**. This window is also called **Integrated Development Environment (IDE)** window and is shown in Figure 34-1. The windows in this environment that are shown in the figure, are opened by choosing **Insert > UserForm** from the menu bar in the IDE environment.

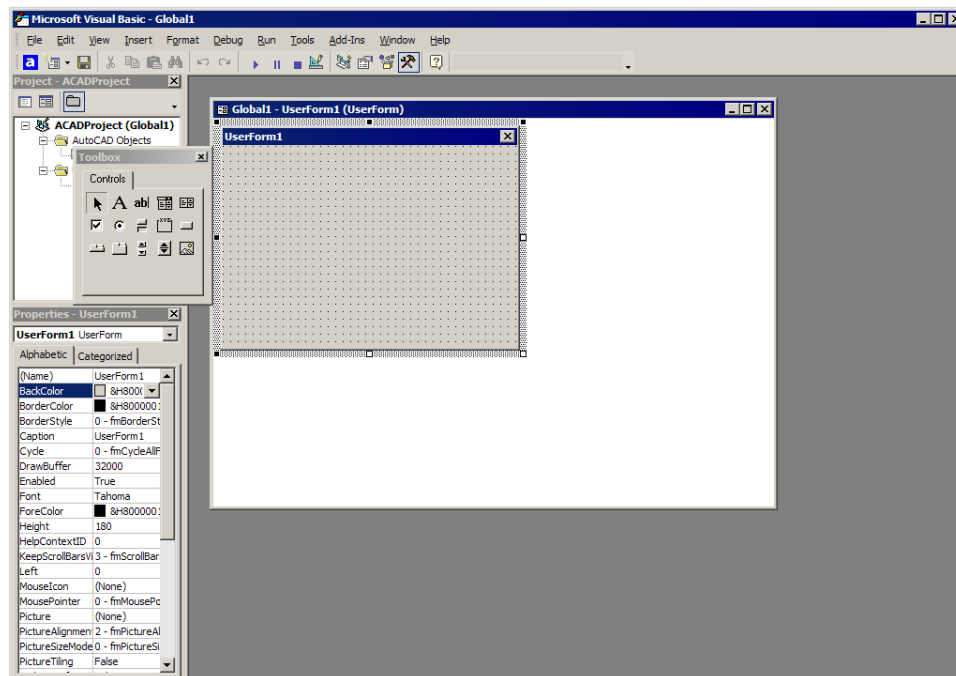


Figure 34-1 Visual Basic Editor with Project Explorer, Properties window, Module code window, User Form, and Toolbox

These windows are discussed next.

Project Explorer Window

The **Project Explorer** window lists all the modules, User forms, and other objects used in a project. The items in the **Project Explorer** window are known as objects. Objects are discussed later in the chapter. This window contains three folders, **AutoCAD Objects**, **Forms**, and **Modules** as shown in Figure 34-2. The **AutoCAD Objects** folder contains **ThisDrawing** object, which is the current AutoCAD drawing. The **Forms** folder contains Visual Basic forms used in the current project. A form is a custom dialog box you create to interact with the user. The **Modules** folder contains code modules for the current project. The code module object contains generic procedures and functions. When you add a form or code module to your project, VBA adds it to the corresponding folder in the Project Explorer window.

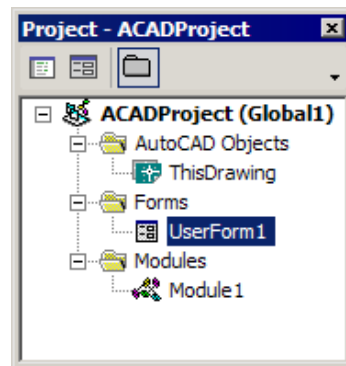


Figure 34-2 Project Explorer window



Note

The **Forms** and **Modules** folder are added in the **Project Explorer** window by choosing the options to add from the **Insert** menu in the menu bar.



If the **Project Explorer** window is not displayed, choose the **Project Explorer** button from the **Standard** toolbar.

The **Project Explorer** window is used when the **Module code** window is opened on the screen and you want to access the User form. For Example, if you want to view the code of the User form, right-click on the User form in the **Project Explorer** window to display a shortcut menu. Choose the **View Code** option from the shortcut menu. The **User form code** window is displayed.

Properties Window

The **Properties** window shows all the properties of a selected object. In VBA, object refers to the form and to every control that is on the form. These controls are buttons, textbox, labels, and other objects for user interface that can be added to the form. These objects have characteristics such as color, size, name, caption, background, and so on. These characteristics not only affect the appearance of the object but the way it respond to the user's action. All

these characteristics of the object are called properties that are displayed in the **Properties** window.

The **Properties** window shown in Figure 34-3 is used to modify the properties of the User form, buttons, and modules. You can select an object from the drop-down list present at the top in the window. This list contains all the objects that are in the form. On the left in the **Properties** window are the names of the properties and on the right are their value.

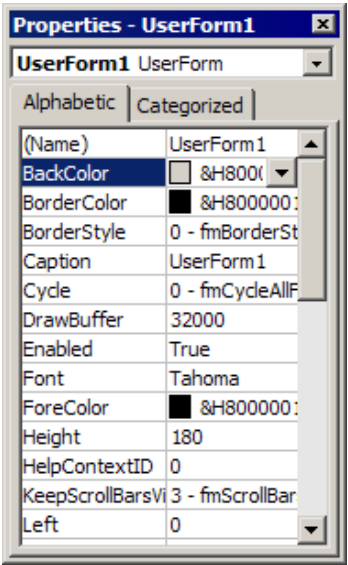


Figure 34-3 Properties window

Module Window

The **Module** window is used to type the code. To open the **Module** window, choose **Insert > Module** from the menu bar. The **Module** window appears as shown in Figure 34-4, and you can type the code in it. You can resize the **Module** window by dragging it from one of the corners or edges. By default, the first module is called Module1.

User Form and Toolbox

The User form shown in Figure 34-5, is the front end of the program and provides an easy way to interact with the user. On the User form you can place various controls like buttons, check boxes, radio buttons, and so on in order to get the user input. You can insert a User form by choosing **Insert > UserForm** from the menu bar in the VBA IDE. When the User

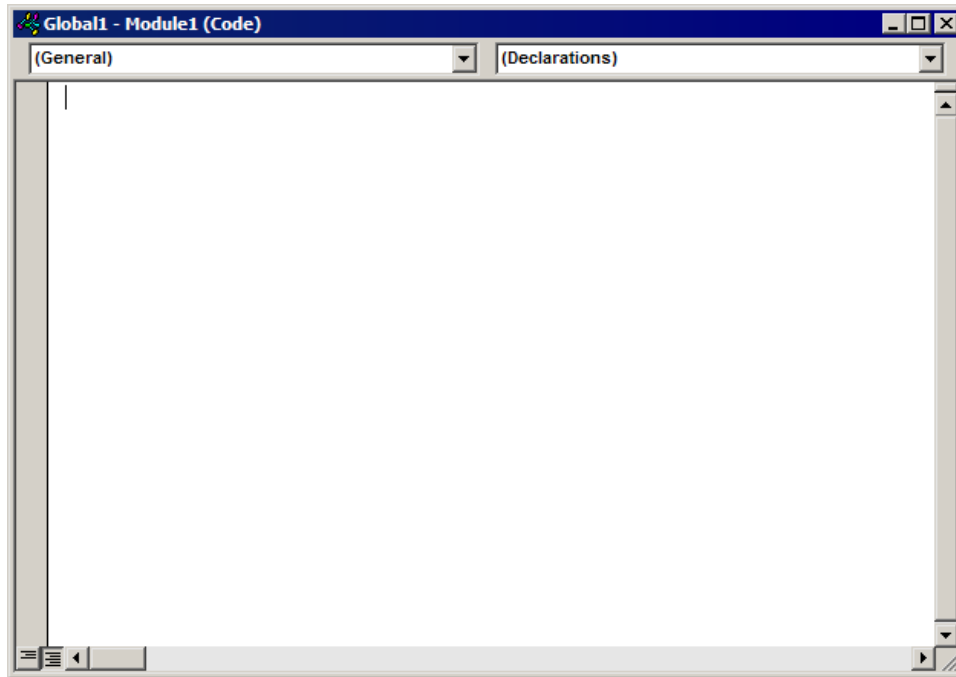


Figure 34-4 Module window

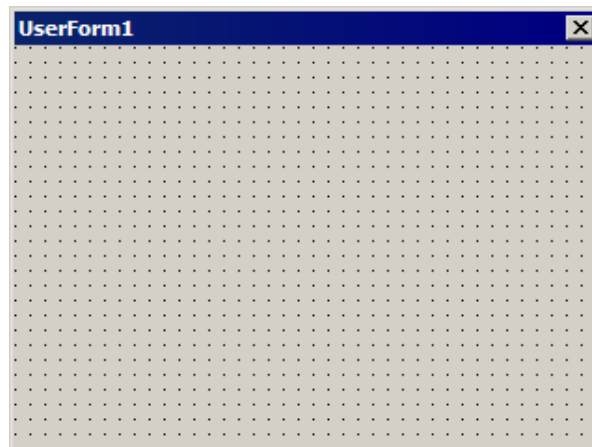


Figure 34-5 User form

form is inserted, the **Toolbox** appears as shown in Figure 34-6. This box contains the various controls that are added to the form by selecting and dragging to the form.

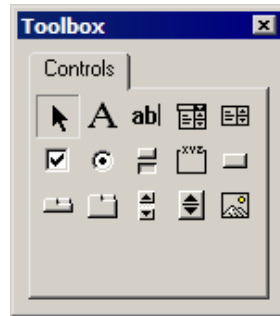


Figure 34-6 Toolbox

OBJECTS IN VBA

The primary purpose of VBA is to control the functionality of the application and automate it with VBA commands. This is made possible through **Objects**. This means that an application makes itself accessible to another application using **Objects**. The basic objects of AutoCAD are **Application object** and **Document object**.

Application Object

The application that you are working on is called an **Application object**. For example, AutoCAD is an **Application object**. There are some methods and properties associated with **Application object**. These methods are **Quit**, **ZoomAll**, **Runmacro**, and so on. The **Quit** method exits the application, the **ZoomAll** method zooms the objects and fits them in the current viewport, and **Runmacro** executes a macro for an application.

The properties of an Application object are height, width, caption, and so on. The height property specifies the height of the window, the width property specifies the width of the application window, and the caption is the string that appears on the application window's title bar. This property can be used to find out whether a window contains a specific document.

Document Object

The drawing that is opened in the application window is called **Document object**. A **Document object** is any document that can be opened with AutoCAD or any document or drawing that can be displayed in the AutoCAD window. All open documents are referred to by a documents collection that is formed by **Document objects**. The documents collection consists of the following methods and properties.

1. The **Count property**, which returns the number of items in the collection (number of drawings opened in a session).
2. The **Add method**, which creates a member object (drawing) and adds it to the appropriate collection.

Examples of objects include drawings or documents, geometric elements such as lines or circles, and user interface controls to handle input and output to programs or macros. The

“visual” part of Visual Basic refers to familiar user interface controls such as check boxes, scroll bars, and command buttons as used to open or save files in Windows applications. These controls are screen objects that may be dragged from a Toolbox onto the background object called a form. An example of a simple form is the background window of a dialog box. In the course of dragging a control from the Toolbox onto a form, the coding that provides the functionality of the control is simultaneously added to the form object. Several aspects of true object-oriented programming (OOP) languages such as C++ are missing from Visual Basic. So VB is considered an object-based, rather than an object-oriented, programming language. The missing aspects are more than made up for by the development tools and environment available to the user.

Functions known as **methods** have been defined in the AutoCAD object library to perform an action on an object, for example, drawing a line in a drawing. The method **AddLine** adds a line object to a drawing. **Properties** are functions that set or return information about the state of an object. In the example of a line drawn in a drawing the Color, Layer, Linetype, and Start X would be some of the properties of the Line object.

ADD METHOD

The way to draw in paper space, model space, or in a block is to use the **Add method** such as **AddCircle**, **AddLine**, **AddArc**, and **AddText**. Instead of thinking of drawing in the model space of the current drawing, think in terms of adding a geometry. **Dot notation** is required to clarify on which object the method is acting. The dot notation reference starts with the most global object first narrowing to the right. Dot notation is used in a similar way with properties.

Using the AddCircle Method to Draw a Circle

The **AddCircle** method requires a predefined center point and radius. Both of these arguments are required. Other input options used in the AutoCAD circle command such as three-points, two-points, or tangent-tangent-radius require additional programming in VBA to calculate the center point and radius for the **AddCircle** method. The format of the **AddCircle** method is given next.

ThisDrawing.ModelSpace.AddCircle centerpoint, radius

or

Set Circle1 = ThisDrawing.ModelSpace.AddCircle(centerpoint, radius)

where **centerpoint:** Center point, double precision vector

radius: Radius, double precision number

Examples:

1. **ThisDrawing.ModelSpace.AddCircle(cnt1, 10)**

The above code allows you to draw a circle whose center point is cnt1 and radius is 10 where cnt1 is predefined in the program.

2. **ThisDrawing.ModelSpace.AddCircle(0,1,3, 8)**

The above code allows you to draw a circle whose center point is (0,1,3) and radius is 8.

**Note**

The arguments of the first form of the **AddCircle** method follow a required space. The second form of the add method, where the arguments are inside parentheses, is used where the circle object is assigned to a name for use later in the program. For the examples below, *point1* and *point2* are predefined points consisting of a vector of variant/double precision or double precision coordinate values. These may be defined with assignment statements as shown in Example 1 on page 34-13.

The pound sign (#) signifies a double precision number in Basic. Double precision, floating point number are positive numbers in the range 4.94065645841247E-324 to 1.79769313486232E308 and negative numbers in the range -1.79769313486232E308 to -4.94065645841247E-324.

Examples

```
ThisDrawing.ModelSpace.AddCircle point1, 3#
Set Circle2= ThisDrawing.ModelSpace.AddCircle(point2, 4#)
```

Using the AddLine Method to Draw a Line

The **AddLine** method requires two predefined endpoints. The **AddLine** method has a syntax similar to **AddCircle** as given next.

ThisDrawing.ModelSpace.AddLine firstpoint, secondpoint

where **firstpoint:** First point, double precision vector
secondpoint: Second point, double precision vector

1. ThisDrawing.ModelSpace.AddLine(x1, x2)

The above code draws a line from point x1 to x2, where x1 and x2 are predefined in the program.

2. ThisDrawing.ModelSpace.AddLine(0,1,1, 5,8,0)

The above code draws a line from coordinate (0,1,1) to (5,8,0).

**Note**

The second form of the Add method, where the arguments are inside parentheses, may be used for the **AddLine** method or any of the other **Add** methods below where the object is assigned to a name for later use in the program. A complication for using the Add method this way is that the object defined must be declared in a **Dim** statement in the General declarations.

Examples:

```
ThisDrawing.ModelSpace.AddLine point1, point2
Set Line1=ThisDrawing.ModelSpace.AddLine(point1, point2)
```

Using the AddArc Method to Draw an Arc

The **AddArc** method format is given next.

ThisDrawing.ModelSpace.AddArc ctrpt, radius, StartAng, EndAng

where **ctrpt:** Center point, double precision vector
radius: Radius, double precision number
StartAng: Arc starting angle in radians, double precision
EndAng: Arc ending angle in radians, double precision

Examples:

1. **ThisDrawing.ModelSpace.AddArc point1, 4#, 0#, 1.570796327**
2. **ThisDrawing.ModelSpace.AddArc(cnt1, rad, ang1, ang2)**
 The above code draws an arc whose center point is cnt1, radius is rad, start angle is ang1, and end angle is ang2 and all are predefined in the program.
3. **ThisDrawing.ModelSpace.AddArc(0,0,0, 8, 3.4, 1.453)**

Using the AddText Method to Write Text

The **AddText** method requires a predefined string, insertion point and text height. The **AddText** method syntax is:

ThisDrawing.ModelSpace.AddText textString\$, point1, textHeight

where **textString\$:** Actual text to be displayed
point1: Position point, double precision vector
textHeight: Text Height, positive double precision number

Example

ThisDrawing.ModelSpace.AddText “.063 TYP, 4 PLACES”, point1, 0.25#

FINDING HELP ON METHODS AND PROPERTIES

You can find excellent help in the VBA **Integrated Development Environment** with good examples showing the exact syntax necessary for the **AddLine**, **AddCircle**, or any of the particular methods or properties needed to write a parametric program. To access help from the VBA IDE, choose **Help > Microsoft Visual Basic Help**.

You can also find general information such as on Methods, help on Visual Basic key words, and many non-AutoCAD VBA programming examples taken from Excel, Word, or PowerPoint in AutoCAD 2004. The topics under “AutoCAD 2004 Help” from the Help pull-down menu include “**ActiveX and VBA Developer’s Guide**,” shown in Figure 34-7.

Help is available through a shortcut from the AutoCAD drawing editor’s help system or from the Visual Basic Editor Integrated Design Environment (IDE). Look under VBA and ActiveX Automation. The **Index** and **Find** tabs of this help menu are very useful. Another way to get help in the VBA IDE, is to use the **Object Browser** that can be accessed from the IDE **View** pull-down menu. Select **AutoCAD** in the drop-down list. Select object **AcadModelSpace** in the left window, as shown in Figure 34-8. The question mark help

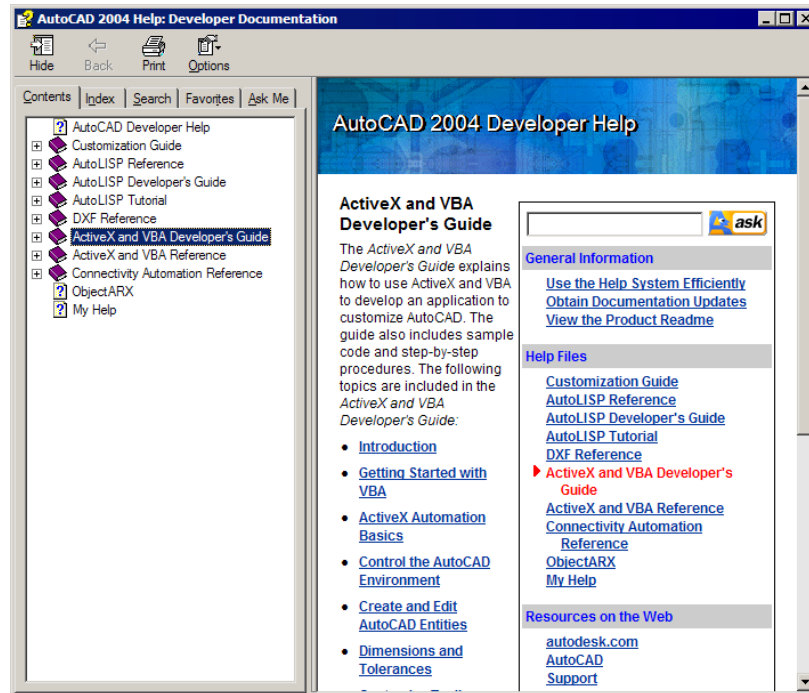


Figure 34-7 ActiveX and VBA Developer's Guide

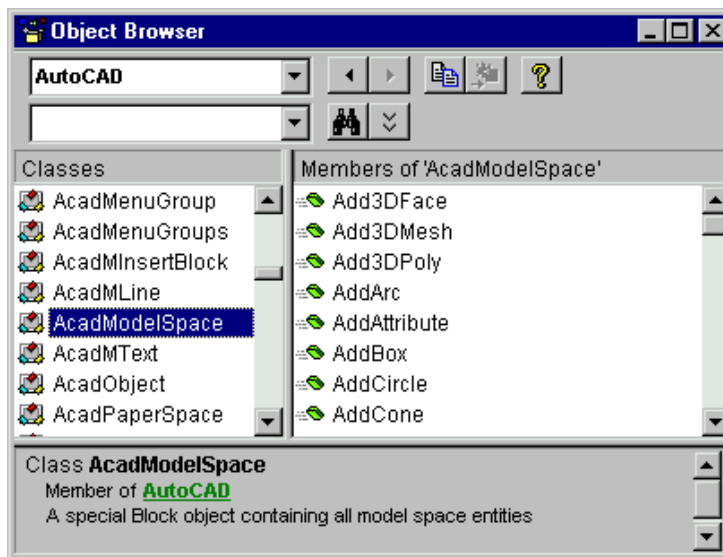


Figure 34-8 Object Browser

button on the toolbar or **F1** takes you to a screen where you can access the ModelSpace Collection of help screens on methods, properties, and examples.

LOADING AND SAVING VBA PROJECTS

To start or load a VBA project in the AutoCAD Drawing Editor, choose **Tools > Macro > Visual Basic Editor** from the menu bar, see Figure 34-9.

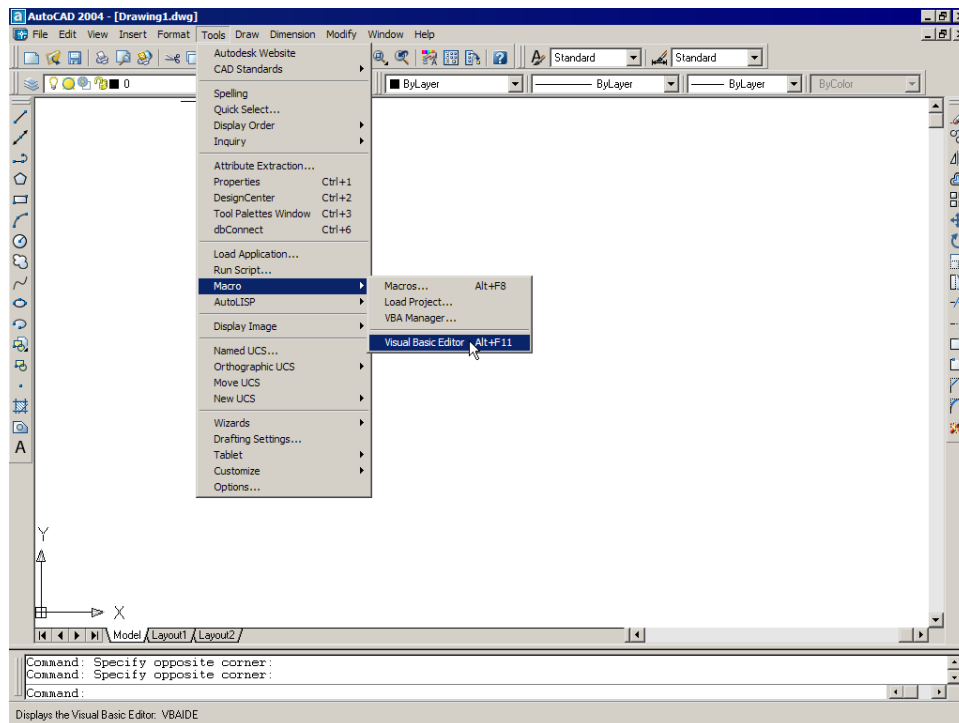


Figure 34-9 Starting the Visual Basic Editor using the **Tools** menu

You will enter the **Integrated Development Environment (IDE)**. The IDE consists of a number of useful windows that can be sized, shown, hidden, or otherwise customized to suit your needs. These windows were discussed earlier in this chapter.

It is a good practice to save your VBA program before testing it. To save a project from the VBA IDE, choose **Save** from the **File** menu. The menu items may be more quickly accessed with the combination of the ALT key and the underlined letter in the menu. AutoCAD VBA projects have the file extension *.dwb*.

Example 1

Write a program that will draw a circle centered at 5,5,0 with radius 2, as shown in Figure 34-10. The User Interface Form is shown in Figure 34-11.

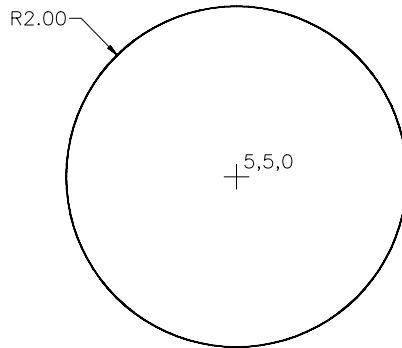


Figure 34-10 Circle for Example 1

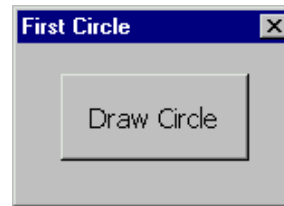


Figure 34-11 User form

1. Start AutoCAD and open a new file using the imperial setting of **Start from Scratch** option.
2. Choose **Tools > Macro > Visual Basic Editor** from the menu bar.
3. Choose **Insert > Module** from the menu bar. If the text editor covers the entire screen, reduce the size by clicking the **Restore Window** button on the top right corner.
4. Insert a User form by choosing **Insert > UserForm** option from the menu bar. The default names will be Module1 and UserForm1.
5. Click the form after insertion to make it the active window. To insert a control such as a command button onto the form, choose the corresponding button on the toolbox. The symbol shown as the second button from the left in the second row of the toolbox in Figure 34-12 inserts the command button control.
5. Now, resize the **UserForm** and **Command Button** to the desired size and edit the caption on the button. This can be done using the **Properties** window at the bottom left corner of screen. The name of the button should be **Draw Circle**. Edit the caption of User form as **First Circle** using the **Properties** window.

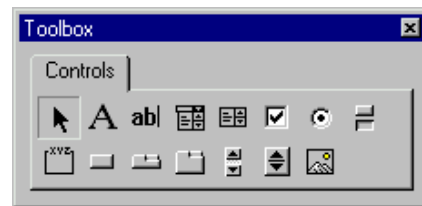


Figure 34-12 The **Toolbox** for creating the User Interface Form

A **project** is the name given to the forms, controls, modules, and programming that makes up a Visual Basic program. To finish a project it is necessary to write the code underlying the visual control(s). One way to get to the code window for the command button control is to double-click on the button and enter the code for `CommandButton1_Click()`. Remember that Visual Basic is event driven. Lines 1 through 17 correspond to the code that runs when the

command button is chosen.

The following is a listing of the Visual Basic program for Example 1. Enter the code given below between the lines where the cursor is blinking. The line code in line number 4 will be available in the code window by default therefore you need not type it again. **The line numbers at the right are for reference only and are not a part of the programming.**

Command Button Code

'UserForm1 Code to Draw Circle, Radius 2	1
'Centered at 5,5,0. Trigger is mouse click on	2
'command button marked Draw Circle	3
Private Sub CommandButton1_Click()	4
Dim CenterPoint(0 To 2) As Double	5
Dim Radius As Double	6
'Data	7
CenterPoint(0) = 5	8
CenterPoint(1) = 5	9
CenterPoint(2) = 0	10
Radius = 2	11
	12
'OLE Automation Object Call	13
ThisDrawing.ModelSpace.AddCircle CenterPoint, Radius	14
Unload Me	15
End Sub	16

Explanation

Lines 1 to 3

The first lines are comments or remarks describing the function of the program. Comments make understanding and modifying a program easier and should be used frequently. Comments start with a Rem or use an apostrophe ('). These lines are ignored when the program is run.

Line 4

Private Sub CommandButton1_Click()

This line defines where **Sub CommandButton1_Click()** starts. The subroutine is executed when **CommandButton1** is chosen. It is the code that draws the circle when the command button is chosen. There is no need to type this line because VBA generates it automatically as soon as the command button control is added to the form.

Lines 5 and 6

Dim CenterPoint(0 To 2) As Double

Dim Radius As Double

These two lines are necessary to establish the double precision variable type for the arguments needed by the AddCircle method.

Lines 8 through 11

CenterPoint(0) = 5

CenterPoint(1) = 5

CenterPoint(2) = 0

Radius = 2

Here the circle center and radius are assigned values. The center point is defined by X, Y, and Z coordinate values.

Line 14

ThisDrawing.ModelSpace.AddCircle CenterPoint, Radius

This line applies the **AddCircle** method to the **ModelSpace** object, which is part of the **ThisDrawing** object. Note a blank space between the keyword **AddCircle** and the first argument, **CenterPoint** is given.



Note

Keywords are words that have been assigned distinct meanings by the programming language. They are reserved and cannot be used as user-defined variables.

Line 15

Unload Me

This is a method that removes **UserForm1** from memory and returns the focus back to AutoCAD.

Line 16

End Sub

Like line 4, this line is generated automatically.

Module1 Code

The following code is entered in the **Module1** code window.

'Module 1 General Declarations	1
Sub DrawCircle()	2
UserForm1.Show	3
End Sub	4

Explanation

Line 2 through 4

Sub DrawCircle()

UserForm1.Show

End Sub

This subroutine's function is to create a Macro name, **DrawCircle**, which appears in the **Macro name** edit box of the **Macros** dialog box invoked by choosing **Macro > Macros** from the **Tools** menu in the main AutoCAD window. These four lines of code belong to the **Module1** object. Another way to run a project is to choose **Run Sub/UserForm** from the **Run** menu in the VBA IDE.

6. Choose the **Run Macro** button from the **Standard** toolbar to execute the program.

GETTING USER INPUT

The successful execution of a program depends on the input from the end user. For this purpose, VBA has various **Get** methods that can be used in the program in order to get the input from the user.

GetPoint Method

The **GetPoint** method allows you to enter the X, Y coordinates or X, Y, Z coordinates of a point. The coordinates of the point can be entered using the keyboard or using pointing device. When you enter the point parameter, it draws a line from the specified point to the current position of the cursor.

The format of the **GetPoint** method is:

P = ThisDrawing.Utility.GetPoint([Point], [Prompt])

Enter a point from the keyboard or select a point in the AutoCAD graphics editor

where **[Point]:** Optional reference point, rubber-band origin

[Prompt]: Optional prompt to be displayed on screen

Examples:

1. **pnt1=ThisDrawing.Utility.GetPoint(“Enter 1st Point”)**

This code generates a prompt that asks you to enter the 1st Point. The value of this point is assigned to pnt1.

2. **Pt2=ThisDrawing.Utility.GetPoint(Pnt1,“Enter 2nd Point”)**

This code generates a prompt that asks you to enter the 2nd point. The value of this point is assigned to Pt2.

GetDistance Method

The **GetDistance** method lets you enter a distance on the command line, a distance from a given point, or two points, and it then returns the distance as a double precision number. Use this method after the **GetPoint** method. The format of the **GetDistance** method is given next.

d = ThisDrawing.Utility.GetDistance([point], [prompt])

where **[point]:** Optional reference point, rubber band origin

[prompt]: Optional prompt to be displayed on screen

Examples:

1. **d1 = ThisDrawing.Utility.GetDistance(cen, “specify radius”)**

This code generates a prompt and allows you to enter a point with respect to an existing point. The value of distance between the two points is assigned to d1.

2. `x = ThisDrawing.Utility.GetDistance(p1, "specify the height")`

This code generates a prompt and allows you to enter a point with respect to an existing point. The value of distance between the two points is assigned to x.

GetAngle Method

The **GetAngle** method allows you to enter an angle, either using the keyboard in degrees or by selecting two points. In the case of selecting points, the positive horizontal direction is taken as one leg of the angle, the first point selected as the vertex and the second point defines the second leg. If the point argument is specified, AutoCAD uses this point as the first point or angle vertex. The **GetAngle** method returns the value of the angle in radians as a double precision value. The format of the **GetAngle** method is given next.

```
ang = ThisDrawing.Utility.GetAngle([point], [prompt])
```

where	ang:	Angle in radians
	[point]:	Optional vertex point
	[prompt]:	Optional screen prompt to clarify angle selection

Examples

1. `a1 = ThisDrawing.Utility.GetAngle(, "Enter taper angle in degrees")`

This code generates a prompt and allows you to enter a point with respect to an existing point. It assigns the value of the angle between the two points to a1.

2. `ang3 = ThisDrawing.Utility.GetAngle(p1, "Specify the center point")`

This code generates a prompt and allows you to enter a point with respect to an existing point. It assigns the value of the angle between the two points to ang3.



Note

*The angle you enter is affected by the angle setting. The angle settings can be changed by changing the value of the **ANGBASE** and **ANGDIR** system variables. The default settings for measuring an angle are as follows:*

*The angle is measured with respect to the positive X axis (3 o'clock position). The value for 3 o'clock corresponds to the current value of the **ANGBASE** system variable, which is 0. **ANGBASE** could be set in any of four 90-degree quadrant directions*

*The angle is positive if it is measured in the counterclockwise direction and negative if it is measured in the clockwise direction. The value of this setting is saved in the **ANGDIR** system variable. The **GetOrientation** method has the same syntax as the **GetAngle** method but ignores the **ANGBASE** and **ANGDIR** system variables. The 0 angle is always at 3 o'clock, and angles are always positive counterclockwise.*

Example 2

Write a program that will draw a triangle with user supplied vertices P1, P2, and P3 as in Figure 34-13. This program is to use the **GetPoint** and **AddLine** methods. The User Interface Form for this example is shown in Figure 34-14.

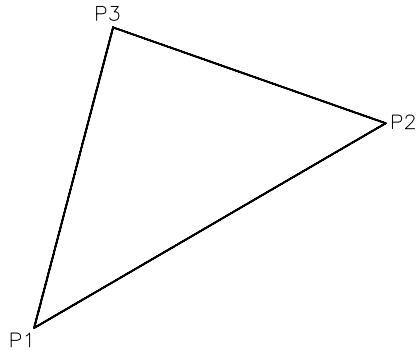


Figure 34-13 Triangle with user-defined points

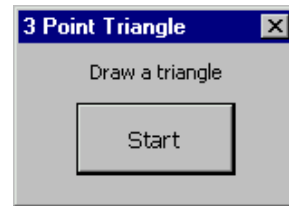


Figure 34-14 User Interface Form for Example 2

1. Start AutoCAD and open a new file using the imperial setting of **Start from Scratch** option.
2. Choose **Tools > Macro > Visual Basic Editor** from the menu bar.
3. Choose **Insert > Module** from the menu bar.
4. Choose **UserForm** from the **Insert** menu to insert a User form.
5. Drag and insert the **Command** button from the toolbar to the UserForm with the mouse to produce an interface form similar to Figure 34-14.
6. Double-click the command button to open the code window and type the code given below.

The following file is a listing of the project for Example 2. **The line numbers at the right are for reference only and are not a part of the program.**

Command Button Code

```

'The function of this routine is to draw          1
'a triangle from 3 user specified points         2
'The trigger is a mouse click on the command    3
'button labeled Start.                          4
'pnt1, pnt2, pnt3 are variant by default        5
'Returns a point in WCS                         6
Private Sub CommandButton1_Click()              7
UserForm1.Hide                                 8
pnt1 = ThisDrawing.Utility.GetPoint(, "Provide the First Point: ") 9
'Returns a variant vector since GetPoint returns a point in WCS 10
'And draws a rubber-band line from the optional first point      11
pnt2 = ThisDrawing.Utility.GetPoint(pnt1, "Second Point? ")      12

```

'Draw first side of triangle	13
ThisDrawing.ModelSpace.AddLine pnt1, pnt2	14
pnt3 = ThisDrawing.Utility.GetPoint(pnt2, "3rd Point? ")	15
ThisDrawing.ModelSpace.AddLine pnt2, pnt3	16
ThisDrawing.ModelSpace.AddLine pnt3, pnt1	17
Unload Me	18
End Sub	19

Explanation

Lines 1-6, 10, 11, and 13

These lines are comments or remarks describing the function of the following line or lines. Comments make understanding and modifying a program easier and should be used liberally. Comments start with a Rem or use an apostrophe ('). These lines are ignored when the program is run.

Line 7

Private Sub CommandButton1_Click()

This line defines where Sub CommandButton1_Click() starts. The subroutine code is executed when **CommandButton1** is chosen.

Line 8

UserForm1.Hide

This statement hides the user interface form and returns focus to AutoCAD. If UserForm from the previous example was not loaded, the UserForm for this example will be **UserForm1** by default. In this event you should change the code in line 8 to **Userform1.Hide**.

Line 9, 12 and 15

pnt1 = ThisDrawing.Utility.GetPoint(, "Provide the First Point:")

pnt2 = ThisDrawing.Utility.GetPoint(pnt1, "Second Point?")

pnt3 = ThisDrawing.Utility.GetPoint(pnt2, "3rd Point?")

The **GetPoint** method is used without a reference point in line 15. The point may be specified either with the keyboard or mouse. When it is used with a reference point we see a rubber-band line attached from the reference point to the mouse cursor.

Lines 14, 16, and 17

ThisDrawing.ModelSpace.AddLine pnt1, pnt2

ThisDrawing.ModelSpace.AddLine pnt2, pnt3

ThisDrawing.ModelSpace.AddLine pnt3, pnt1

The **AddLine** method requires the two point arguments to be double precision vectors with 3 components. Note the required space before the first point.

Line 18 and 19

Unload Me

End Sub

These lines remove UserForm1 from memory, return the focus back to AutoCAD, and end the subroutine.

Module2 Code

The following code is entered in the Module2 code window.

Option Explicit	1
Sub Triangle()	2
UserForm1.Show	3
End Sub	4

Explanation

Line 1

Option Explicit

The inclusion of this line forces explicit declaration of all variable types. This minimizes common inconsistent variable usage errors and typographic errors as they are quickly caught at run time. Otherwise, undeclared variable types would be Variant by default.

Lines 2-4

Sub Triangle()

UserForm1.Show

End Sub

This subroutine's function is to create a Macro name, Triangle, that can be run from AutoCAD by choosing **Macro > Macros** from the **Tools** menu. Lines 1-4 are a part of Module1. The remaining lines of code below are a part of the UserForm2 object. Again, if UserForm1 from the previous example are not still loaded, the UserForm for this example would be named UserForm1 and Module1 similarly would be named Module1, by default. In this event, you should change the names in lines 2-4 accordingly.

Example 3

Create a red colored solid cylinder by specifying its radius and height.

1. Start AutoCAD and open a new file using the imperial setting of **Start from Scratch** option.
2. Choose **Tools > Macro > Visual Basic Editor** from the menu bar.
3. Choose **Insert > Module** from the menu bar. If the text editor covers the entire screen, reduce the size by clicking the **Restore Window** button on the top right corner.
3. Click the **Properties Window** button from the toolbar.
4. Change the Name to CreateCylinder in the **Properties** window.
5. Close the **Properties** window and maximize the CreateCylinder (Code) window if necessary.

6. Type the following code in the CreateCylinder (Code) window:

```
Public Sub DrawCylinder()
Dim cen As Variant
Dim r As Double
Dim h As Double
Dim cyl As Acad3DSolid
cen = ThisDrawing.Utility.GetPoint(, "Specify center point: ")
r = ThisDrawing.Utility.GetDistance(cen, "Specify radius: ")
h = ThisDrawing.Utility.GetDistance(, "Specify height: ")
Set cyl = ThisDrawing.ModelSpace.AddCylinder(cen, rad, ht)
ThisDrawing.SendCommand ("VPOINT -1,-1,1 SHADEMODE GOURAUD ")
ThisDrawing.SendCommand ("CHPROP ")
ThisDrawing.SendCommand ("LAST ")
ThisDrawing.SendCommand ("C RED ")
ThisDrawing.SendCommand ("UCSICON ")
ThisDrawing.SendCommand ("NOO ")
End Sub
```

7. Choose the **Save** button on the **Standard** toolbar. The **Save** dialog box appears. Save the code as the *cylinder.dvb* file.
8. Exit the VBA environment by choosing the View AutoCAD button from the toolbar.
9. Choose **Tools > Macro > Macros** from the menu bar. The **Macros** dialog box appears.
10. The **Macro name** text box of the **Macro** dialog box shows the name of the project and its location.
11. Choose the **Run** button.
12. Follow the prompt sequences in the Command prompt and create the cylinder. You can use the keyboard to enter the values. You can also specify the values using two points on the screen.

VBA creates a shaded cylinder of red color. The viewpoint also automatically changes to the SW isometric view.

POLARPOINT AND ANGLEFROMXAXIS METHODS

PolarPoint Method

The **PolarPoint** method defines a point at a given angle and distance from a given point. It has the syntax:

P = ThisDrawing.Utility.PolarPoint(Point, Angle, Distance)

where	Point:	Reference point, rubber-band origin
	Angle:	Angle in radians, double precision

Distance: Distance from point, double precision

AngleFromXAxis Method

The **AngleFromXAxis method** calculates the angle of a line defined by two points from the horizontal axis in radians. The format of the **AngleFromXAxis method** is:

ang = ThisDrawing.Utility.AngleFromXAxis(point1, point2)

where **point1:** Start point of the line

point2: End point of the line

Exercise 1

Write a Visual Basic program that will draw a line between two points, as in Figure 34-15. The user may either enter coordinates or choose points with the mouse. The graphical screen should draw a rubber-band on the screen to the current position of the mouse cursor if the second point is entered with the mouse. Use a graphical user interface form similar to the one shown in Figure 34-16.

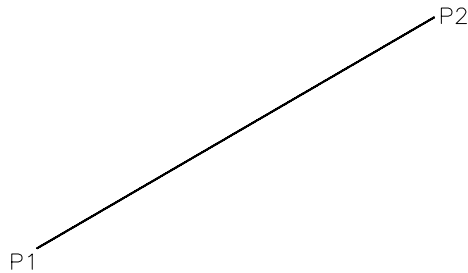


Figure 34-15 Line with user-defined end

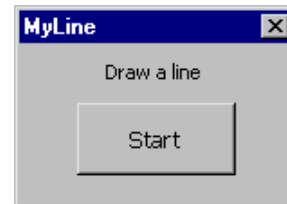


Figure 34-16 User Interface Form for Exercise

Example 4

Write a Visual Basic program that will draw a triangle based on a given line produced from two points P1 and P2, on an included angle and on a length of the second side shown in Figure 34-17. To create the **Text Boxes** on the user interface form, Figure 34-18, use the **TextBox** button. The labels and command button on the form are created by clicking or dragging the respective icons from the toolbox.

1. Create a new drawing file and enter the VBA IDE.
2. Insert the User form and module from the **Insert** menu.
3. Create the User form as shown in Figure 34-18 and double-click on the **Draw SAS Triangle**

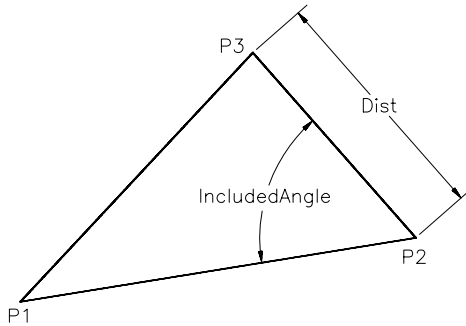


Figure 34-17 Side Angle Side Triangle

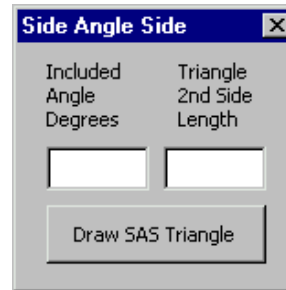


Figure 34-18 User Interface Form for Example 3

button to open the code window. Type the code that is given next.

The following file is a listing of the Visual Basic program for Example 4. **The line numbers at the right are for reference only and are not a part of the program.**

Command Button Code

```

(Declarations) (General)                                     1
Const PI = 3.141592654                                       2
Public IncludedAngle As Double 'pi-converted input angle     3
Public Angle As Double 'included angle                       4
Public Dist As Double 'length of 2nd side                    5
                                                                6
'This procedure draws a triangle from 2 sides and an included angle SAS 7
'Program trigger is the command button labeled "Draw SAS Triangle" 8
'Additional feature is use of text boxes for included angle, 2nd side length
'input                                                         9
'Included angle is angle between base and 2nd side            10
                                                                11
Private Sub CommandButton1_Click()                             12
UserForm1.Hide                                                13
'p1 is 1st point of base, variant type by default            14
'p2 is 2nd point of base, variant type by default            15
p1 = ThisDrawing.Utility.GetPoint(, "Enter or select 1st base point:") 16
p2 = ThisDrawing.Utility.GetPoint(p1, "Enter select 2nd base point:") 17
ThisDrawing.ModelSpace.AddLine p1, p2 'draw base line        18
If TextBox1.Text = "" Then                                     19
Angle = ThisDrawing.Utility.GetAngle(p2, "Enter angle from horiz.
or select included angle:")                                   20
Else                                                           21
Angle = ThisDrawing.Utility.AngleFromXAxis(p1, p2) + IncludedAngle 22
End If                                                         23

```



```

If TextBox2.Text = "" Then                24
    Dist = ThisDrawing.Utility.GetDistance(p2, "Enter or select dist.
    from base point:")                  25
End If                                    26
p3 = ThisDrawing.Utility.PolarPoint(p2, Angle, Dist) 27
ThisDrawing.ModelSpace.AddLine p2, p3      28
ThisDrawing.ModelSpace.AddLine p1, p3      29
Unload Me                                  30
End Sub                                    31

Private Sub textBox1_Change()              32
    IncludedAngle = PI - Val(TextBox1.Text) * PI / 180 33
End Sub                                    34

Private Sub textBox2_Change()              35
    Dist = Val(TextBox2.Text)              36
End Sub                                    37

```

Explanation

Line 2

Const PI = 3.141592654

The constant pi used in this routine is declared to demonstrate this type of statement.

Lines 3-5

Public IncludedAngle As Double

Public Angle As Double

Public Dist As Double

These variables are declared to be public, which means any procedure can access them from any module in the project (file) without passing the argument in a parameter list. Also three variables are declared to be double precision as required by the AutoCAD methods in which they will be employed.

Lines 7-12

‘This module draws a triangle from 2 sides and an included angle SAS

‘Program trigger is the command button labeled “Draw SAS Triangle”

‘Additional feature is use of text box for included angle, 2nd side length input

‘Included angle is angle between base and 2nd side

Private Sub CommandButton1_Click()

These lines give the purpose and trigger of the procedure **CommandButton1_Click**, which starts with line 8. Purpose and trigger remarks are useful for all event-driven subroutines.

Line 13

UserForm1.Hide

This statement hides the screen interface form and returns focus to AutoCAD. Otherwise the

user would have to close the form with the button at the top right corner of the window to proceed with entering input in AutoCAD.

Lines 14-17

'p1 is 1st point of base, variant type by default

'p2 is 2nd point of base, variant type by default

p1 = ThisDrawing.Utility.GetPoint(, "Enter or select 1st base point:")

p2 = ThisDrawing.Utility.GetPoint(p1, "Enter or select 2nd base point:")

These lines get input from the AutoCAD graphic screen or command line for the two points defining the base side of the triangle.

Lines 19-23 and 33-35

If textBox1.Text = "" Then

Angle=ThisDrawing.Utility.GetAngle(p2,"Enter angle from Horiz. or select included angle:")

Else

Angle=ThisDrawing.Utility.AngleFromXAxis(P1,P2)+IncludedAngle

End If

Private Sub textBox1_Change()

IncludedAngle = PI - Val(textBox1.Text) * PI / 180 'in radians

End Sub

The If ... Then ... Else statement checks TextBox1 for an entry. If no angle has been entered on the UserForm, the **GetAngle** method is used. If a numeric angle was entered on the form it is converted by the subroutine textBox1_Change() to a radian-included angle. Adding the radian angle returned by the **AngleFromXAxis** method gives the AutoCAD polar angle for the vertex (third) point.

Lines 24-26

If textBox2.Text = "" Then

Dist = ThisDrawing.Utility.GetDistance(p2, "Enter or select dist. from base point:")

End If

If no numeric entry for the second side length was entered on the user form, these lines use the **GetDistance** method.

Lines 37-39

Private Sub textBox2_Change()

Dist = Val(textBox2.Text)

End Sub

If a numeric entry for the second side length was entered on the user form, these lines convert from text to numeric form.

Lines 18, 28 and 29

ThisDrawing.ModelSpace.AddLine p1, p2

ThisDrawing.ModelSpace.AddLine p2, p3

ThisDrawing.ModelSpace.AddLine p1, p3

These methods draw the base, second, and third sides of the triangle.

Module3 Code

The following code is entered in the Module3 code window.

Option Explicit	1
Sub SAS()	2
UserForm1.Show	3
End Sub	4

Exercise 2

Write a program in the AutoCAD VBA IDE that will draw a triangle based on a given line produced from two points P1 and P2, on an adjacent angle at one end, and another angle at the other end as shown in Figure 34-19. Employ a UserForm similar to the one shown in Figure 34-20.

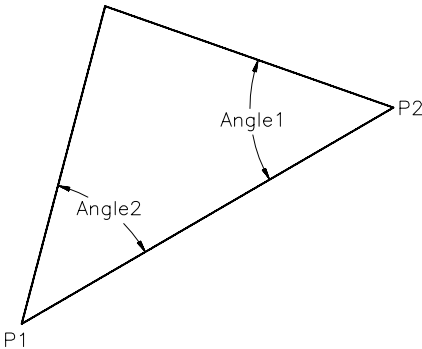


Figure 34-19 Angle Side Angle Triangle with user-defined points defining the base

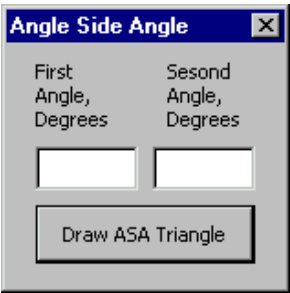


Figure 34-20 User Interface Form for Exercise 2

Exercise 3

Write a program in the AutoCAD VBA IDE that will draw a triangle based on a given line produced from two points P1 and P2, and two distances, which are the lengths of the sides adjoining each end as in Figure 34-21. Employ a UserForm similar to the one shown in Figure 34-22.

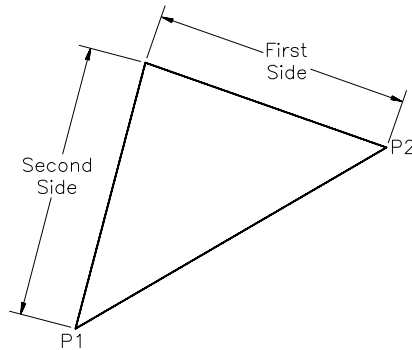


Figure 34-21 Side Side Triangle with user-defined points defining the base

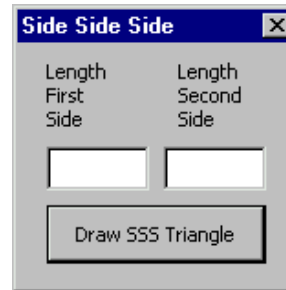


Figure 34-22 User Interface Form for Exercise 3

ADDITIONAL VBA EXAMPLES

More VBA examples are included with AutoCAD 2004 beyond the large number in the help files. These are considered sample files and are located in the **Sample/VBA** subdirectory of the AutoCAD 2004 directory and the AutoCAD 2004 CD ROM.

Self-Evaluation Test

Answer the following questions and then compare your answers to the answers given at the end of this chapter.

1. VBA in AutoCAD has a separate environment in which you can do all the work related to a project. This environment is called Integrated Development Environment (IDE). (T/F)
2. To invoke the VB editor, choose **Tools > Macros > Visual Basic Editor** from the menu bar. (T/F)
3. The primary purpose of VBA is to control the functionality of the application and automate it with VBA commands. (T/F)
4. The application that you are working on is called a **Document object**. (T/F)
5. Examples of objects include drawings or documents, geometric elements such as lines or circles, and user interface controls to handle input and output to programs or macros. (T/F)

6. Which one of the following methods is used to draw a line?
- (a) **AddArc** (b) **AddLine**
(c) **AddCircle** (d) None
7. Which one of the following methods allow the user to specify the distance on the screen?
- (a) **GetPoint** (b) **GetDistance**
(c) **GetAngle** (d) None
8. Which of the following objects come first in the hierarchy?
- (a) ActiveDocument (b) AddMethod
(c) Application (d) None
9. _____ notation is required to clarify on which object the method is acting.
- (a) Star (b) Dot
(c) Blank (d) None
10. The environment that is provided in AutoCAD to work with VB is called:
- (a) Integrated Aided Environment (b) Integrated Using Environment
(c) Integrated Design Environment (d) None

Review Questions

Answer the following questions.

- _____ lets you access and manipulate the objects and functionality of AutoCAD from Excel or another application supporting ActiveX.
- Before you can use AutoCAD's objects in some other software supporting ActiveX, you must make the application aware that the AutoCAD _____ Library is available on the computer.
- There are _____ programmers using Visual Basic.
- Visual Basic is considered an object- _____ , rather than an object-oriented, programming language.
- Functions known as _____ have been defined in the AutoCAD object library to perform an action on an object, for example, drawing a line in a drawing.
- _____ are functions that set or return information about the state of an object.

7. The term **IDE**, which is the Visual Basic Editor, stands for _____.
8. A _____ (extension *.dxb*) is the name given to the forms, controls, modules, and programming making up a saved AutoCAD Visual Basic file.
9. _____ forces explicit declaration of all variable types, which minimizes common inconsistent variable usage errors.
10. _____ is the variable type returned by the **GetPoint** method.
11. The _____ method always measures the angle with a positive *X* axis (3 o'clock position) and in a counterclockwise direction.
12. The _____ method allows you to enter the *X, Y* coordinates or *X, Y, Z* coordinates of a point.
13. The _____ method lets you retrieve the value of an AutoCAD system variable.
14. The _____ method defines a point at a given angle and distance from the given point.
15. The _____ method lets you enter a distance on the command line, a distance from a given point, or two points.
16. The _____ method allows you to enter an angle, either from the keyboard in degrees or by selecting two points.

Exercises

Exercise 4

Write a Visual Basic program that will draw an equilateral triangle inside the circle (Figure 34-23).

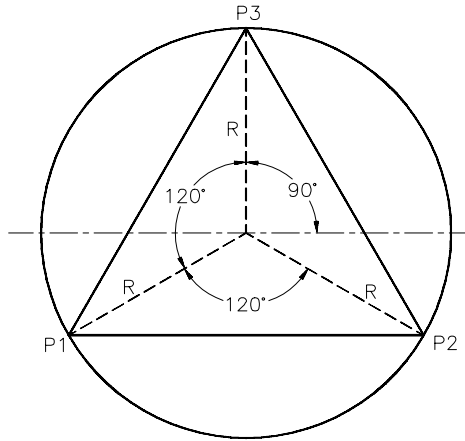


Figure 34-23 Equilateral triangle inside a circle

Exercise 5

Write a Visual Basic program that will draw a square of sides S and a circle tangent to the four sides of the square as shown in Figure 34-24. The base of the square makes an angle, ANG , with the positive X axis. The program should allow you to enter the starting point $P1$, length S , and angle ANG on a user interface form or in the AutoCAD graphical interface.

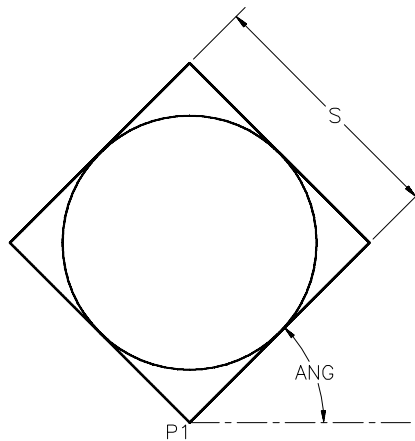


Figure 34-24 Square of side S and at an angle ANG

Exercise 6

Write a Visual Basic program that will draw a line at an angle A and then generate the given

number of lines (N), parallel to the first line with an offset distance S (Figure 34-25) as entered on a user form or from the keyboard.

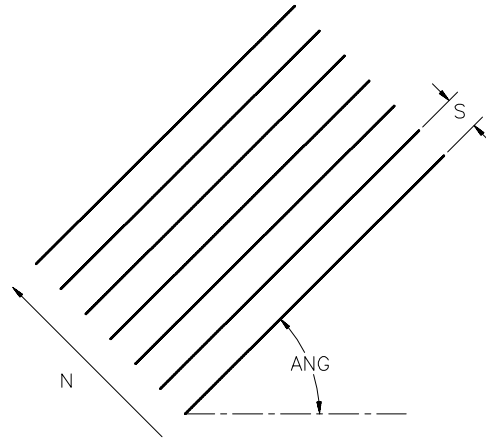


Figure 34-25 N number of lines offset at a distance of S

Exercise 7

Write a program that will draw a slot shown in Figure 34-26 with center lines. The program should allow you to enter slot length, slot width, and the layer name for center lines on a user interface form or in the AutoCAD graphical interface.

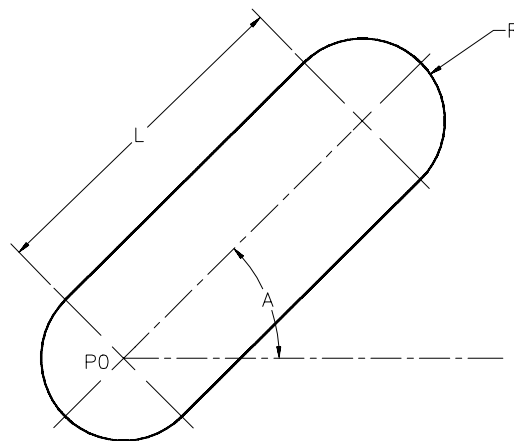


Figure 34-26 Slot of length L and radius R

Exercise 8

Write a Visual Basic program that will draw two lines tangent to two circles, as shown in Figure 34-27. The program should allow you to enter the circle diameters and the center distance between the circles on a user interface form or in the AutoCAD graphical interface.

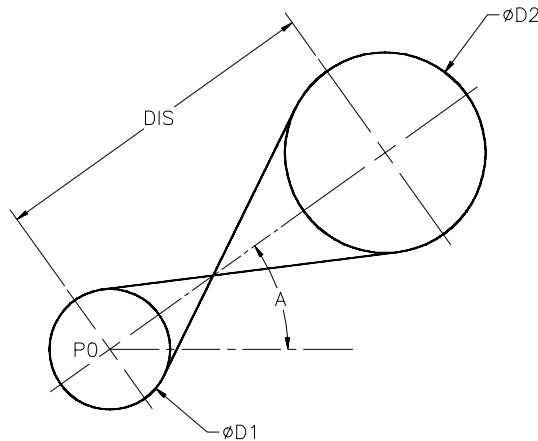


Figure 34-27 Circles with tangent lines

Exercise 9

Write a VBA program that will draw a hub with the key slot, as shown in Figure 34-28. The user should enter the four parameters in the note below on a user form or in the AutoCAD graphical interface. Use the program inside a circle of Diameter D1 to produce a keyed bushing with the test dimensions.

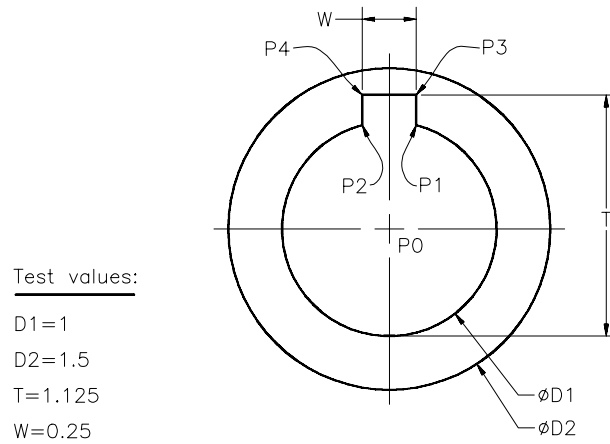


Figure 34-28 Hub with Keyway

Exercise 10

Write a VBA program to draw the tangent arc cam shown in Figure 34-29.

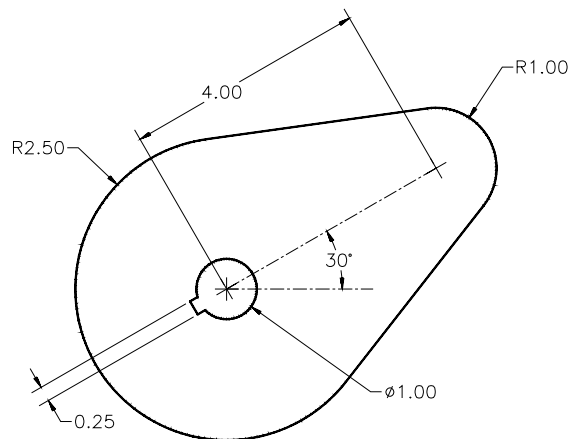


Figure 34-29 Tangent Arc Flat Plate Cam

Exercise 11

Write a VBA program to draw the circular arc cam shown in Figure 34-30.

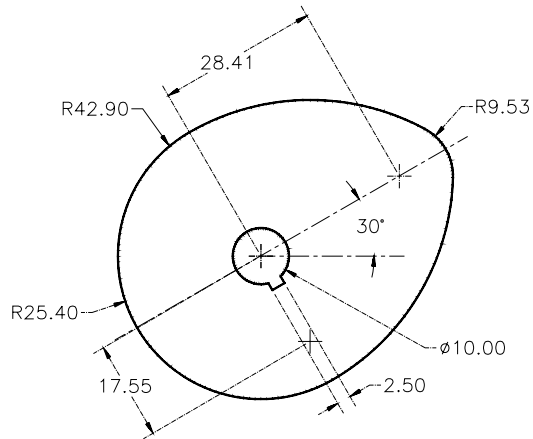


Figure 34-30 Circular Arc Cam

Project Exercise 1

Write a Visual Basic program that will draw the two views of a bushing as shown in Figure 34-31. The program should allow you to enter the starting point P0, lengths L1, L2, and the bushing diameters ID, OD, HD on a user interface form or in the AutoCAD graphical interface. The distance between the front view and the side view of bushing is DIS ($DIS = 1.25 * HD$). The program should also draw the hidden lines in the HID layer and center lines in the CEN layer. The center lines should extend 0.75 units beyond the object line.

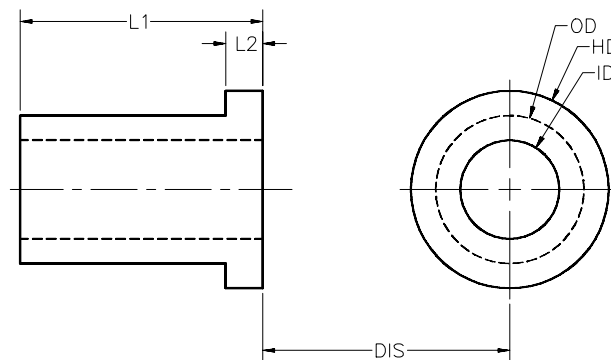


Figure 34-31 Two views of bushing

Project Exercise 2

Draw the following parametric drawing in a rectangle 320mm wide by 200mm high. The block depends on one number, Q , called the data set number. The dimensions as shown in Figure 34-32 are given by: $X = Q + 160$, $Z = 240 - X$, $Y = 120 - Z$, and $R = Z/4$. There are 40mm between views and 20mm between a view and the surrounding rectangle at the closest points. The circle is centered in the block.

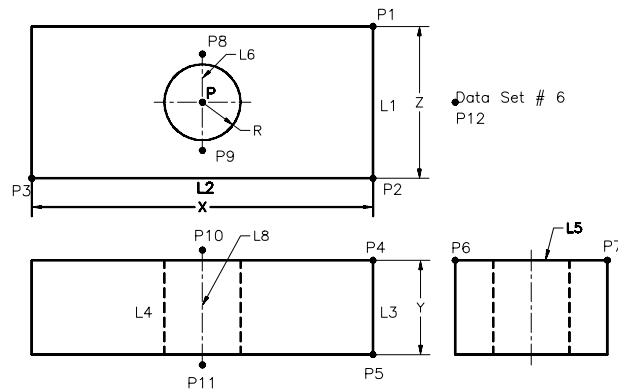


Figure 34-32 Parametric Block with Hole

Answers to the Self-Evaluation Test

1 - T, 2 - T, 3 - T, 4 - F, 5 - T, 6 - b, 7 - b, 8 - c, 9 - b, 10 - c.