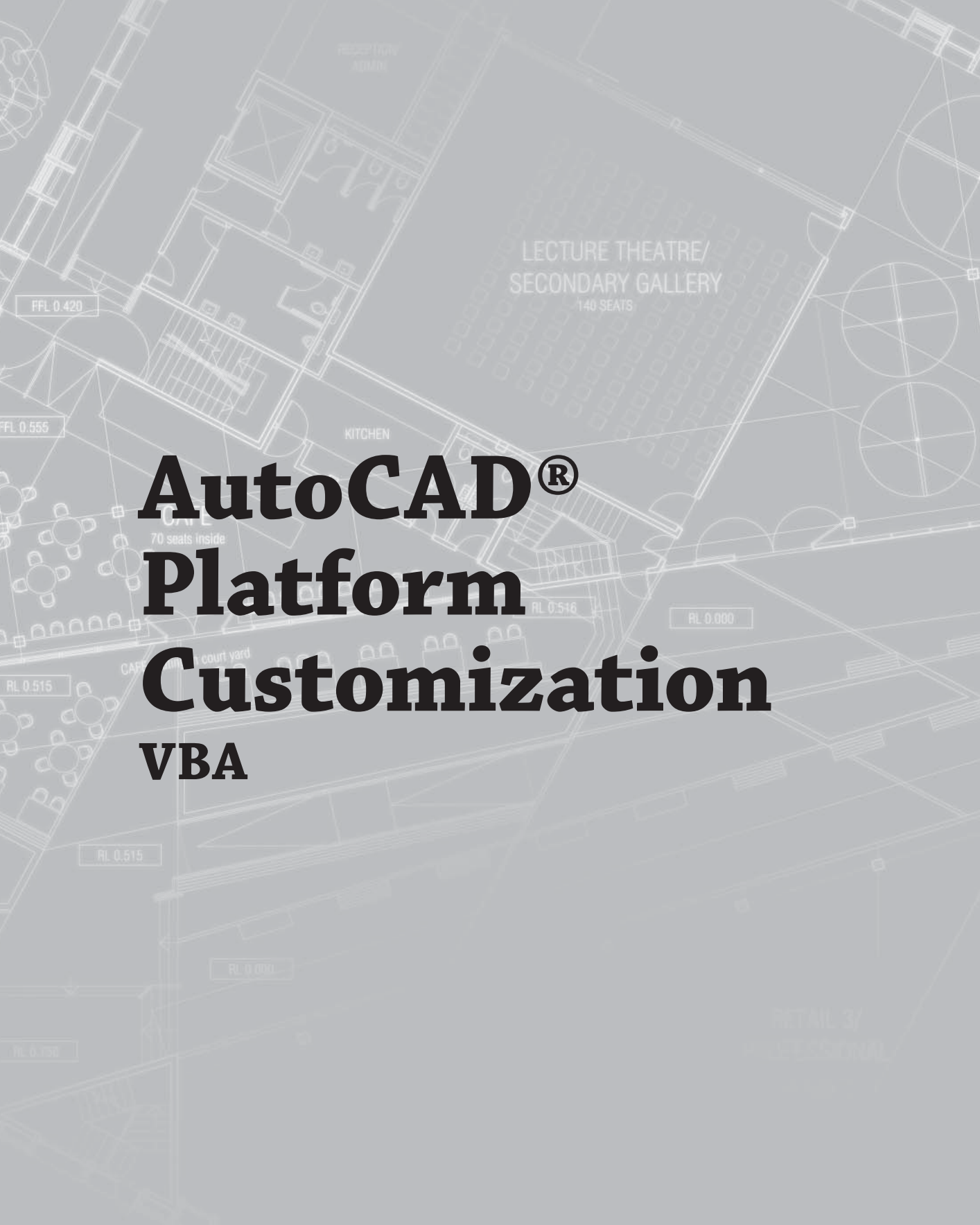

LEE AMBROSIUS
LECTURE THEATRE/
SECONDARY GALLERY
140 SEATS
FFL 0.420
FFL 0.555
KITCHEN
CAFE
70 seats inside
RL 0.516
RL 0.000
CAFE seating in court yard
RL 0.515
AutoCAD® Platform Customization
VBA
SYBEX
A Wiley Brand



AutoCAD® Platform Customization

VBA

AutoCAD® Platform Customization

VBA

Lee Ambrosius

Autodesk®
Official Training Guide

SYBEX®
A Wiley Brand

Senior Acquisitions Editor: Stephanie McComb
Development Editor: Mary Ellen Schutz
Technical Editor: Richard Lawrence
Production Editor: Dassi Zeidel
Copy Editor: Liz Welch
Editorial Manager: Pete Gaughan
Production Manager: Kathleen Wisor
Associate Publisher: Jim Minatel
Book Designers: Maureen Forys, Happenstance Type-O-Rama; Judy Fung
Proofreader: Candace Cunningham
Indexer: Ted Laux
Project Coordinator, Cover: Brent Savage
Cover Designer: Wiley
Cover Image: © Smileyjoanne/iStockphoto.com

ISBN: 978-1-118-90044-4 (ebk.)
ISBN: 978-1-118-90698-9 (ebk.)

Library of Congress Control Number: 2015936845

10 9 8 7 6 5 4 3 2 1

To my friend Kathy Enderby: You were one of the first people to encourage me to follow my passion for programming and sharing what I had learned with others. Thank you for believing in me all those years ago and for being there when I needed someone to bounce ideas off—especially during those late-night scrambles right before deploying a new software release.

Acknowledgments

I have to give a very special thanks to all the great folks at Sybex, especially Willem Knibbe, for working on and helping to get this project off the ground after a few years of talking about it. The next two people I would like to thank are Mary Ellen Schutz and Dassi Zeidel, the development and production editors on this book; you two made sure I stayed on track and delivered a high-quality book. I also want to thank Liz Welch (copyeditor), Candace Cunningham (proofreader), and Ted Laux (indexer) for the work you all did on this book.

Thanks to all the folks at Autodesk, who put in the long hours and are dedicated to the work they do on the Autodesk® AutoCAD® product. I cannot forget one of the most important individuals on this book, my technical editor, Richard Lawrence. Richard is a great friend who I met many years ago at Autodesk University. He is a passionate and driven user of AutoCAD and is always looking to improve the way he uses AutoCAD. Richard, I appreciate everything that you have done to make this book better. Congrats on making it through your first book as a technical editor.

About the Author

Lee Ambrosius first started working with AutoCAD R12 for DOS in 1994. As a drafter, he quickly discovered that every project included lots of repetition. Lee, not being one to settle for "this is just the way things are," set out on a path that would redefine his career. This new path would lead him into the wondrous world of customization and programming—which you might catch him referring to as "the rabbit hole."

In 1996, Lee began learning the core concepts of customizing the AutoCAD user interface and AutoLISP. The introduction of VBA in AutoCAD R14 would once again redefine how Lee approached programming solutions for AutoCAD. VBA made it much easier to communicate with external databases and other applications that supported VBA. It transformed the way information could be moved between project-management and manufacturing systems.

Not being content with VBA, in 1999 Lee attended his first Autodesk University and began to learn ObjectARX®. Autodesk University had a lasting impression on him. In 2001, he started helping as a lab assistant. He began presenting on customizing and programming AutoCAD at the event in 2004. Along the way he learned how to use the AutoCAD Managed .NET API.

In 2005, Lee decided cubicle life was no longer for him, so he ventured off into the CAD industry as an independent consultant and programmer with his own company, HyperPics, LLC. After he spent a few years as a consultant, Autodesk invited him to work on the AutoCAD team; he has been on the AutoCAD team since 2007. For most of his career at Autodesk, Lee has worked primarily on customization and end-user documentation. Recently, he has been working on the AutoLISP, VBA, ObjectARX, .NET, and JavaScript programming documentation.

In addition to working on the AutoCAD documentation, Lee has been involved as a technical editor or author for various editions of the *AutoCAD and AutoCAD LT Bible*, *AutoCAD for Dummies*, *AutoCAD & AutoCAD LT All-in-One Desk Reference for Dummies*, *AutoCAD 3D Modeling Workbook for Dummies*, and *Mastering AutoCAD for Mac.* He has also written white papers on customization for Autodesk and a variety of articles on customization and programming for *AUGIWorld*, published by AUGI®.

Contents at a Glance

Contents

Introduction

Welcome to *AutoCAD Platform Customization: VBA*! Have you ever thought to yourself, why doesn't the Autodesk® AutoCAD® program include every feature I need? Why isn't it streamlined for the type of work I perform? If so, you are not alone. AutoCAD at its core is a drafting platform that, through programming, can be shaped and molded to more efficiently complete the tasks you perform on a daily basis and enhance your company's workflows. Take a deep breath. I did just mention programming, but programming isn't something to fear. At first, just the idea of programming makes many people want to run in the opposite direction—myself included. The productivity gains are what propelled me forward. Programming isn't all that different from anything else you've tried doing for the first time.

In many ways, learning to program is much like learning a foreign language. For many new to Visual Basic for Applications (VBA), the starting place is learning the basics: the syntax of the programming language and how to leverage commands and system variables. Executing commands and working with system variables using the `SendCommand` and `PostCommand` methods can be a quick way to get started and become comfortable with VBA. After you are comfortable with the syntax of VBA and the `SendCommand` and `PostCommand` functions, you can begin to learn how to access the AutoCAD Object library to develop more complex and robust programs.

About This Book

AutoCAD Platform Customization: VBA provides you with an understanding of the VBA programming language and how it can be used in combination with the AutoCAD Object library to improve your productivity. This book is designed to be more than just an introduction to VBA and the AutoCAD Object library; it is a resource that can be used time and again when developing VBA programs for use with AutoCAD. As you page through this book, you will notice that it contains sample code and exercises that are based on real-world solutions.

This book is the third and final book in a series that focuses on customizing and programming AutoCAD. The three-book series as a whole is known as *AutoCAD Platform Customization: User Interface, AutoLISP, VBA, and Beyond*, which will be available as a printed book in 2015. Book 1 in the series, *AutoCAD Platform Customization: User Interface and Beyond*, was published in early 2014 and focused on CAD standards and general customization of AutoCAD; Book 2, *AutoCAD Platform Customization: AutoLISP*, was published in mid-2014 and covers the AutoLISP programming language.

Is This Book for You?

AutoCAD Platform Customization: VBA covers many aspects of VBA programming for AutoCAD on Windows. If any of the following are true, this book will be useful to you:

- You want to develop and load custom programs with the VBA programming language for use in the AutoCAD drawing environment.
- You want to automate the creation and manipulation of drawing objects.
- You want to automate repetitive tasks.
- You want to help manage and enforce CAD standards for your company.

NOTE VBA programming isn't supported for AutoCAD on Mac OS.

VBA in AutoCAD

VBA is often overlooked as one of the options available to extend the AutoCAD program. There is no additional software to purchase, but you must download and install a release-specific secondary component to use VBA. You can leverage VBA to perform simple tasks, such as inserting a title block with a specific insertion point, scale, and rotation and placing the block reference on a specific layer. To perform the same tasks manually, end users would have to first set a layer as current, choose the block they want to insert, and specify the properties of the block, which in the case of a title block are almost always the same.

The VBA programming language and AutoCAD Object library can be used to do the following:

- Create and manipulate graphical objects in a drawing, such as lines, circles, and arcs
- Create and manipulate nongraphical objects in a drawing, such as layers, dimension styles, and named views
- Perform mathematical and geometric calculations
- Request input from or display messages to the user at the Command prompt
- Interact with files and directories in the operating system
- Read from and write to external files
- Connect to applications that support ActiveX and COM
- Display user forms and get input from the end user

VBA code statements are entered into the Visual Basic Editor and stored in a DVB file. Once a VBA project has been loaded, you can execute the macros through the Macros dialog box. Unlike standard AutoCAD commands, macros cannot be executed from the Command prompt, but once executed, a macro can prompt users for values at the Command prompt or with a user form. It is possible to execute a macro from a command macro that is activated with a command button displayed in the AutoCAD user interface or as a tool on a tool palette.

What to Expect

This book is organized to help you learn VBA fundamentals and how to use the objects in the AutoCAD Object library. Additional resources and files containing the example code found throughout this book can be found on the companion web page, `www.sybex.com/go/autocadcustomization`.

Chapter 1: Understanding the AutoCAD VBA Environment In this chapter, you'll get an introduction to the Visual Basic Editor. I begin by showing you how to verify whether the VBA environment for AutoCAD has been installed and, if not, how to install it. After that, you are eased into navigating the VBA Editor and managing VBA programs. The chapter wraps up with learning how to execute macros and access the help documentation.

Chapter 2: Understanding the Visual Basic for Application In this chapter, you'll learn the fundamentals of the VBA programming language and how to work with objects. VBA fundamentals include a look at the syntax and structure of a statement, how to use a function, and how to work with variables. Beyond syntax and variables, you learn to group multiple statements into a custom procedure.

Chapter 3: Interacting with the Application and Documents Objects In this chapter, you'll learn to work with the AutoCAD application and manage documents. Many of the tasks you perform with an AutoCAD VBA program require you to work with either the application or a document. For example, you can get the objects in a drawing and even access end-user preferences. Although you typically work with the current document, VBA allows you to work with all open documents and create new documents. From the current document, you can execute commands and work with system variables from within a VBA program, which allows you to leverage and apply your knowledge of working with commands and system variables.

Chapter 4: Creating and Modifying Drawing Objects In this chapter, you'll learn to create and modify graphical objects in model space with VBA. Graphical objects represent the drawing objects, such as a line, an arc, or a circle. The methods and properties of an object are used to modify and obtain information about the object. When working with the objects in a drawing, you can get a single object or step through all objects in a drawing.

Chapter 5: Interacting with the User and Controlling the Current View In this chapter, you'll learn to request input from an end user and manipulate the current view of a drawing. Based on the values provided by the end user, you can then determine the end result of the program. You can evaluate the objects created or consider how a drawing will be output and use that information to create named views and adjust the current view in which objects are displayed.

Chapter 6: Annotating Objects In this chapter, you'll learn how to create and modify annotation objects. Typically, annotation objects are not part of the final product that is built or manufactured based on the design in the drawing. Rather, annotation objects are used to communicate the features and measurements of a design. Annotation can be a single line of text that is used as a callout for a leader, a dimension that indicates the distance between two

drill holes, or a table that contains quantities and information about the windows and doors in a design.

Chapter 7: Working with Blocks and External References In this chapter, you'll learn how to create, modify, and manage block definitions. Model space in a drawing is a special named block definition, so working with block definitions will feel familiar. Once you create a block definition, you will learn how to insert a block reference and work with attributes along with dynamic properties. You'll complete the chapter by learning how to work with externally referenced files.

Chapter 8: Outputting Drawings In this chapter, you will learn how to output the graphical objects in model space or on a named layout to a printer, plotter, or electronic file. Named layouts will be used to organize graphical objects for output, including title blocks, annotation, floating viewports, and many others. Floating viewports will be used to control the display of objects from model space on a layout at a specific scale. After you define and configure a layout, you learn to plot and preview a layout. The chapter wraps up with covering how to export and import file formats.

Chapter 9: Storing and Retrieving Custom Data In this chapter, you will learn how to store custom information in a drawing or in the Windows Registry. Using extended data (Xdata), you will be able to store information that can be used to identify a graphical object created by your program or define a link to a record in an external database. In addition to attaching information to an object, you can store data in a custom dictionary that isn't attached to a specific graphical object in a drawing. Both Xdata and custom dictionaries can be helpful in making information available between drawing sessions; the Windows Registry can persist data between sessions.

Chapter 10: Modifying the Application and Working with Events In this chapter, you will learn how to customize and manipulate the AutoCAD user interface. You'll also learn how to load and access externally defined custom programs and work with events. Events allow you to respond to an action that is performed by the end user or the AutoCAD application. There are three main types of events that you can respond to: application, document, and object.

Chapter 11: Creating and Displaying User Forms In this chapter, you will learn how to create and display user forms. User forms provide a more visual approach to requesting input from the user.

Chapter 12: Communicating with Other Applications In this chapter, you will learn how to work with libraries provided by other applications. These libraries can be used to access features of the Windows operating system, read and write content in an external text or XML file, and even work with the applications that make up Microsoft Office.

Chapter 13: Handling Errors and Deploying VBA Projects In this chapter, you will learn how to catch and handle errors that are caused by the incorrect use of a function or the improper handling of a value that is returned by a function. The Visual Basic Editor provides tools that allow you to debug code statements, evaluate values assigned to user-defined variables, identify where within a program an error has occurred, and determine how errors should be handled. The chapter wraps everything up with covering how to deploy a VBA project on other workstations for use by individuals at your company.

Bonus Chapter 1: Working with 2D Objects and Object Properties In this chapter, you build on the concepts covered in Chapter 4, "Creating and Modifying Drawing Objects." You will learn to create additional types of 2D objects and use advanced methods of modifying objects; you also learn to work with complex 2D objects such as regions and hatch fills. The management of layers and linetypes and the control of the appearance of objects are also covered.

Bonus Chapter 2: Modeling in 3D Space In this chapter, you learn to work with objects in 3D space and 3D with objects. 3D objects can be used to create a model of a drawing which can be used to help visualize a design or detect potential design problems. 3D objects can be viewed from different angles and used to generate 2D views of a model that can be used to create assembly directions or shop drawings.

Bonus Chapter 3: Development Resources In this chapter, you discover resources that can help expand the skills you develop from this book or locate an answer to a problem you might encounter. I cover development resources, as well as places you might be able to obtain instructor-led training and interact with fellow users on extending AutoCAD. The online resources listed cover general customization, AutoLISP, and VBA programming in AutoCAD.

NOTE Bonus Chapter 1, Bonus Chapter 2, and Bonus Chapter 3 are located on the companion website.

Companion Website

An online counterpart to this book, the companion website contains the sample files required to complete the exercises found in this book, in addition to the sample code and project files used to demonstrate some of the programming concepts explained in this book. In addition to the sample files and code, the website contains resources that are not mentioned in this book, such as the bonus chapters. The companion website can be found at `www.sybex.com/go/autocadcustomization`.

Other Information

This book assumes that you know the basics of your operating system and AutoCAD 2009 or later. When appropriate, I indicate when a feature does not apply to a specific operating system or release of AutoCAD. Most of the images in this book were taken using AutoCAD 2014 in Windows 8.

Neither AutoCAD LT® nor AutoCAD running on Mac OS support the VBA programming platform, none of the content in this book can be used if you are working on Mac OS.

Styles and Conventions of This Book

This book uses a number of styles and character formats—bold, italic, monotype face, and all uppercase or lowercase letters, among others—to help you distinguish between the text you read, sample code you can try, text that you need to enter at the AutoCAD Command prompt, or the name of an object class or method in one of the programming languages.

As you read through this book, keep the following conventions in mind:

- User-interface selections are represented by one of the following methods:
  - Click the Application button ➢ Options.
  - On the ribbon, click the Manage tab ➢ Customization ➢ User Interface.
  - On the menu bar, click Tools ➢ Customize ➢ Interface.
  - In the drawing window, right-click and click Options.
- Keyboard input is shown in bold (for example, type **`cui`** and press Enter).
- Prompts that are displayed at the AutoCAD Command prompt are displayed as monospace font (for example, `Specify a start point:`).
- AutoCAD command and system variable names are displayed in all lowercase letters with a monospace font (for example, `line` or `clayer`).
- VBA function and AutoCAD Object library member names are displayed in mixed-case letters with a monospace font (for example, `Length` or `SendCommand`).
- Example code and code statements that appear within a paragraph are displayed in monospace font. Code might look like one of the following:
  - `MsgBox "ObjectName: " & oFirstEnt.ObjectName`
  - The `MsgBox` method can be used to display a text message to the user
  - `' Gets the first object in model space`

Contacting the Author

I hope that you enjoy *AutoCAD Platform Customization: VBA* and that it changes the way you think about completing your day-to-day work. If you have any feedback or ideas that could improve this book, you can contact me using the following address:

Lee Ambrosius: `lee_ambrosius@hyperpics.com`

On my blog and website, you'll find additional articles on customization and samples that I have written over the years. You'll find these resources here:

Beyond the UI: `http://hyperpics.blogs.com`

HyperPics: `www.hyperpics.com`

If you encounter any problems with this publication, please report them to the publisher. Visit the book's website, `www.sybex.com/go/autocadcustomization`, and click the Errata link to open a form and submit the problem you found.

Chapter 1

Understanding the AutoCAD VBA Environment

More than 15 years ago, Visual Basic (VB) was the first modern programming language I learned. This knowledge was critical to taking my custom programs to a whole new level. VB allows you to develop stand-alone applications that can communicate with other programs using Microsoft's Object Linking and Embedding (OLE) and ActiveX technologies. Autodesk® AutoCAD® supports a variant of VB known as Visual Basic for Applications (VBA) that requires a host application to execute the programs you write; it can't be used to develop stand-alone executable files.

I found VB easier to learn than AutoLISP® for a couple of reasons. First, there are, in general, many more books and online resources dedicated to VB. Second, VB syntax feels more natural. By natural, I mean that VB syntax reads as if you are explaining a process to someone in your own words, and it doesn't contain lots of special characters and formatting like many other programming languages.

As with learning anything new, there will be a bit of hesitation on your part as you approach your first projects. This chapter eases you into the AutoCAD VBA environment and the VB programming language.

What Makes Up an AutoCAD VBA Project?

Custom programs developed with VBA implemented in the AutoCAD program are stored in a project that has a `.dvb` file extension. VBA projects contain various objects that define a custom program. These objects include the following:

- Code modules that store the custom procedures and functions that define the primary functionality of a custom program
- `UserForms` that define the dialog boxes to be displayed by a custom program
- Class modules that store the definition of a custom object for use in a custom program
- Program library references that contain the dependencies a custom program relies on to define some or all of the functionality

The AutoCAD VBA Editor is an integrated development environment (IDE) that allows for the creation and execution of macros stored in a project file. A macro is a named block of code that can be executed from the AutoCAD user interface or privately used within a project. You can also enter and execute a single VBA statement at the AutoCAD Command prompt using the `vbastmt` command.

The most recent generation of VB is known as VB.NET. Although VB and VB.NET have similar syntax, they are not the same. VBA, whether in AutoCAD or other programs such as Microsoft Word, is based on VB6 and not VB.NET. If you are looking for general information on VBA, search the Internet using the keywords VBA and VB6.

Real World Scenario

If You Have Conversations Like This, You Can Code Like This

The summer intern had one job—add a layer and a confidentiality note to a series of 260 production drawings. September arrived, the intern left for school, and now your manager is in your cubicle.

"Half of these drawings are missing that confidentiality note Purchasing asked for. I need you to **add** that new **layer**, name it **Disclaimer**, and then **add** the confidentiality note as **multiline text** to **model space**. The note should be located on the new **Disclaimer** layer at **0.25,0.1.75,0** with a **height** of **0.5,** and the text should read **Confidential: This drawing is for use by internal employees and approved vendors only**. Be sure to check to see **if paper space** is **active.** If it is, **then** set **model space active** per the new standards before you **save** each drawing," he says.

"I can do that," you respond.

"Can you manage it by close of day tomorrow? The parts are supposed to go out for quote on Wednesday morning."

"Sure," you tell him, knowing that a few lines of VBA code will allow you to make the changes quickly.

So, you sit down and start to code. The conversation-to-code translation flows smoothly. (Notice how many of the words in the conversation flow right into the actual VBA syntax.)

```
With ThisDrawing
  .Layers.Add "Disclaimer"

  Dim objMText As AcadMText

  Dim insPt(2) As Double
  insPt(0) = 0.25: insPt(1) = 1.75: insPt(2) = 0

  Set objMText = .ModelSpace.AddMText(insPt, 15, _
    "Confidential: This drawing is for use by internal " & _
    "employees and approved vendors only")

  objMText.Layer = "Disclaimer"
  objMText.Height = 0.5

  If .ActiveSpace = acPaperSpace Then
    .ActiveSpace = acModelSpace
   End If

  .Save
End With
```

What You'll Need to Start

To complete the exercises in this chapter and create and edit VBA project files, you must have the following:

- AutoCAD 2006 or later
- Autodesk AutoCAD VBA Enabler for AutoCAD 2010 or later

Beginning with AutoCAD 2010, the AutoCAD VBA Enabler is an additional component that must be downloaded and installed to enable VBA support in the AutoCAD drawing environment. (For AutoCAD 2000 through AutoCAD 2009, VBA capabilities were part of a standard install.)

NOTE The Autodesk website (`http://www.autodesk.com/vba-download`) allows you to download the Autodesk AutoCAD VBA Enabler for AutoCAD 2014 and 2015 (Microsoft Visual Basic for Applications Module). If you need the VBA Enabler for AutoCAD 2010 through 2013, you will want to check with your local Autodesk Value Added Reseller.

Without the VBA Enabler, you won't have access to the VBA Editor and can't create or execute VBA code contained in a DVB file with AutoCAD 2010 and later releases. All of the VBA commands were available without an additional download and install. Changes in the later AutoCAD releases were made due to Microsoft's planned deprecation of the VBA technology and editor, only to eventually extend its life cycle because of its continued importance to Microsoft Office. Microsoft planned to move to Visual Studio Tools for Applications (VSTA) as the replacement for VBA, but the company backed off because there was no easy migration from VBA to VSTA.

NOTE Although I mention AutoCAD 2006 or later, everything covered in this chapter should work without any problems going all the way back to AutoCAD 2000. The first release of the AutoCAD program that supported VBA was AutoCAD R14, and much has remained the same since then as well, with the exception of being able to work with multiple documents in AutoCAD 2000 and later.

Determine If the AutoCAD VBA Environment Is Installed

Prior to working with the AutoCAD VBA Editor, you must ensure that the VBA environment is installed on your workstation. The following steps explain how to determine whether VBA is installed and, if necessary, how to download the AutoCAD VBA environment for installation. These steps are important if you are using AutoCAD 2010 or later.

1. Launch AutoCAD if it isn't already running.
2. At the Command prompt, type **`vbaide`** and press Enter.
3. If the VBA - Not Installed message box is displayed, the AutoCAD VBA environment hasn't been installed. Continue to the next step.
4. Click the `http://www.autodesk.com/vba-download` link to open your system's default web browser to the download website.

5. Click the link for the AutoCAD VBA Enabler that matches the version of AutoCAD installed on your workstation.
6. Save the AutoCAD VBA Enabler to a folder on your local workstation.

Install the AutoCAD 2015 VBA Enabler

After downloading the AutoCAD 2015 VBA Enabler using the steps explained in the previous section, follow these steps to install it:

1. Close the AutoCAD program and double-click the downloaded self-extracting executable for the AutoCAD VBA module.
2. In the Extract To message, accept the default destination location and click OK.
3. When the AutoCAD VBA Enabler installer opens, click Install.
4. On the next page of the installer, accept the default destination location and click Install.
5. On the Installation Complete page, click Finish.
6. Launch AutoCAD.
7. At the Command prompt, type **vbaide** and press Enter.

   The VBA Editor is displayed, indicating that the AutoCAD VBA environment has been installed.

NOTE If you downloaded the VBA Enabler for a different release of the AutoCAD program, follow the on-screen instructions for that release of the VBA Enabler.

Getting Started with the VBA Editor

The VBA Editor (see Figure 1.1) is the authoring environment used to create custom programs that are stored in a VBA project. The following tasks can be performed from the VBA Editor:

- Access and identify the components in a VBA project
- View and edit the code and components stored in a loaded VBA project
- Debug the code of a procedure during execution
- Reference programming libraries
- Display contextual help based on the code or component being edited

Any of the following methods can be used to display the VBA Editor:

- On the ribbon, click the Manage tab ➢ Applications panel ➢ Visual Basic Editor.
- At the Command prompt, type **vbaide** and press Enter.
- When the VBA Manager is open, click Visual Basic Editor.
- When loading a VBA project, in the Open VBA Project dialog box, check the Open Visual Basic Editor check box before clicking Open.

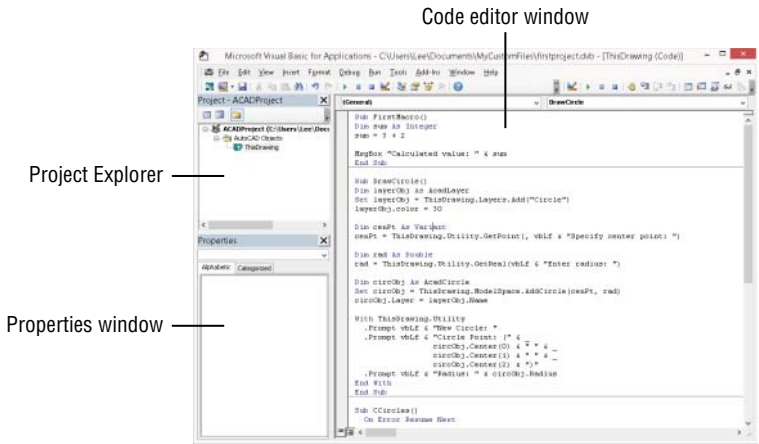


Figure 1.1
The VBA Editor allows for the development of a VBA program

Identifying the Components of a VBA Project

VBA supports four types of components to define the functionality of a custom program. Each component can be used to store code, but the code in each component serves a distinct purpose within a VBA project. Before you begin learning the basic features of the VBA Editor, you should have a basic understanding of the component types in a VBA project.

The following provides an overview of the component types:

Code Module Code modules, also referred to as standard code modules, are used to store procedures and define any global variables for use in the module or globally across the VBA project. I recommend using code modules to store procedures that can be executed from the AutoCAD user interface or used across multiple projects.

When you add a new code module to a VBA project, you should give the module a meaningful name and not keep the default name of `Module1`, `Module2`, and so on. Standard industry naming practice is to add the prefix of `bas` to the name of the module. For example, you might name a module that contains utility procedures as `basUtilities`. I explain how to define procedures and variables in the "Learning the Fundamentals of the VBA Language" section in Chapter 2, "Understanding Visual Basic for Applications."

Class Module Class modules are used to define a custom class—or *object*. Custom classes aren't as common as code modules in a VBA project, but they can be helpful in organizing and simplifying code. The variables and procedures defined in a class module are hidden from all other components in a VBA project, unless an instance of the class is created as part of a procedure in another component.

When you add a new class module to a VBA project, you should give the module a meaningful name and not keep the default name of `Class1`, `Class2`, and so on. Standard industry naming practice is to add the prefix of `cls` to the name of the module. For example, you might name a module that contains a custom class named employee as `clsEmployee`. I explain how to define procedures and variables and work with objects in the "Learning the Fundamentals of the VBA Language" section in Chapter 2.

ThisDrawing ThisDrawing is a specially named object that represents the current drawing and is contained in each VBA project. The ThisDrawing component can be used to define events that execute code based on an action performed by the user or the AutoCAD program. Variables and procedures can be defined in the ThisDrawing component, but I recommend storing only the variables and procedures related to the current drawing in the ThisDrawing component. All other code should be stored in a code module. I explain how to work with the current drawing and events in Chapter 3, "Interacting with the Application and Documents Objects," and Chapter 10, "Modifying the Application and Working with Events."

`UserForm` `UserForms` are used to define custom dialog boxes for use in a VBA program. A `UserForm` can contain controls that present messages to the user and allow the user to provide input. When you add a new `UserForm` to a VBA project, you should give the `UserForm` a meaningful name and not keep the default name of `UserForm1`, `UserForm2`, and so on. Standard industry naming practice is to add the prefix of `frm` to the name of the `UserForm`. For example, you might name a `UserForm` that contains a dialog box that draws a bolt as `frmDrawBolt`. I explain how to create and display a `UserForm` in Chapter 11, "Creating and Displaying User Forms."

The following explains how to add a new component to a VBA project and change its name:

1. In the VBA Editor with a project loaded, on the menu bar, click Insert.
2. Click UserForm, Module, or Class Module to add a component of that type to the VBA project.
3. In the Project Explorer, select the new component.
4. In the Properties window, in the (Name) field, type a new name and press Enter.

Using Components in Multiple VBA Projects

A component added to a VBA project can be exported, and then imported into another VBA project. Exporting a component creates a copy of that component; any changes to the component in the original VBA project don't affect the exported copy of the component. Importing the component into a VBA project creates a copy of the component in that VBA project.

The following steps can be used to export a VBA component to a file:

1. In the VBA Editor, Project Explorer, select the component to export.
2. On the menu bar, click File ➢ Export File.
3. In the Export File dialog box, browse to the location to store the exported file and enter a filename. Click Save.

The following steps can be used to import an exported file into a VBA project:

1. In the VBA Editor, Project Explorer, select a loaded project to set it current.
2. On the menu bar, click File ➢ Import File.
3. In the Import File dialog box, browse to and select the exported file. Click Open.

Navigating the VBA Editor Interface

The VBA Editor interface contains a variety of tools and windows that are used to manage and edit the components and code stored in a VBA project. While all of the tools and windows in the VBA Editor will be important over time, there are four windows that you should have a basic understanding of when first getting started:

- Project Explorer
- Properties window
- Code editor window
- Object Browser

Accessing Components in a VBA Project with the Project Explorer

The Project Explorer window (see Figure 1.2) lists all of the VBA projects that are currently loaded into the AutoCAD drawing environment and the components of each loaded project. By default, the Project Explorer should be displayed in the VBA Editor, but if it isn't you can display it by clicking View ➢ Project Explorer or pressing Ctrl+R.

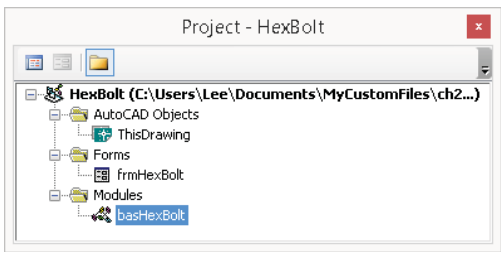


Figure 1.2
The Project Explorer lists loaded projects and components

When the Project Explorer is displayed, you can

- Select a project to set it as the current project; the name of the current project is shown in bold. Some tools in the VBA Editor work on only the current project.
- Expand a project to access its components.
- Toggle the display style for components; alphabetically listed or grouped by type in folders.
- Double-click a component to edit its code or `UserForm` in an editor window.
- Right-click to export, import, or remove a component.

Using the Properties Window

The Properties window (see Figure 1.3) allows you to change the name of a component in a loaded VBA project or modify the properties of a control or `UserForm`. Select a component or `UserForm` from the Project Explorer, or a control to display its properties in the Properties window. Click in a property field, and enter or select a new value to change the current value of the property. The Properties window is displayed by default in the VBA Editor, but if it isn't you can display it by clicking View ➢ Properties Window or pressing F4.

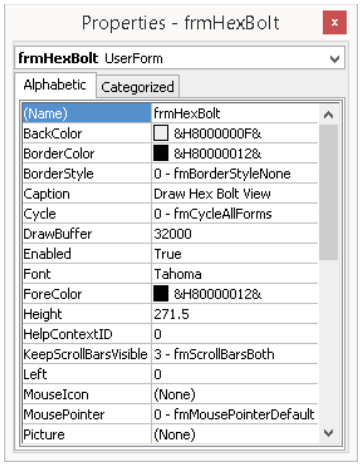


FIGURE 1.3
Modify the properties of a component, `UserForm`, or control

EDITING CODE AND CLASS MODULES IN EDITOR WINDOWS

A code editor window (see Figure 1.4) is where you will write, edit, and debug code statements that are used to make up a custom program. You display a code editor window by doing one of the following in the Project Explorer:

- Double-clicking a code or class module
- Right-clicking a `UserForm` and then clicking View Code

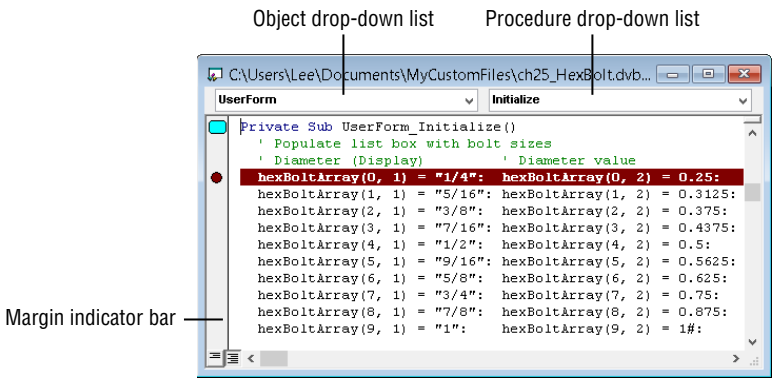


FIGURE 1.4
Edit code statements stored in a code or class module.

The code editor window supports many common editing tools: copy and paste, find and replace, and many others. In addition to common editing tools, it supports tools that are designed specifically for working with VBA code statements, and some of these tools allow you to accomplish the following:

- Autocomplete a word as you type
- Find and replace text across all components in a VBA project

- Comment and uncomment code statements
- Add bookmarks to allow you to move between procedures and code statements
- Set breakpoints for debugging

The text area is the largest area of the code editor window and where you will spend most of your time. The Object drop-down list typically is set to (General), which indicates you want to work with the General Declaration area of the code window. When working in the code editor window of a `UserForm`, you can select a control or the `UserForm` to work with from the Object drop-down list. The Object drop-down list is also used when working with events.

Once an object is selected, a list of available events or procedures for the selected object is displayed in the Procedure drop-down list. Select a procedure from the drop-down list to insert the basic structure of that procedure. Enter the code statements to execute when the procedure is executed. I explain how to work with events in Chapter 10 and `UserForms` in Chapter 11.

The margin indicator bar of the code editor window helps you know where a bookmark or breakpoint is inserted by displaying an oval for a bookmark or a circle for a breakpoint. I discuss more about breakpoints in Chapter 13, "Handling Errors and Deploying VBA Projects."

Exploring Loaded Libraries with the Object Browser

The Object Browser (see Figure 1.5) allows you to view the classes and enumerated constants defined in a referenced programming library. Each AutoCAD VBA project contains a reference to the VBA and AutoCAD Object libraries. I discuss referencing other libraries in Chapter 12, "Communicating with Other Applications." You can display the Object Browser by clicking View ➢ Object Browser or pressing F2.

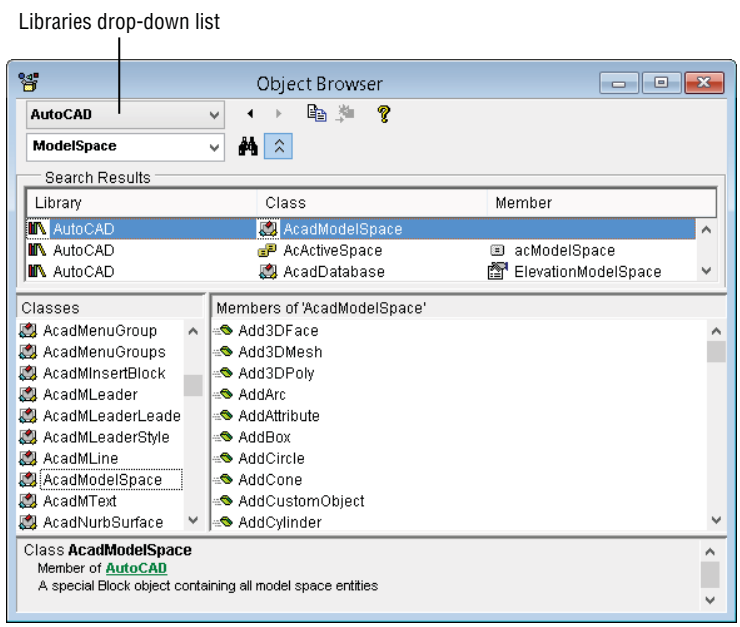


Figure 1.5 Members of an object in a referenced library can be viewed in the Object Browser.

A class is used to create an instance of an object, which I discuss in the "Working with Objects" section in Chapter 2. An enumerated constant is a set of integer values with unique

names that can be used in a code statement. Using a constant name makes the integer value easier to understand, and also protects your code when values change. For example, the constant name of `acBlue` is equal to an integer value of 5. If the meaning of 5 were changed to mean a different color than blue, the constant of `acBlue` would be updated with the new integer and no changes to your code would need to be made if you used the constant.

When the Object Browser is displayed, you can select a class or enumerated constant from the Classes list. The Classes list contains all the classes and enumerated constants of the referenced libraries in the VBA project. You can filter the list by selecting a referenced library from the Libraries drop-down list located at the top of the Object Browser. Select a class or enumerated constant from the Classes list to see its members, which are methods, properties, events, or constant values. Select a member to learn more about it and press F1 to display its associated help topic. I explain how to access the AutoCAD VBA documentation in the "Accessing the AutoCAD VBA Documentation" section later in this chapter.

Working with Other Windows

The four windows that I described in the previous sections are the main windows of the VBA Editor; they are used the most frequently. You will use some additional windows on occasion. These are primarily used for creating `UserForms` and debugging VBA statements. (I discuss creating `UserForms` in Chapter 11 and debugging in Chapter 13.)

Here are the windows you will use when creating `UserForms` and debugging:

Immediate Window The Immediate window allows you to execute code statements in real time, but those code statements are not saved. Not all code statements can be executed in the Immediate window, such as statements that define a procedure and declare variables. Text messages and values assigned to a variable can be output to the Immediate window for debugging purposes with the `Print` method of the `Debug` object. I discuss more about the `Debug` object and Immediate window in Chapter 13.

Watches Window The Watches window allows you to monitor the current value assigned to the variables used in the procedures of your VBA project as they are being executed. When an array or object is assigned to a variable, you can see the values assigned to each element in the array and the current property values of the object in the Watches window. In the code editor window, highlight the variable you want to watch, and right-click. Click Add Watch and then when the Add Watch dialog box opens click OK. I discuss more about the Watches window in Chapter 13.

`UserForm` Editor Window The `UserForm` editor window allows you to add controls and organize controls on a `UserForm` to create a custom dialog box that can be displayed from your VBA project. You add controls to a `UserForm` from the Toolbox window. While the `UserForm` editor window is current, the Format menu on the menu bar contains tools to lay out and align the controls on a `UserForm`. I explain how to create and work with `UserForms` in Chapter 11.

Toolbox Window The Toolbox window contains the controls that can be added to a `UserForm` when displayed in the `UserForm` editor window. Click a tool and then drag it into the `UserForm` editor window to place an instance of the control. Right-click over one of the tools on the window and click Additional Controls to display the Additional Controls dialog box. Click any of the available controls to make it available for use in a `UserForm`. I explain how to add controls to a `UserForm` in Chapter 11.

Setting the VBA Environment Options

There are several settings that affect the behavior of the AutoCAD VBA environment and not just the currently loaded VBA projects. These settings can be changed in the Option dialog box of the VBA environment (see Figure 1.6), which can be displayed using one of the following methods:

- After the Macros dialog box has been opened with the `vbarun` command, click Options.
- At the Command prompt, type **`vbapref`** and press Enter.

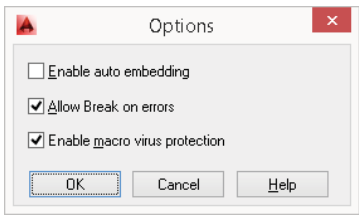


FIGURE 1.6 Changing the VBA environment settings

Here is an explanation of the settings in the Options dialog box:

Enable Auto Embedding The Enable Auto Embedding option creates a new empty VBA project each time a drawing file is opened and embeds that empty project into the drawing file. A new project is created and embedded only if the drawing opened doesn't already contain an embedded project. This option is disabled by default.

Allow Break On Errors The Allow Break On Errors option displays a message box that allows you to step into a procedure if an error is produced during execution. You can then use the debugging tools offered by the VBA Editor to locate and handle the error. I discuss debug procedures in Chapter 13. This option is enabled by default.

Enable Macro Virus Protection The Enable Macro Virus Protection option, when enabled, displays a message box during the loading of a DVB file. I recommend leaving this option enabled to ensure that a drawing file with an embedded VBA project isn't opened in the AutoCAD drawing environment. This reduces the risk of accidentally running malicious code. The option is enabled by default.

Managing VBA Programs

VBA programs developed in the AutoCAD VBA environment can be stored in a project file or embedded in a drawing file. VBA projects can also be embedded in a drawing template (DWT) or drawing standards (DWS) file. By default, VBA programs developed in the AutoCAD VBA environment are stored in a project file with a `.dvb` file extension and then are loaded into the AutoCAD drawing environment as needed.

DVB files can be managed externally from Windows Explorer or File Explorer, or from within AutoCAD whenever the file is loaded into the AutoCAD drawing environment. General file-management tasks on a DVB file can be performed using Windows Explorer or File Explorer.

Once the DVB file is loaded into the AutoCAD drawing environment, you can manage it using the VBA Manager (see Figure 1.7). The VBA Manager allows you to do the following:

- Create a new VBA project
- Save a VBA project to a DVB file
- Load a VBA project from a DVB file into the AutoCAD drawing environment
- Unload a VBA project from the AutoCAD drawing environment
- Edit the components and code stored in a VBA project
- Embed or extract a VBA project from a drawing file

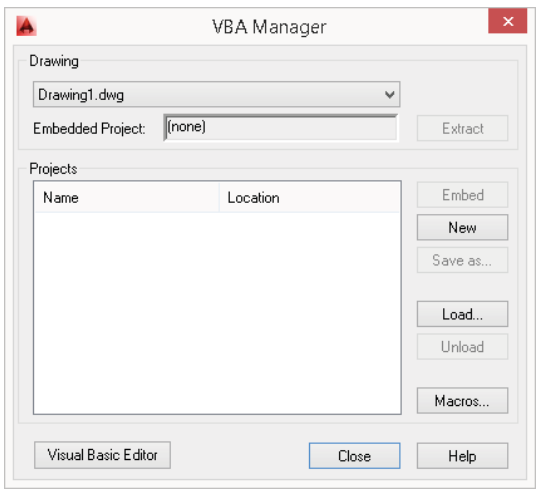


FIGURE 1.7
Managing loaded VBA programs

There are two ways to display the VBA Manager in AutoCAD:

- On the ribbon, click the Manage tab ➢ Applications panel title bar and then click VBA Manager.
- At the Command prompt, type **vbaman** and press Enter.

Creating a New VBA Project

A new VBA project can be created automatically by the AutoCAD program or manually as needed. When the VBA environment is initialized the first time during an AutoCAD session, a new VBA project is created automatically unless a VBA project has already been loaded into memory. If you want to create a new project after the VBA environment has been initialized, do one of the following:

- When the VBA Manager is open, click New.
- At the Command prompt, type **vbanew** and press Enter.

Each new VBA project is assigned two default names: a project name and a location name. The project name is an internal name used by the AutoCAD program to differentiate the procedures and components in each loaded VBA project. The default project name for a new VBA project is `ACADProject`; I recommend assigning a descriptive project name for each VBA project

you create. A project name can contain alphanumeric characters and underscores, but can't start with a number or underscore character.

The location name of a VBA project is the same as a filename and is used to specify where the DVB file is stored. Since a new VBA project exists only in memory, it is assigned the default location name of Global1. The location name is incremented by one for each new VBA project created during an AutoCAD session; thus the second and third VBA projects have location names of Global2 and Global3, respectively. When you save VBA projects, they are stored in DVB files locally or on a network. To ensure that AutoCAD knows where the DVB files are located, you add the locations of your DVB files to the AutoCAD support file search and trusted paths. I discuss how to add a folder to the AutoCAD support file search and trusted paths in Chapter 13.

Saving a VBA Project

New VBA projects can be saved to disc using the Save As option in the VBA Manager or Save in the VBA Editor. When an existing project is loaded in memory, the Save As option can be used to create a copy of the project on disc or to overwrite an existing VBA project file. Typically, changes made to an existing project file that already has been loaded in the VBA environment are saved to the project file using the Save option in the VBA Editor. I discussed the VBA Editor earlier in the "Getting Started with the VBA Editor" section.

The following explains how to save a VBA project:

1. In the VBA Editor, click File ➢ Save. Alternatively, on the Standard toolbar click Save.
2. If the project hasn't been previously saved, the Save As dialog box is displayed. Otherwise, the changes to the VBA project are saved.
3. When the Save As dialog box opens, browse to the folder you wish to use to store the VBA project.
4. In the File Name text box, type a descriptive filename for the project and click Save.

NOTE A DVB file can be password-protected to restrict the editing of the components and code stored in the file. I discuss how to assign a password to a VBA project in Chapter 13.

Loading and Unloading a VBA Project

Before a VBA project can be edited and before the code stored in the project can be executed, the project must be loaded into the AutoCAD VBA environment. The process for loading a project into the AutoCAD VBA environment is similar to opening a drawing file.

Manually Loading a VBA Project

A VBA project can be manually loaded using the VBA Manager or the vbaload command. The following explains how to manually load a VBA project:

1. On the ribbon, click the Manage tab and then click the Applications panel title bar. Click Load Project. (As an alternative, at the Command prompt, type **vbaload** and press Enter.)
2. When the Open VBA Project dialog box opens, browse to and select ch01_hexbolt.dvb.
3. Clear the Open Visual Basic Editor check box and click Open.

4. If the File Loading - Security Concern dialog box is displayed, click Load to load the file into memory. (You can click Do Not Load to cancel a load operation.)
5. If the AutoCAD dialog box is displayed, click Enable Macro to allow the execution of the code in the project. (You can click Disable Macros to load a file but not allow the execution of the code, or Do Not Load to cancel without loading the project into memory.)

NOTE You can download the sample VBA project file `ch01_hexbolt.dvb` used in the following exercise from `www.sybex.com/go/autocadcustomization`.

Place the file in the `MyCustomFiles` folder within the `Documents` (or `My Documents`) folder or another location you are using to store custom program files.

As an alternative, DVB and other types of custom program files can be dragged and dropped onto an open drawing window in the AutoCAD drawing environment. When you drop a DVB file onto an open drawing window, AutoCAD prompts you to load the file and/or to enable the macros contained in the VBA project file.

Automatically Loading a VBA Project

The Load Project button in the VBA Manager and the `vbaload` command require input from the user to load a VBA project, which isn't ideal when you want to integrate your VBA projects as seamlessly as possible into the AutoCAD drawing environment. A script, custom AutoLISP program, or command macro from the AutoCAD user interface can all be used to load a VBA project without user input. The following outlines some of the methods that can be used to load a VBA project without user input:

- Call the `-vbaload` command. The `-vbaload` command is the command-line version of the `vbaload` command. When the `-vbaload` command is started, the `Open VBA Project:` prompt is displayed. Provide the name of the DVB file as part of the macro or script file.
- Call the AutoLISP `vl-vbaload` function. The AutoLISP `vl-vbaload` function can be used to load a DVB file from a custom AutoLISP program. If the DVB file that is passed to the `vl-vbaload` function isn't found, an error is returned that should be captured with the AutoLISP `vl-catch-all-apply` function.
- Create a VBA project file named `acad.dvb` and place it in one of the AutoCAD support file search paths. AutoCAD looks for a file named `acad.dvb` during startup and if the file is found, that file is loaded automatically.
- Use the Startup Suite (part of the Load/Unload Applications dialog box that opens with the `appload` command). When a DVB file is added to the Startup Suite, the file is loaded when the first drawing of a session is opened. Removing a file from the Startup Suite causes the file not to be loaded in any future drawings that are opened or in AutoCAD sessions. If you want to use the Startup Suite to load DVB files, you must add the files to the Startup Suite on each workstation and AutoCAD user profile.
- Create a plug-in bundle. Plug-in bundles allow you to load DVB and other custom program files in AutoCAD 2013 or later. A plug-in bundle is a folder structure with a special name and metadata file that describes the files contained in the bundle.

I discuss each of these methods in greater detail in Chapter 13.

Manually Unloading a VBA Project

When a VBA project is no longer needed, it can be unloaded from memory to release system resources. A VBA project can be manually unloaded from memory using the VBA Manager or the `vbaunload` command. The following explains how to unload the `ch01_hexbolt.dvb` file with the VBA Manager:

1. On the ribbon, click the Manage tab and then click the Applications panel title bar to expand the panel. Click VBA Manager. (If the ribbon isn't displayed or the release of the AutoCAD program you are using doesn't support the ribbon, at the Command prompt type **`vbaman`** and press Enter.)
2. When the VBA Manager dialog box opens, in the Projects list select HexBolt and click Unload.
3. If prompted to save changes to the VBA project, click Yes if you made changes that you wish to save or No to discard any changes.

Automatically Unloading a VBA Project

If you want to unload a DVB file as part of a script, custom AutoLISP program, or command macro from the AutoCAD user interface, you will need to use the `vbaunload` command. When the `vbaunload` command starts, the `Unload VBA Project:` prompt is displayed. Provide the filename and full path of the DVB file you want to unload; the path you specify must exactly match the path for the DVB file that was loaded into the AutoCAD drawing environment. If it doesn't, the unload fails and an error message will be displayed. A failed execution of the `vbaunload` command doesn't cause the program calling the command to fail.

TIP I recommend using the AutoLISP `findfile` function to locate the DVB file in the AutoCAD support file search paths when loading and unloading a DVB file to ensure that the correct path is provided.

Embedding or Extracting a VBA Project

A VBA project can be embedded in a drawing file to make the components and code in the project available when the drawing file is opened in the AutoCAD drawing environment. Only one VBA project can be embedded in a drawing file at a time. Embedding a VBA project in a file can be helpful to make specific tools available to anyone who opens the file, but there are potential problems using this approach. Here are the main two problems with embedding a VBA project file into a drawing:

- Embedding a VBA project triggers a security warning each time a drawing file is opened, which could impact sharing drawing files. Many companies will not accept drawings with embedded VBA projects because of potential problems with viruses and malicious code.
- Embedding a VBA project that is stored in a DVB file results in a copy of that project being created and stored in the drawing file. The embedded project and the original DVB file are kept separately. This can be a problem if the project is embedded in hundreds of drawing

files and needs to be revised. Each drawing file would need to be opened, the project extracted, and the revised project re-embedded.

So, while you can embed a VBA project, I don't recommend doing it.

Embedding a VBA Project

The following explains how to embed a VBA project in a current drawing file:

1. On the ribbon, click the Manage tab ➢ Applications panel title bar and then click VBA Manager.
2. When the VBA Manager opens, select a VBA project to embed from the Projects list. Load the VBA project you want to embed if it isn't already loaded.
3. Click Embed.

Extracting a VBA Project

Extracting a VBA project reverses the embedding process. After a project is selected for extraction, you can either export the project to a DVB file or discard the project. The following explains how to extract a VBA project from a current drawing file:

1. On the ribbon, click the Manage tab and then click Applications panel title bar to expand the panel. Click VBA Manager.
2. When the VBA Manager opens, in the Drawing section click Extract. (If the Extract button is disabled, there is no VBA project embedded in the current drawing.)
3. In the AutoCAD message box, click Yes to remove and export the VBA project to a DVB file. Specify a filename and location for the project you wish to extract. Click No if you wish to remove the VBA project from the drawing file without saving the project.

Executing VBA Macros

VBA projects contain components that organize code and define user forms and custom classes. A component can contain one or more procedures that are used to perform a task on the objects in a drawing or request input from an end user. Most procedures are defined so they are executed from other procedures in a VBA project and not from the AutoCAD user interface. A procedure that can be executed from the AutoCAD user interface is known as a *macro*. I explain how to define a procedure in Chapter 2.

A macro can be executed using the Macros dialog box (see Figure 1.8). In addition to executing a macro, the Macros dialog box can also be used to do the following:

- Execute and begin debugging a macro
- Open the VBA Editor and scroll to a macro's definition
- Create the definition of a new macro based on the name entered in the Macro Name text box

- Remove a macro from a loaded project
- Display the VBA Manager
- Change the VBA environment options

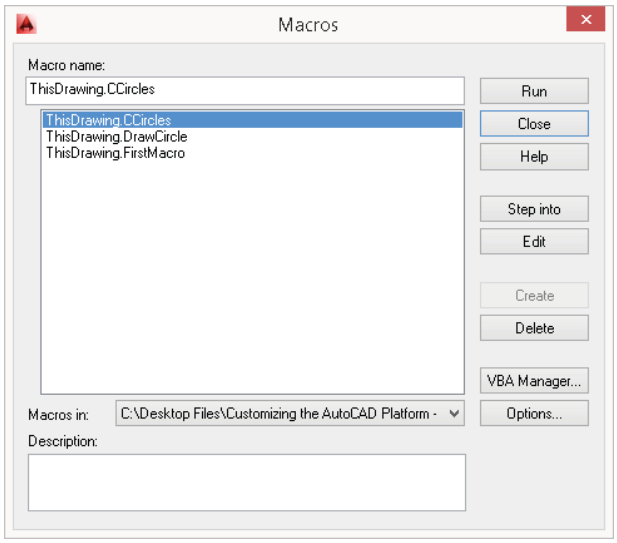


FIGURE 1.8
Executing a macro stored in a VBA project

The following methods can be used to display the Macros dialog box:

- On the ribbon, click the Manage tab ➢ Applications panel ➢ Run VBA Macro.
- At the Command prompt, type **vbarun** and press Enter.
- When the VBA Manager opens, click Macros.

The Macros dialog box requires input from the user to execute a macro in a loaded VBA project. If you want to execute a macro as part of a script, custom AutoLISP program, or command macro from the AutoCAD user interface you can use one of the following methods:

Command Line The `-vbarun` command is the command-line version of the `vbarun` command. When the `-vbarun` command is started, the `Macro name:` prompt is displayed.

AutoLISP The AutoLISP `vl-vbarun` function can be used to execute a macro in a loaded DVB file from a custom AutoLISP program. If the macro isn't found, an error message is displayed but the error doesn't cause the program to terminate.

The name of the macro to execute with the `-vbarun` command or `vl-vbarun` function must be in the following format:

```
DVBFilename.ProjectName!MacroName
```

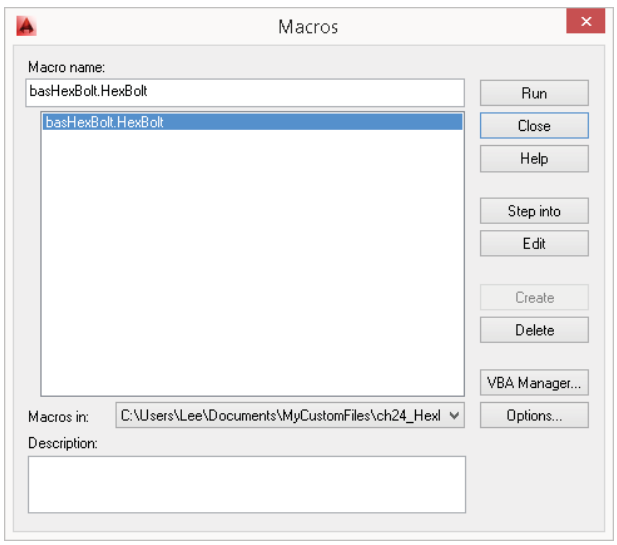
For example, you would use the string value `firstproject.dvb!ThisDrawing.CCircles` to execute the `CCircle` macro in the `ThisDrawing` component of the `firstproject.dvb` file.

These steps explain how to execute the macro named `hexbolt`:

1. On the ribbon, click the Manage tab ➢ Applications panel ➢ Run VBA Macro (or at the Command prompt, type **vbarun** and press Enter).
2. When the Macros dialog box opens, click the Macros In drop-down list and choose `ch01_hexbolt.dvb`.

   Figure 1.9 shows the macro that is stored in and can be executed from the `ch01_hexbolt.dvb` file with the Macros dialog box.

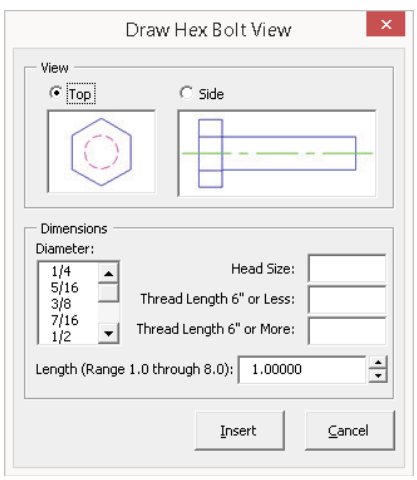
FIGURE 1.9
Edit, debug, and execute macros from the Macros dialog box.



3. In the Macros list, choose basHexBolt.HexBolt and click Run.

   The Draw Hex Bolt View dialog box, shown in Figure 1.10, is displayed.

FIGURE 1.10
Custom dialog box used to draw a top or side view of a hex bolt



4. In the Diameter list box, choose 3/8 and click Insert.
5. At the `Specify center of bolt head:` prompt, specify a point in the drawing area to draw the top view of the hex bolt.
6. When the Draw Hex Bolt View dialog box reappears, in the View section click the Side option or image. Click Insert.
7. At the `Specify middle of bolt head:` prompt, specify a point in the drawing area to draw the side view of the hex bolt.
8. When the Draw Hex Bolt View dialog box reappears again, click Cancel.

Figure 1.11 shows the top and side views of the hex bolt that were drawn with the macro.

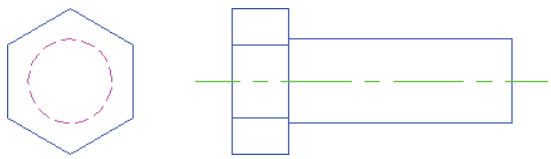


FIGURE 1.11
Views of the completed hex bolt

Accessing the AutoCAD VBA Documentation

The AutoCAD VBA documentation is available from the AutoCAD product Help landing page and the VBA Editor. The documentation is composed of two documentation sets: the AutoCAD Object Library Reference and the ActiveX Developer's Guide. Although this book is designed to make it easy to learn how to use the AutoCAD Object library and the VBA programming language, you will want to refer to the documentation that is provided with the AutoCAD product too, as it just isn't possible to cover every function and technique here.

The topics of the AutoCAD Object Library Reference explain the classes, methods, properties, and constants that make up the AutoCAD Object library. The ActiveX Developer's Guide topics can be used to explore advanced techniques and features that aren't covered in this book.

You can see the AutoCAD VBA and ActiveX documentation written for AutoCAD 2015 here:

```
http://help.autodesk.com/view/ACD/2015/ENU/
```

On the Autodesk AutoCAD 2015 Help landing page, click the Developer Home Page link. On the AutoCAD Developer Help Home Page, use the AutoCAD Object Library Reference and Developer's Guide links under the ActiveX/VBA section to access the AutoCAD VBA and ActiveX documentation.

When working in the VBA Editor, you can access the AutoCAD Object Library Reference and Microsoft Visual Basic for Applications Help by doing the following:

1. In a code editor window, highlight the keyword, statement, data type, method, property, or constant that you want to learn more about.
2. Press F1.

Help can also be accessed from the Object Browser. In the Object Browser, select a class, method, property, or constant and then press F1 to open the associated help topic. I discussed the Object Browser earlier, in the "Exploring Loaded Libraries with the Object Browser" section.

Chapter 2

Understanding Visual Basic for Applications

The Visual Basic for Applications (VBA) programming language is a variant of the Visual Basic 6 (VB6) programming language that was introduced in 1998. Though similar, VB6 isn't exactly the same as the current version of Visual Basic (known as VB.NET). Unlike VB6, which allows you to develop stand-alone applications, VBA programs require a host application. The host application provides the framework in which the VBA program can be executed; Microsoft Word and the Autodesk® AutoCAD® program are examples of host applications.

VBA was first introduced as a preview technology and modern programming alternative to AutoLISP® and ObjectARX® with AutoCAD Release 14 back in 1997. It was not until after the release of AutoCAD R14.01 that VBA was officially supported. The implementation of VBA in the AutoCAD program at that time was huge to the industry, as the learning curve between AutoLISP and C++ was steep, and the number of developers who knew VBA was growing rapidly.

Here are some of the reasons I recommend using VBA for your custom programs:

- Individuals with VB/VBA experience often can be found in-house (check in your company's IS/IT department); finding someone fluent in AutoLISP or ObjectARX is much rarer.
- VB/VBA resources are easier to locate—on the Internet or at your local library.
- Connecting to external applications and data sources is simpler using VB/VBA.
- VBA programs are relatively low maintenance; programs written for the last release of the AutoCAD program (even those written a decade ago) often run in the latest release with little to no change.

Learning the Fundamentals of the VBA Language

Before you learn to use VBA to automate the AutoCAD drawing environment, it is essential to have a basic understanding of the VBA or VB6 programming language. If you are not familiar with the VBA or VB6 programming language, I recommend reading this chapter before moving on.

In addition to this chapter, the Microsoft Visual Basic for Applications Help from the Help menu on the VBA Editor's menu bar and your favorite Internet search engine can be great

resources for information on the VBA programming language. The following are a couple of web resources that can help you get started on locating additional information on VBA and VB6:

- Microsoft's Programming Resources for Visual Basic for Applications page (`http://support.microsoft.com/kb/163435`)
- Microsoft Developer Network: Visual Basic 6.0 Language Reference (`http://msdn.microsoft.com/en-us/library/aa338033(v=vs.60).aspx`)

Creating a Procedure

Most of the code you write in VBA will be grouped into a named code block called a *procedure*. If you are familiar with AutoLISP or another programming language, you might be familiar with the terms *function* or *method*. VBA supports two types of procedures:

Subroutine (or Sub) A named code block that doesn't return a value

Function A named code block that does return a value

The definition of a procedure always starts with the keyword `Sub` or `Function` followed by its designated name. The procedure name should be descriptive and should give you a quick idea of the purpose of the procedure. The naming of a procedure is personal preference—I like to use title casing for the names of the functions I define to help identify misspelled function names. For example, I use the name `CreateLayer` for a function that creates a new layer. If I enter **`createlayer`** in the VBA Editor window, the VBA Editor will change the typed text to `CreateLayer` to match the procedure's definition.

After the procedure name is a balanced set of parentheses that contains the arguments that the procedure expects. Arguments aren't required for a procedure, but the parentheses must be present. The `End Sub` or `End Function` keywords (depending on the type of procedure defined) must be placed after the last code statement of the procedure to indicate where the procedure ends.

The following shows the basic structures of a `Sub` procedure:

```
Sub ProcedureName()

End Sub

Sub ProcedureName(Arg1 As DataType, ArgN As DataType)

End Sub
```

Here's an example of a custom procedure named `MyDraftingAids` that changes the values of two system variables—`osmode` to `35` and `orthomode` to `1`.

```
Sub MyDraftingAids()
  ThisDrawing.SetVariable "osmode", 35
  ThisDrawing.SetVariable "orthomode", 1
End Sub
```

When defining a procedure of the `Function` type, you must indicate the type of data that the procedure will return. In addition to indicating the type of data to return, at least one code statement in the procedure must return a value. You return a value by assigning the value to the procedure's name.