

CHAPTER

1

**VBA and You:**  
How Can You Get the  
Most Out of Microsoft  
Office?

**Chapter Objectives**

- **Learn VBA history**
- **Understand why to use VBA**
- **Understand programming concepts**
- **Use the Word VBA IDE**
- **Use the Excel VBA IDE**
- **Record macros**
- **Manipulate macros**
- **Implement macros**
- **Create programming solutions using VBA**

## chapter case

# Daggitt Development

During her summer internship, Maggie, a junior at the university, wants to learn more about using computer applications in business. Luckily, she gets an internship at Daggitt Development (DD for short), a local consulting company that develops software applications for small businesses. DD provides customized software to businesses for new payroll applications, electronic appointment books, or any other type of office tool.

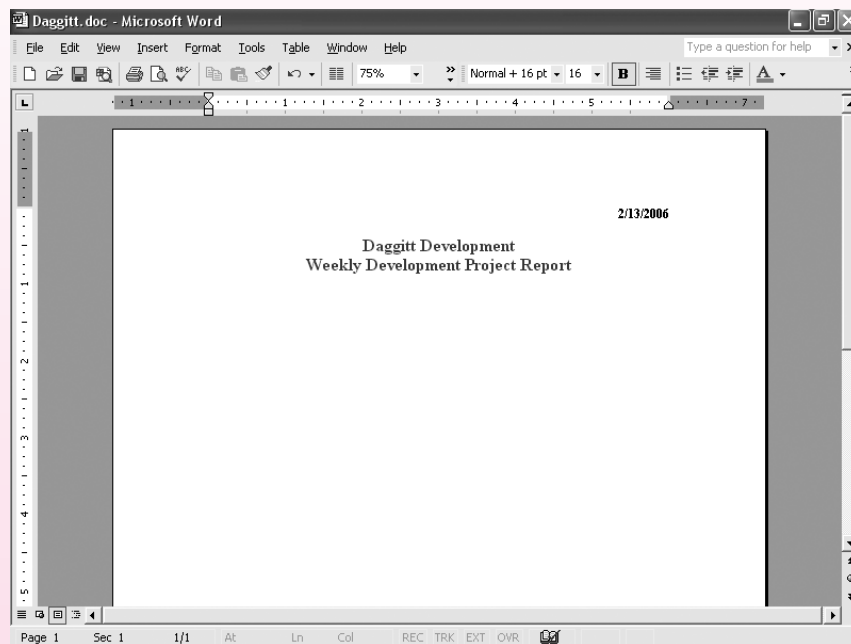
Cody, a senior programmer, tells Maggie that DD's most demanded service is the creation of macro programs for businesses to use within Microsoft Office. To do this, DD uses Visual Basic for Applications, or VBA. Using VBA, DD can tailor Microsoft Word, Excel, Access, PowerPoint, and Outlook to meet a customer's needs. Cody notes that DD transforms aspects of horizontal software into vertical software. Although Maggie is not sure what this means, Cody assures her she will learn soon enough.

Because Maggie has experience with Microsoft Office applications, Cody is ready to start her on her first project. Maggie will develop a customized macro in Microsoft Office for a local business, Bob's Baklava. This macro is similar to the one DD uses to create the heading on its Weekly Development Project Report (see Figure 1.1). Cody tells her not to worry that she hasn't used VBA before. He knows that she can learn how to use VBA as she works on her project.

Maggie's first tasks are to learn about VBA and programming concepts, become familiar with the VBA Integrated Development Environment (IDE), and learn how to record, edit, and run VBA macros. This chapter will help both you and Maggie become better prepared to customize applications and get the most out of Microsoft Office.

## Introduction

Chapter 1 introduces you to VBA programming and why people use it to customize applications. You'll also learn some of the history behind VBA and how to think and write like a programmer. We'll spend time touring the VBA IDE so that you are familiar with this programming environment. Finally, you'll learn how to record, manipulate, and implement VBA macros to customize Microsoft Word and Excel.



**FIGURE 1.1**  
Daggitt Development's  
Weekly Development  
Project Report

## SESSION 1.1 INTRODUCTION TO VBA PROGRAMMING

Like Maggie, you may think you don't have the necessary programming knowledge and skills to create customized Microsoft Office applications. However, many companies use VBA because it's easy to learn and use, yet powerful enough to solve complicated problems. **VBA**, or **Visual Basic for Applications**, is a programming language that works within certain software applications. A **programming language** contains specific rules and words that explain the logical steps to solve a problem. Although VBA works within many software applications, it's primarily used in Microsoft Office applications: Access, Excel, Outlook, PowerPoint, and Word. VBA is the programming language of choice for end user development of Microsoft Office applications. **End user development** is when computer users (such as you) develop and maintain computer applications with little or no help from technical specialists. However, you still need some programming knowledge and skills to work with VBA. The more programming knowledge and skills you have, the more you'll be able to accomplish using VBA.

In this section we'll explain where VBA came from, share some general characteristics of programming languages, and finally explain why we need VBA in Microsoft Office.

### Where Did VBA Come From?

Before VBA was a programming language called BASIC. **BASIC** stands for Beginner's All-purpose Symbolic Instruction Code. Two professors from Dartmouth College—John Kemeny and Thomas Kurtz—created BASIC in 1964 so that students would have a simple language to learn how to program computers. BASIC's popularity grew and before long (in 1969) an eighth grader named Bill Gates started using it. Of course, this is the same Bill Gates who started a company called Micro-Soft with his friend Paul Allen in 1975. You can only imagine what happened after that.

### BASIC Evolution

During the 1970s, BASIC took on various forms as Gates and Allen applied—or ported—it to many different computer systems, such as Altair, Apple, Commodore, and Atari. A computer language is **portable** when it has the ability to work on a variety

## CHAPTER OUTLINE

- 1.1 Introduction to VBA Programming VBA 1.3**
- 1.2 Using the VBA Integrated Development Environment (IDE) VBA 1.8**
- 1.3 Recording and Manipulating Macros VBA 1.23**
- 1.4 Summary VBA 1.35**

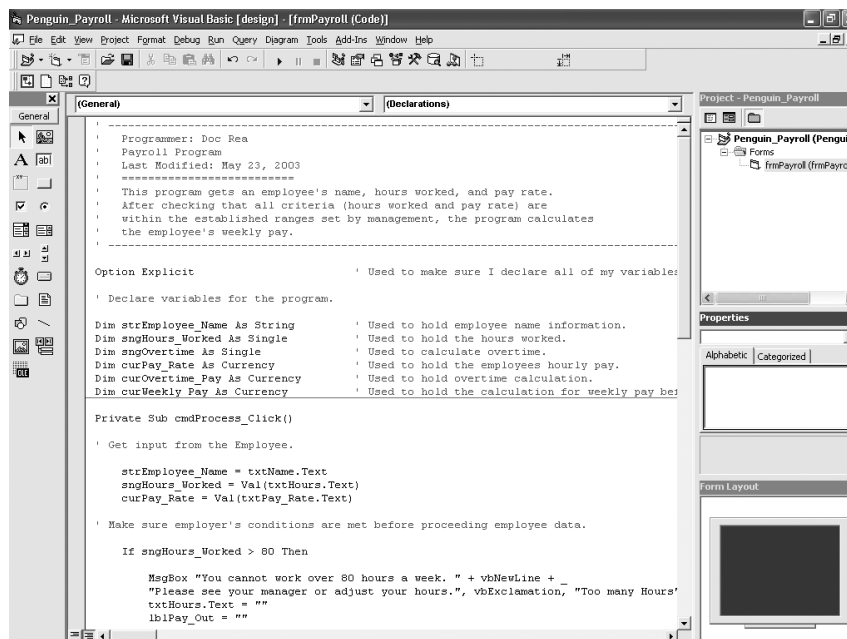
**VBA 1.3**

**VISUAL BASIC FOR APPLICATIONS**

of computers. Microsoft next developed an operating system called MS-DOS for IBM's first personal computer. It included a programming language called GW-BASIC. Microsoft was well on its way to becoming the powerful company it is today.

BASIC has many forms, but the most popular is probably Visual Basic. **Visual Basic** or **VB** is a graphical event-driven programming language. An **event-driven programming language** relies on actions and events to run. For example, an action or event occurs when you click on an icon or type a word. Visual Basic relies on a graphical user interface, such as Windows. A **graphical user interface**, or **GUI**, is a graphic- or icon-driven interface on which you point and click with your mouse to use software. Figure 1.2 is a comparison of the GW-BASIC programming environment with the VB integrated development environment (IDE). An **integrated development**

**FIGURE 1.2**  
Comparison of GW-BASIC  
and the VB IDE



*environment*, or *IDE*, is an application that provides programming tools to create, debug, and manage software programs. Notice how many options are available to the VB programmer as compared to the GW-BASIC programmer. More importantly, which one would you rather look at? Remember, these languages have the same source—BASIC—even though they're different.

## The Birth of VBA

Microsoft introduced Visual Basic in 1992. Programmers use VB to create software programs. A *programmer* is a specialist who writes software to meet users' needs. Also in 1992, Microsoft released its first version of the popular Microsoft Office productivity suite. But it wasn't until 1997, when Microsoft released Office 97, that Microsoft included VBA in nearly all of the Office applications: Access, Excel, PowerPoint, and Word. Finally, in Office 2000, Microsoft included VBA in Outlook. Microsoft Office XP and Microsoft Office 2003 include even more VBA functionality.

## What Is VBA?

When you program in Visual Basic, you're using a programming language to develop a complete software program that will run by itself. For example, you might create a game or accounting software. By contrast, when you use VBA, you're using a programming language that will create customized solutions within a software application, such as Microsoft Word. More specifically, VBA is a scripting language. A *scripting language* is a programming language that works within another application to perform tasks.

Maybe you want to allow users to customize how their Word documents will look. Or perhaps you want users to be able to track dinner reservations in an Excel spreadsheet without having to know how to use Excel. Or maybe you want to send formatted e-mail messages from a list of e-mail addresses in Access. With VBA, you can do all of this.

## Why Use VBA?

You know that most businesses use Microsoft Office because it meets many of their everyday business needs. Business users need to type letters and memos (Microsoft Word), manage budgets and figures (Microsoft Excel), send and receive e-mail (Microsoft Outlook), and give presentations (Microsoft PowerPoint). Some business users also need to keep track of inventories or other data (Microsoft Access). Microsoft Office can meet all of these needs for the majority of users because it's horizontal market software. *Horizontal market software* is general business software that has applications in many industries.

Although Microsoft Office does meet most business needs, a business often requires more specialized applications. One solution is to hire a consulting company to write a new software program. But many times a business can use VBA instead simply to customize Microsoft Office applications and solve the problem. VBA allows a business to create vertical market software out of a Microsoft Office application. *Vertical market software* is software that is unique to a particular industry. A business that uses VBA to customize an existing Microsoft Office application such as Excel, instead of hiring a consultant, saves a great deal of money.

## General Programming Concepts

Now you know VBA's history and why you should use it. You probably are eager to learn how to use VBA effectively. But first you'll need to understand some basic programming concepts as well as how programmers solve business problems so you can understand how to apply VBA to a business requirement.



**FIGURE 1.3**  
Maggie's pseudocode.

```

Start Program

  Open Word

  Save Word Document

  Start Macro Recorder
  Open New Document
  Insert Date
  Insert Spaces
  Insert Report Heading
  Make Heading Bold
  Align Heading Center
  Make Font color red
  Stop Macro Recorder

Stop Program

```

### Thinking Like a Programmer

As a programmer, your first priority is to know what a business needs. In this text, we'll help you pinpoint these needs. Figuring out a business problem takes practice, and you first must learn how to program to solve problems. Don't worry, though; we include some exercises that allow you to tackle this step on your own.

Once programmers understand the problem—such as a company's need for a payroll program to automatically print weekly paychecks—they map out the necessary steps to solve it. Programmers map out the problem in pseudocode. **Pseudocode** uses English statements to create an outline of the steps necessary for a piece of software to operate. Programmers call these steps an algorithm. An **algorithm** is a set of specific steps that solve a problem or carry out a task. An algorithm is like a

dessert recipe, in that a recipe lists all the steps necessary to create a scrumptious dessert. In programming, the sweet reward is a working piece of software.

You'll see examples of pseudocode throughout this text. We use it to describe each programming problem we approach. Figure 1.3 is an example of pseudocode Maggie wrote to help solve her first programming assignment in VBA. Notice that Maggie didn't type her pseudocode. Some programmers type pseudocode and some write it. We'll put this pseudocode to work for us later in the chapter.

Programmers also use program flowcharts to plot out the algorithm. A **program flowchart** is a graphical depiction of the detailed steps that a piece of software will perform. We'll use program flowcharts later in the text as we solve programming problems.

Once programmers write their algorithm in pseudocode or a program flowchart, they test it to make sure there are no logic errors. A **logic error** is a mistake in the way an algorithm solves a problem. For example, a payroll program is supposed to calculate overtime for anyone working more than 40 hours a week. If the program doesn't calculate overtime for someone working 50 hours a week, for example, it's a logic error.

### Writing Like a Programmer

Now that you're familiar with how programmers think, let's look at how they write. Programmers call the process of writing software coding. **Coding** is when you translate your algorithm into a programming language. Coding looks different depending on what type of programming language you're using. This is because each programming language has a specific syntax. **Syntax** is a set of rules to follow. We'll focus only on VBA syntax in this text. Figure 1.4 shows an example of VBA code.

Notice that some words appear in blue. These are reserved words. **Reserved words** are words that a programming language has set aside for its own use. In Figure 1.4, you'll notice `Dim` is colored blue. `Dim` is a reserved word VBA uses to create a variable. We'll discuss variables in detail in Chapter 2. Notice also that many lines start with a single apostrophe and are a different color (green). Programmers call these explanations comments. **Comments** tell other programmers what's happening in software code. The computer ignores comment lines when it runs code.

**tip:** Although our figures are not in color, note these colors as you work in your code throughout the book.

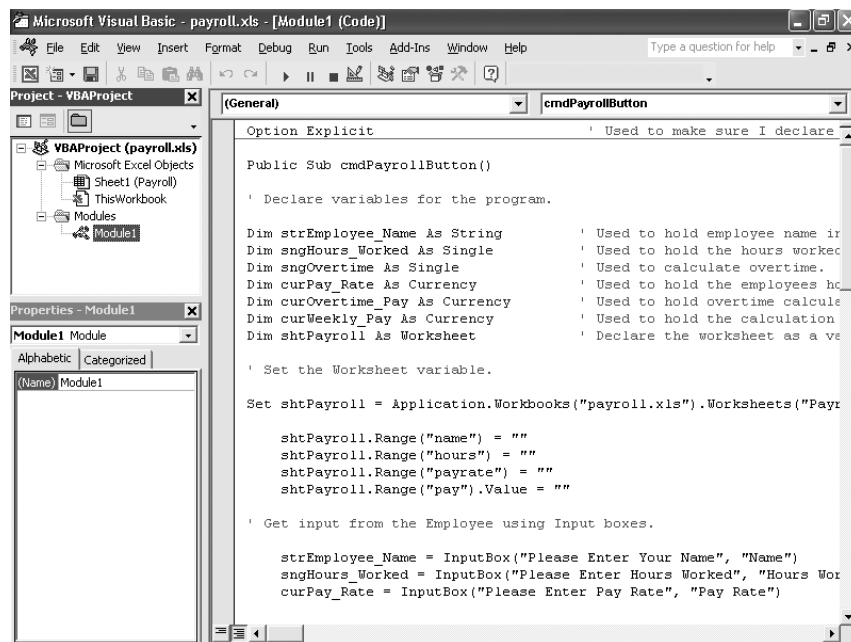


FIGURE 1.4

Notice how the VBA reserved words and comments are colored.

### Working Like a Programmer

You'll spend most of this text learning how to code in VBA, so we'll save the discussion of how to code until later. After programmers have coded a program, they spend some time debugging their code. **Debugging** is the process of finding errors in software code. **Bugs** are a common name for software errors. When you debug your code, you look for syntax and run-time errors.

**Syntax errors** are mistakes in a software code's grammar. Just as misspelling a word is a mistake when writing, misspelling a word or forgetting to mark a comment correctly will cause a syntax error. If you're supposed to use a semicolon (;) but you use a colon (:), you've made a syntax error. **Run-time errors** are mistakes that occur when you run the software code. Software not displaying a window correctly is a run-time error.

### VBA's Role in Microsoft Office

You are an experienced Microsoft Office user. Using Office applications to solve business problems comes easily to you. Why then do you need VBA to help you with Microsoft Office? The answer is simple: you can do more in a shorter period of time.

We've discussed how VBA can make a horizontal market application into a vertical market application. In the next session, we'll show you how you can tailor your Office applications using the macro recorder. A **macro** is a scripting language program that performs a task or a series of tasks. However, using VBA you can go beyond macros and create VBA programs that add functionality and features to Office applications. In this text you'll learn how to create these VBA programs.

Ultimately with Microsoft Office and VBA you can achieve the following:

- **Use the existing power of Microsoft Office** When you begin with an already powerful suite of software applications, you have a distinct advantage over programmers who must program every part of their software application.
- **Add features and functionality to applications quickly** Using VBA you can customize an existing application or create a new application by combining existing Office applications. For example, you can create "Employee of the Month" certificates in Word using data stored in Excel.

- **Create portable software solutions** Once you create a VBA program that works with a certain Office application, you can make minor changes and apply the same program to a similar business need. For example, once Daggitt Development creates a payroll application that works with Excel, it uses the same code as a starting point for its next customer who needs a payroll application.
- **Decrease development time and costs** Because code can be reused, programmers don't lose time creating software applications from scratch. Less time spent means less cost per program.
- **Increase end user involvement** Since business users are already familiar with Microsoft Office, they can help design solutions to their problems. Users will know what they need Excel to do because they can't do it easily with the current application.

If you'd like to learn more about VBA, check out [msdn.microsoft.com/vba](http://msdn.microsoft.com/vba).

## SESSION 1.1

### *making the grade*

1. \_\_\_\_\_ is a programming language that works within certain software applications.
2. A(n) \_\_\_\_\_ is an application that provides programming tools to create, debug, and manage software programs.
3. A(n) \_\_\_\_\_ is a programming language that works within another application to perform tasks.
4. \_\_\_\_\_ is the process of finding errors in software code.
5. A(n) \_\_\_\_\_ is a scripting language program that performs a task or a series of tasks.

## SESSION 1.2 USING THE VBA INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

With your new knowledge you're almost ready to start programming in VBA. Before you start coding, let's explore the application you'll work in—the VBA Integrated Development Environment (IDE). Remember, an IDE is an application that provides programming tools to create, debug, and manage software programs. You'll also hear this IDE called the VBA Editor. **VBA Editor** is simply another name for the VBA IDE. Newer versions of Microsoft VBA documentation refer to the VBA IDE. In this text we'll stick to the preferred term, VBA IDE, because it better describes what you can do with VBA development.

In this section you'll take a tour of the VBA IDE in Microsoft Word and Microsoft Excel. These are the application programs we'll use throughout the book as we develop VBA programs. In order to access the VBA IDE, we'll need to set macro security levels in Microsoft Office. Once we get into the VBA IDE, you'll learn how to use some of its features and functions. Then you'll record a macro and see how to manipulate and customize the macro code. Let's get started.

### Securing Microsoft Office

Before you work with the VBA IDE, we need to discuss macro security levels to protect your system against macro viruses. A **macro virus** is a computer virus hidden in macro code within a file or template. Once the security level is set for an Office application, such as Microsoft Word, all of the Word files on your computer system have the same





security level. You'll need to set the security level for each Office application. Because we'll use Microsoft Excel in this chapter, you should set its security level as well.

To reduce the risk of macro infection in Office files, your system may have the macro security level set to **High** or **Medium**. The **High** level prohibits macros from running with no notification, but if the security level is set to **High**, you'll not be able to run macros. The **Medium** level displays a dialog box each time a macro should run asking if you want to run the macros attached to the document. Set your security level to **Medium** so you can run our examples and your own macros. We do not recommend you set your macro level to **Low** because macro viruses regularly arrive at computer systems through infected files and cause many problems on your computer.

### *task reference*

- Open Microsoft Word
- Select **Security** in the **Tools** menu under **Macro**



### *Setting the Macro Security Level in Word:*

1. Click the taskbar **Start**  button to display the Start menu
2. Select **All Programs** to display the Programs menu
3. Select **Microsoft Word**  on the Programs menu
4. Click on **Microsoft Word**
5. Click **Tools** on the menu bar
6. Select **Macro**
7. Click on **Security**
8. Select the **Security** level tab
9. Click the radio button at **Medium** level
10. Click **OK**
11. Close Microsoft Word

### *task reference*

- Open Microsoft Excel
- Select **Security** in the **Tools** menu under **Macro**

### *Setting the Macro Security Level in Excel:*

1. Click the taskbar **Start**  button to display the Start menu
2. Select **All Programs** to display the Programs menu
3. Select **Microsoft Excel**  on the Programs menu

4. Click on **Microsoft Excel**
5. Click **Tools** on the menu bar
6. Select **Macro**
7. Click on **Security**
8. Select the **Security** level tab
9. Click the radio button at **Medium** level
10. Click **OK**
11. Close Microsoft Excel


### Opening the VBA IDE

In order to use the VBA IDE, you need to start a Microsoft Office application. You can run more than one VBA IDE at a time because each links to its application. Let's start with Microsoft Word.


#### task reference

- Open Microsoft Word
- Select the **Visual Basic Editor** in the **Tools** menu under **Macro**

### Opening the Word VBA IDE:

1. Click the taskbar **Start**  button to display the Start menu, then point to **All Programs** to display the Programs menu
2. Point to **Microsoft Word** on the Programs menu and then click **Microsoft Word**

**tip:** Make sure you have a single new blank document open. The title bar should read **Document1—Microsoft Word**

3. Click **Tools** on the menu bar and then select **Macro**. Click on **Visual Basic Editor** to start the VBA IDE
4. Check the title area. The title bar should read **Microsoft Visual Basic—Document1**
5. Close any open windows within the VBA IDE window by clicking the **Close Window**  button
6. Compare your screen to Figure 1.5. Your screen should now look similar to Figure 1.5


**tip:** Your screen might not look exactly the same. Make sure that you don't have any other Word documents open except for **Document1**. Also, make sure that you have closed all windows within the VBA IDE and have only the main window open

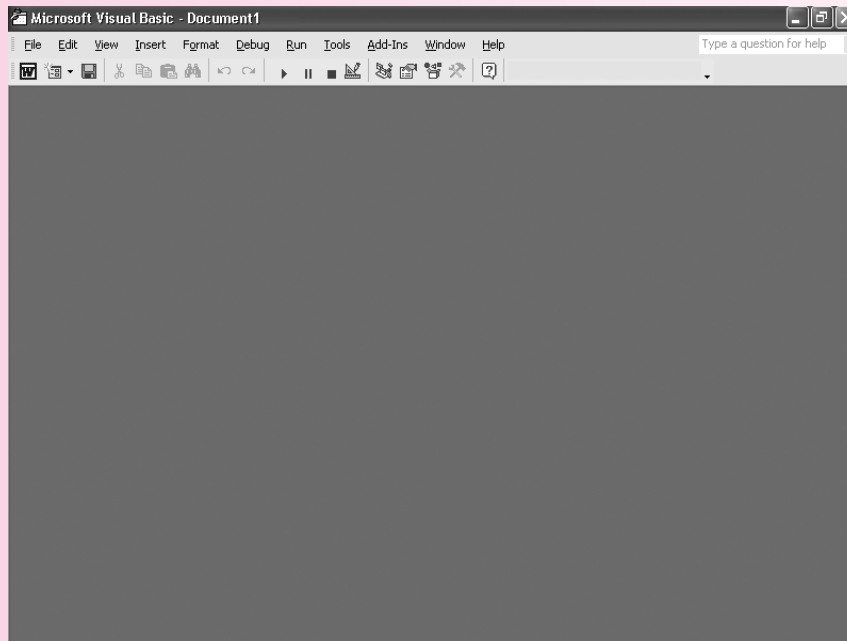
**tip:** To use the Visual Basic toolbar, click **View** on the menu bar and then click **Toolbars**. Select **Visual Basic**

#### anotherway ... to Open the VBA IDE

Press **Alt+F11**

**OR**

Click the **Visual Basic Editor**  button on the Visual Basic toolbar.



**FIGURE 1.5**  
VBA IDE with no open windows

## Touring the VBA IDE

Now that you're in the VBA IDE, let's get acquainted with some of its features. We won't cover everything right now, but we'll show you enough to get started. In this section you'll look at the main windows you'll use in the VBA IDE: the Project Explorer, the Properties Window, and the Code Window.

Before we look at the windows, look at the VBA Help Menu. The **Help Menu** is the most useful feature for learning VBA. Whenever you have a question or wonder why something isn't working in VBA, select the problem item and run Help.

### task reference

- Click on **Help**
- Select **Microsoft Visual Basic Help**

### Opening the VBA IDE Help Menu:

1. Click **Help** on the VBA IDE menu bar
2. Click **Microsoft Visual Basic Help**

**tip:** You should now see the Office Assistant appear and allow you to search for help. It's OK if you get a window labeled **Microsoft Visual Basic Help**. It contains the same help information

**anotherway**  
... to Use Help in  
the VBA IDE

Press F1


## Project Explorer

Now that we're in the VBA IDE, let's look at one of the windows you'll use when you code VBA programs—the Project Explorer. The **Project Explorer** window lists all objects associated with a VBA project. An **object** is an item that contains distinct information. In Chapter 2 we'll discuss objects and how they work within Office and VBA. You can access the Project Explorer with the click of a button.

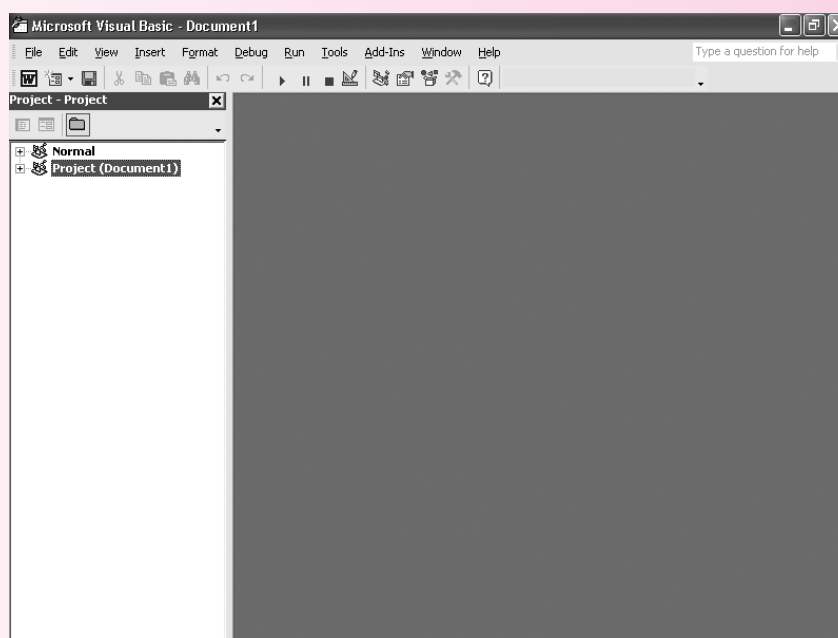
### task reference

- Open the **Project Explorer**

### Opening the VBA IDE Project Explorer:

1. Click on the **Project Explorer**  button on the toolbar
2. Compare your screen with Figure 1.6.

**FIGURE 1.6**  
VBA IDE with Project  
Explorer Directories  
Unexpanded





### anotherway ... to Open the Project Explorer


Select **View** and  
then click on **Project  
Explorer**

**OR**


Press **Ctrl+R**


If your screen doesn't look exactly like Figure 1.6, that's OK. VBA programmers can arrange windows within the VBA IDE in various ways. Check that the Project Explorer window contains the **Normal** and **Project (Document1)** objects as in Figure 1.6. Take a moment to explore the Project Explorer. Notice that the **expand**  button and the **collapse**  button work just like Windows Explorer.



### task reference

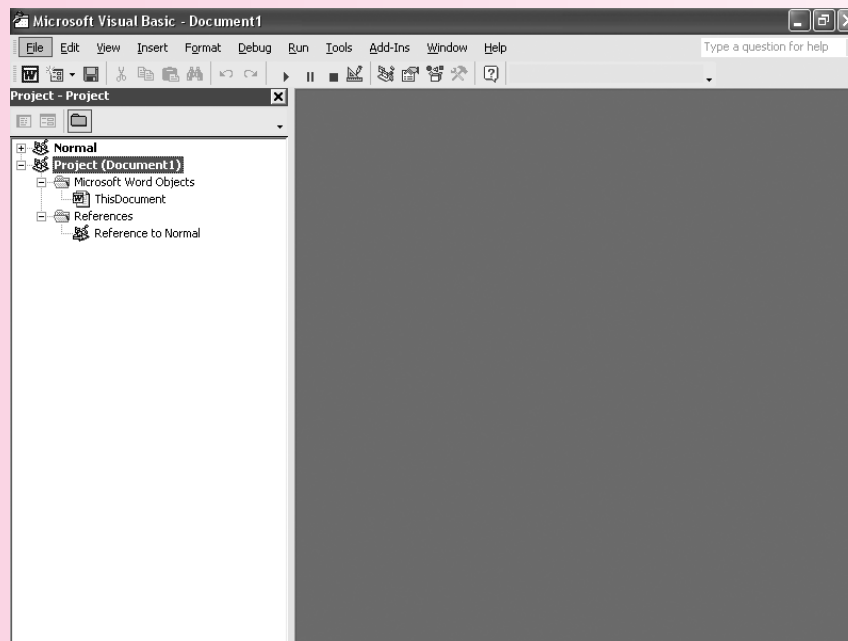
- Open the **Project Explorer**
- Click on the **expand**  button next to the folders

### Viewing Objects in the VBA IDE:

1. Click on the **expand**  button next to **Project (Document1)**

**tip:** If the **collapse**  button is already showing next to **Project (Document1)**, you don't need to click on it


2. Click on the **expand**  button next to **Microsoft Word Objects**
3. Click on the **expand**  button next to **References**
4. Compare your screen with Figure 1.7




**FIGURE 1.7**  
VBA IDE with Project Explorer Directories Expanded

Let's briefly discuss what you see. You'll work with many of these items as you go through the text. You may have first noticed the **Project (Document1)** heading. Of course, this is the actual Word document we opened. Because we haven't saved it yet, it's still called **Document1**. What do you think would happen if we saved our Word document? Let's try it and see.

#### task reference

- Click on the **Word**  button on the toolbar
- Press **Ctrl+S**

### Saving Objects in the VBA IDE:

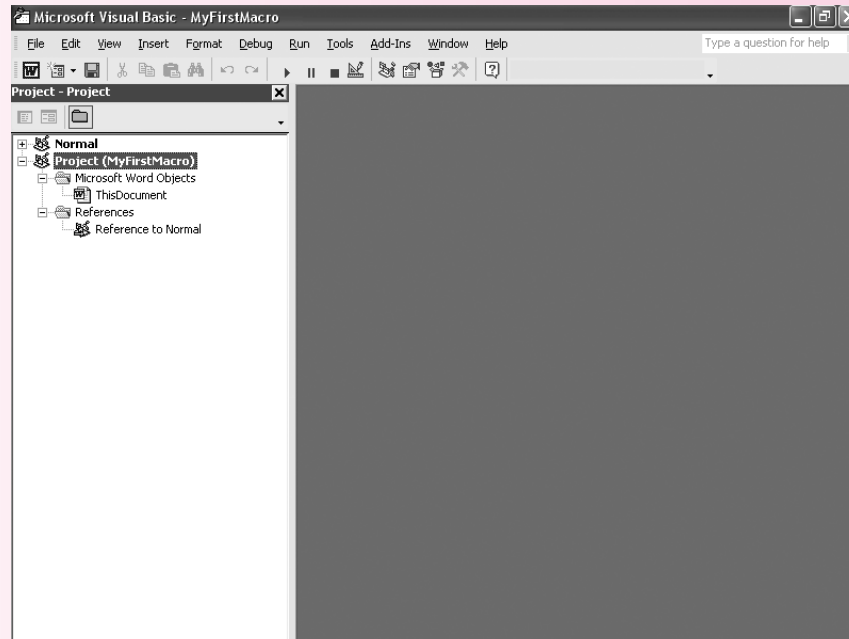
1. Click on the **Word**  button on the toolbar
2. Press **Ctrl+S**
3. Save the file as **MyFirstMacro**



4. Press **Alt+F11**

## 5. Compare your screen with Figure 1.8


**FIGURE 1.8**  
VBA IDE with renamed  
project called  
MyFirstMacro



Your **Project (Document1)** has become **Project (MyFirstMacro)**, and your Word application is the current project. You've actually changed a property within Microsoft Word. An **object property** is a characteristic of an object.

Under Microsoft Word Object you also see **ThisDocument**. Now that you know the Project contains your Word document, do you have any idea what **ThisDocument** is? It's your Word document. Since your Word document contains distinct information, it's an object. You'll work extensively on this object.

We won't do much with the Normal Project. You see another **ThisDocument** in the Microsoft Word Object here as well. However, you should know this is your default document template. This object is what determines your Microsoft Word settings (font type, size, page layout, etc.) for each new document you create. Notice that your **Project (MyFirstMacro)** is connected to the document template in the References area.

**tip:** To see everything included in the **Normal** template, click the **expand**  button next to **Normal**

## Properties Window

You can change many object properties in VBA. One way to do this is through the Properties Window. The **Properties Window** lists properties for each object in the VBA project. Let's take a look.

**task reference**

- Open the VBA IDE **Properties Window**
- Click on the object in the **Project Explorer**
- Change select properties in the **Properties Window**

**anotherway**

... to Open the  
**Properties Window**

Click the **Properties Window**  button

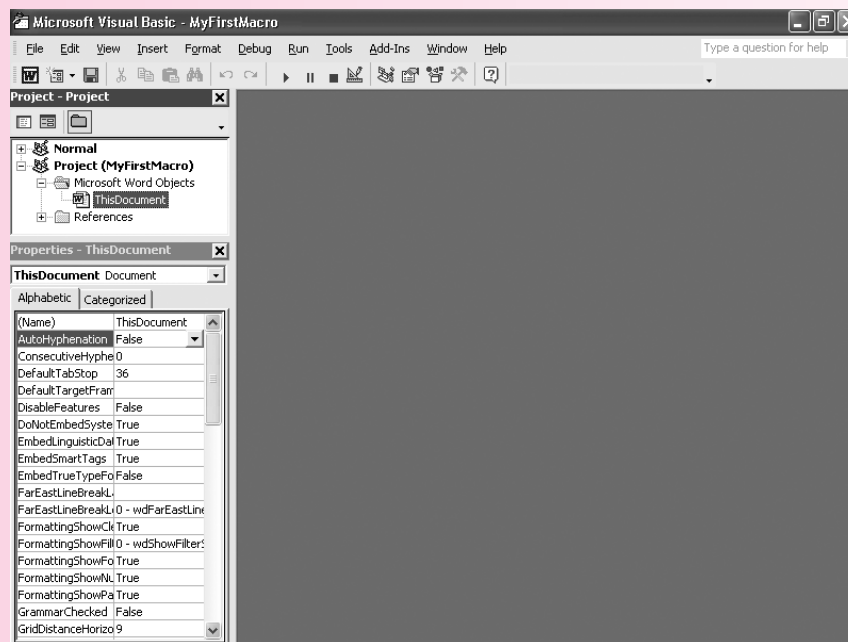
**OR**

Press **F4**

**Changing Object Properties in the VBA IDE:**

1. Click **View** on the menu bar
2. Click **Properties Window**
3. Click on **ThisDocument** under Project
4. Compare your screen with Figure 1.9

**tip:** You'll need to resize the **Properties Window** to see most of the properties at once



**FIGURE 1.9**  
VBA IDE with  
**ThisDocument** in Project  
Explorer and Properties  
Window

**tip:** It's OK if your screen doesn't exactly match Figure 1.9. Just make sure you have both the **Project Explorer** and the **Properties Window** visible

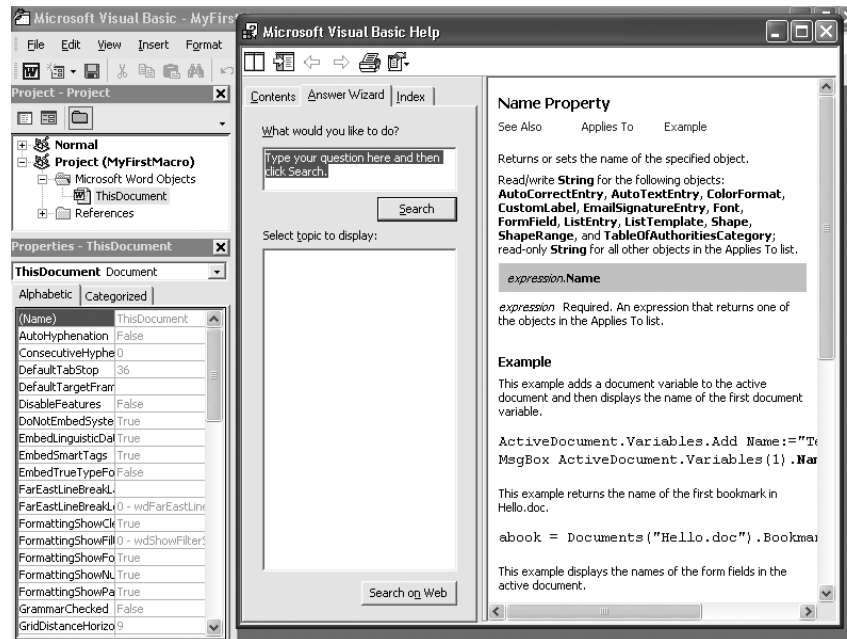
In the Properties Window you see the name of the object (**ThisDocument**) and the type of object (Document). You can also see the many properties associated with the **ThisDocument** document object. You can arrange them alphabetically or categorically. Most programmers choose the arrangement that best fits what they're doing. You'll learn which one works better for you. For now, click on any of the properties in the left-hand column and press **F1**. You'll see detailed information on the selected property. Figure 1.10 on page 1.16 shows the Help information for the **Name Property** and how you use it in a VBA program.

## VBA 1.16

## CHAPTER 1 VBA

## 1.2 Using the VBA Integrated Development Environment (IDE)

**FIGURE 1.10**  
VBA IDE with Name  
Property Help



You also see many VBA code examples in the Help area. You'll find these examples useful as we code VBA programs. For now, let's use the Properties Window to change the name of our project and our Word document object.

### task reference

- Open the VBA IDE **Properties Window**
- Click on the object in the **Project Explorer**
- Change select properties in the **Properties Window**

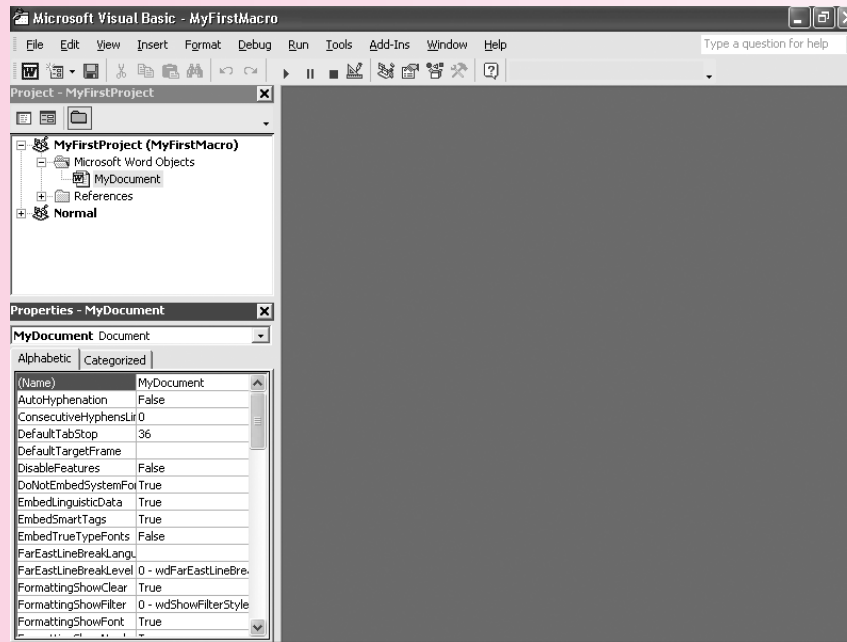
### Changing Object Properties in the VBA IDE:

1. Click on **Project (MyFirstMacro)** in the **Project Explorer**
2. Find the **(Name)** property in the **Properties Window**
3. Double-click on **Project** in the **Properties Window**
4. Type **MyFirstProject**
5. Press the **Enter** key
6. Find the **MyFirstProject (MyFirstMacro)** object in the **Project Explorer**
7. Click on **ThisDocument** under the **Microsoft Word Objects**
8. Find the **(Name)** property in the **Properties Window**
9. Double-click on **ThisDocument** in the **Properties Window**
10. Type **MyDocument**
11. Press the **Enter** key

12. Press **Ctrl+S**

13. Compare your screen with Figure 1.11

**tip:** Make sure you don't use spaces in your property names



**FIGURE 1.11**  
VBA IDE with changed  
name properties

## Code Window

Now that you've navigated the Project Explorer and are familiar with the Properties Window, let's look at the Code Window. The **Code Window** contains the VBA program code. You'll focus much of your attention in the Code Window as you work in VBA programs.

### task reference

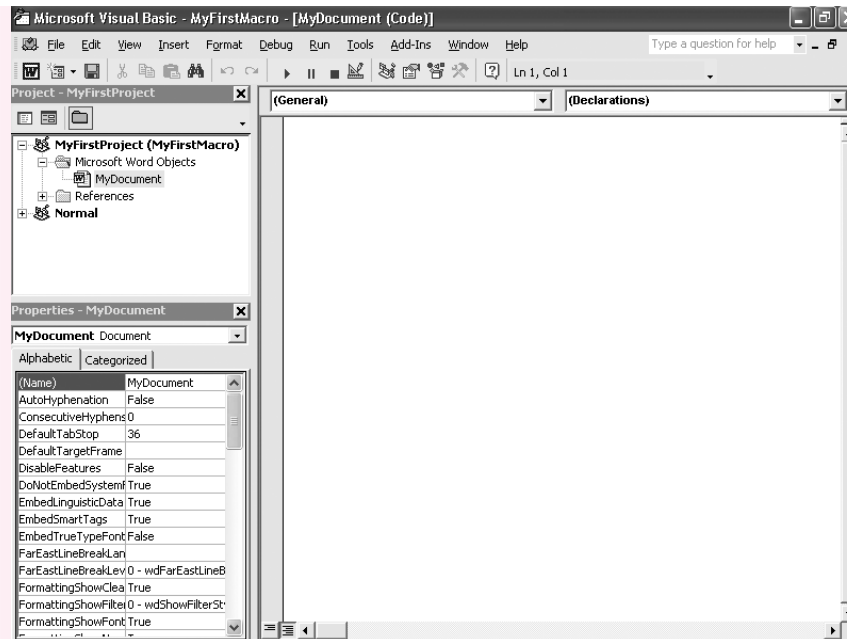
- Open the VBA IDE **Code Window**

### Opening the Code Window in the VBA IDE:

1. Click on **MyDocument** in the **Project Explorer**
2. Click **View** on the menu bar
3. Select **Code**
4. Compare your screen with Figure 1.12 on page 1.18

**tip:** Your **Code Window** may not appear in the same area on the screen depending on how your VBA IDE is configured. However, you should see the **Code Window**

**FIGURE 1.12**  
VBA IDE with all three  
main windows docked



### *another way* ... to Open the Code Window

Double-click  
**MyDocument** in the  
Project Explorer

**OR**

Single-click  
**MyDocument** in the  
Project Explorer and  
then press **F7**

Notice that the title bar in the Code Window reads **MyFirstMacro—MyDocument (Code)**. If you maximize the Code Window, this information will show in the VBA IDE title bar instead. If you see the words `Option Explicit` in the Code Window, that's OK. We'll discuss this in Chapter 2.

## Customizing the VBA IDE Windows

Before we work with some code in the Code Window, let's look at how you can customize your VBA IDE. Throughout the VBA IDE tour, your screen either looked like the figures in the text or it didn't. If your screen looked like our VBA IDE, that's because your VBA IDE was set with the default settings. However, you can change them if you want.

### Docking and Undocking Windows

In all of our figures, you noticed that VBA IDE always placed the Project Explorer and the Properties Window in the same place every time we opened them. However, you can dock or undock VBA IDE windows. A **docked window** is attached to the side of the VBA IDE. An **undocked window** floats inside the VBA IDE and is positioned anywhere. Programmers also call undocked windows **floating windows**. Just think of it in terms of a boat. When you anchor a boat to shore, it's docked. If you remove the anchor, it floats away.

Let's undock the Project Explorer and Properties Window.




### *task reference*

- Right-click on either the **Properties Window**, **Project Explorer**, or **Code Window**
- Click on **Dockable**



### Docking and Undocking Windows in the VBA IDE:

1. Right-click on the **Project Explorer** title bar
2. Click on **Dockable**
3. Right-click on the **Properties Window** title bar
4. Click on **Dockable**
5. Compare your screen with Figure 1.13

**tip:** Once you undock windows you can **minimize** , **maximize** , or **restore**  each window to your own preferences. Make sure to keep track of where each window is. Many times one is simply behind the other in your VBA IDE

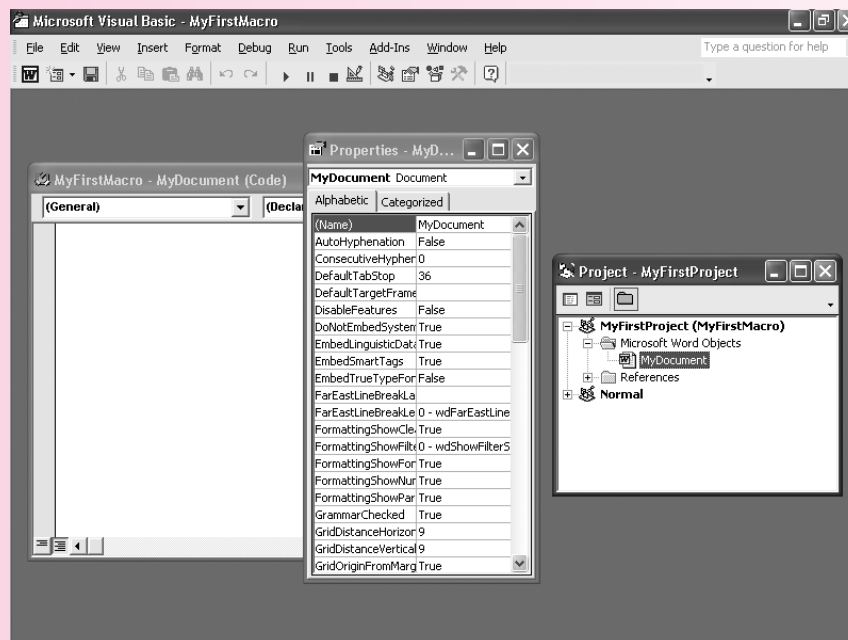


FIGURE 1.13

VBA IDE with all three main windows undocked

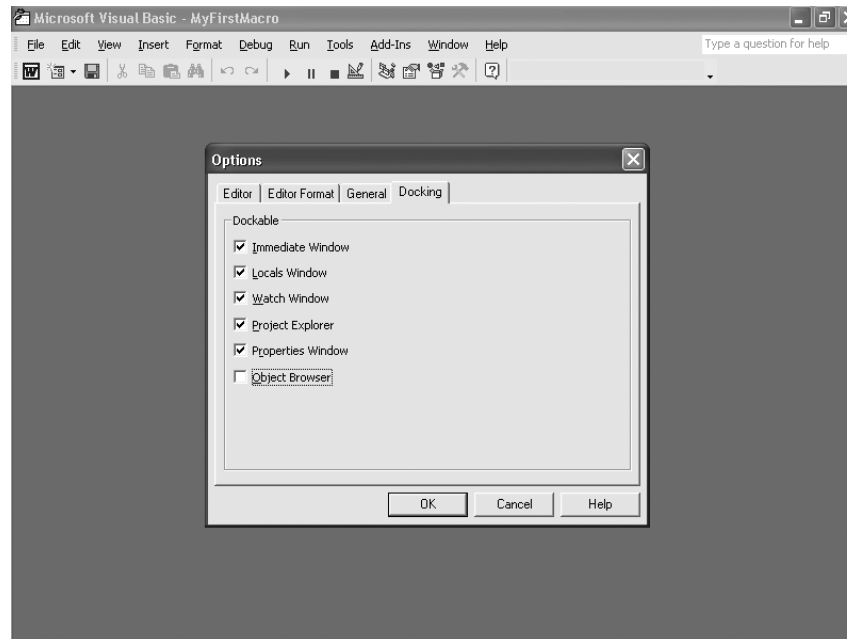
Let's dock the Project Explorer and Properties Window.

### Docking and Undocking Windows in the VBA IDE:

1. Click on **Tools** on the menu bar
2. Click on **Options**
3. Select the **Docking** tab
4. Check next to **Project Explorer**
5. Check next to **Properties Window**
6. Click **OK**

**tip:** Use the **Tools** menu to dock windows when you cannot right-click on a window or find a window in your VBA IDE

**FIGURE 1.14**  
VBA IDE Options window



### Other Customizations

You can choose many ways to customize your VBA IDE in the Options dialog box. Look at the tabs in Figure 1.14. You can change code and window settings with the **Editor** screen; we'll look at this in Chapter 2. Some programmers use the **Editor Format** screen to change how their VBA IDE displays information (we'll keep the defaults here). Finally, with the **General** screen you can change how the VBA IDE handles code errors and compiling. You'll learn more about this later.

Take some time to experiment with the VBA IDE windows. Make sure you're familiar with the interface to minimize confusion.

**tip:** You're done with Microsoft Word for this session. Make sure that you save your file by pressing **Ctrl+S** before you close Microsoft Word


### Exploring the Excel VBA IDE

Now that you've become familiar with the Word VBA IDE, you can more easily use other Office application IDEs. Since you and Maggie will develop VBA programs in both Excel and Word, let's take a look at the Excel VBA IDE.

#### **task reference**

- Open Microsoft Excel
- Select the **Visual Basic Editor** in the **Tools** menu under **Macros**

### Opening the Excel VBA IDE:

1. Click the taskbar **Start**  button to display the Start menu
2. Select **All Programs** to display the Programs menu
3. Point to **Microsoft Excel** on the Programs menu
4. Click **Microsoft Excel**

**tip:** Make sure you have a single new blank document open. The title bar should read **Microsoft Excel—Book1**

5. Click **Tools** on the menu bar
6. Select **Macro**
7. Select **Visual Basic Editor** to start the VBA IDE
8. Check the title area. The title bar should read **Microsoft Visual Basic—Book1**
9. Compare your screen to Figure 1.15

**tip:** Your screen might not look exactly the same. Make sure that you don't have any other Excel spreadsheets open except for **Book1**

### anotherway ... to Open the VBA IDE

Press **Alt+F11**

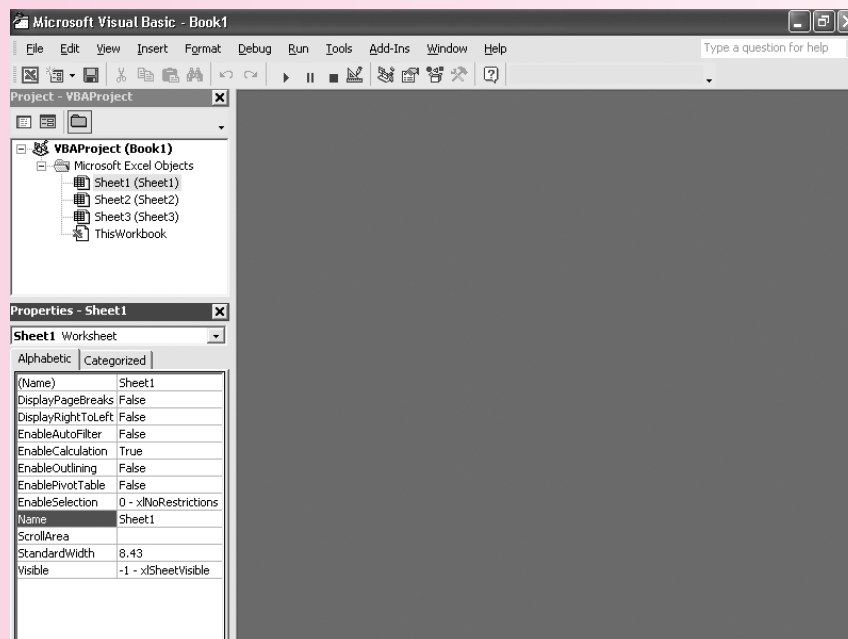




FIGURE 1.15

Excel VBA IDE with  
Project Explorer and  
Properties Window

### Comparing VBA IDEs


You should notice that the Excel VBA IDE looks similar to the Word VBA IDE. The menu bar and the toolbar are familiar, and you can see the Project Explorer, Properties, and Code Windows. You can navigate and customize this IDE just like the Word IDE. Ultimately, all the VBA IDEs in Office have these similar features; however, because this VBA IDE is part of a different Office application, it also has some unique features.

You should see differences in Figure 1.15 such as the title in the title bar. Notice the **Excel**  button instead of the **Word**  button on the toolbar. The content in the Project Explorer and the Properties Window is significantly different.


### Project Explorer

Instead of Word Objects, you now work with Excel Objects. **ThisWorkbook** has replaced **ThisDocument**, and you now see three Sheet objects (**Sheet1**, **Sheet2**, **Sheet3**). Just like **ThisDocument**, **ThisWorkbook** is the object that represents the current Excel Workbook. The Sheet objects represent the three sheets in the Excel Workbook. The more sheets you add, the more sheet objects appear. Let's insert and name a sheet and see what happens.

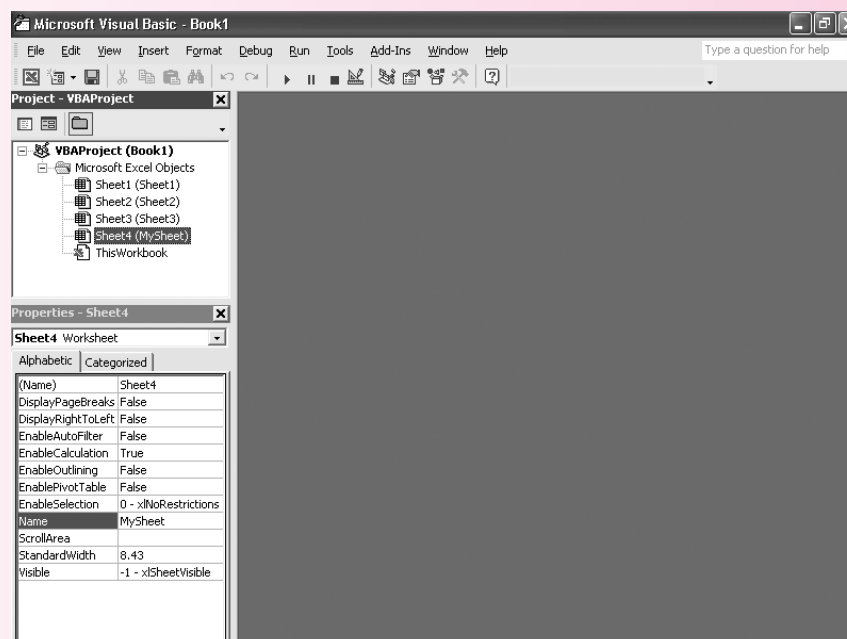
### task reference

- Click on the **Excel**  button
- Click on **Insert** in the menu bar and then **Worksheet**

### Inserting a Sheet into the Excel VBA IDE:

1. Click on the **Excel**  button
2. Click on **Insert** on the menu bar
3. Click on **Worksheet**
4. Double-click on the **Sheet4** tab
5. Type **MySheet**
6. Press **Enter**
7. Press **Alt+F11**
8. Compare your screen with Figure 1.16

**FIGURE 1.16**  
Excel VBA IDE with  
Project Explorer,  
Properties Window, and  
MySheet added



Notice that you now have a new Sheet object (**Sheet4**) named **MySheet**.

### Properties Window

Although the properties in the Properties Window are different than those you've seen in the Word VBA IDE, changing properties is exactly the same. You are familiar with changing the (**Name**) property on a Word object. Try changing this property on an Excel object.

You've spent a lot of time learning the VBA IDE in both Word and Excel. In the next session you'll harness the power of VBA by recording and customizing macros in Word.

**tip:** You're done with Microsoft Excel for this session. You don't need to save your file before you close Microsoft Excel

### making the grade

1. \_\_\_\_\_ is another name for the VBA IDE.
2. The \_\_\_\_\_ is the most useful feature for learning VBA.
3. The \_\_\_\_\_ window lists all objects associated with a VBA project.
4. The \_\_\_\_\_ lists properties for each object in the VBA project.
5. A(n) \_\_\_\_\_ is attached to the side of the VBA IDE.

### SESSION 1.2

## SESSION 1.3 RECORDING AND MANIPULATING MACROS

You learned in Session 1.2 that a macro is a scripting language that performs a task or a series of tasks. You can use these tasks to perform a set of commands simultaneously within an Office application. For example, you can change a font type, size, and color with one macro. You also can assign this macro to a button or a keystroke combination so you can repeatedly perform it. After you record a macro, you can look at the VBA code and see how VBA tells the application to perform the set of commands. Once you know this, you can manipulate the macro code within the VBA IDE to customize it even further. In this section we show you how.

### Recording Macros

You're already familiar with many functions and features in Office applications. For example, you should know how to insert a date in Microsoft Word. You can also align text by selecting it and then choosing how to align it in the formatting toolbar. And you already know many other ways to format a document.

Remember the Daggitt Weekly Development Project Report that Cody showed Maggie (see Figure 1.1 on page 1.3)? He created this using a macro. Every week department supervisors run this macro to create a Word document for the weekly development project report. Cody asks Maggie to create a similar macro for a client, Bob's Baklava. Let's create one, too.



### Using the Macro Recorder

Think of recording a macro as you would think of recording a speech or a song. To record these you need an audio or video recorder. Some of the Office Applications (Excel, Word, and PowerPoint) have a recorder. A *macro recorder* records a sequence of actions in an Office application that you can then repeat.

Just like any recorder, if you make a mistake while you're recording, you have two options: (1) start over from the beginning or (2) edit the mistake and correct it. You're not ready to edit mistakes in VBA yet, so if you make a mistake at any point when you're recording, close the document without saving it and start over.

#### task reference

- Click **Tools** on the menu bar
- Select **Macro**
- Click on **Record New Macro**

### Using the VBA Macro Recorder:

**tip:** Before recording, make sure you have the following Microsoft Word active toolbars: **Standard**, **Formatting**, and **Visual Basic**

1. Start Microsoft Word
2. Save your Word document as **BaklavaWeekly**
3. Click **Tools** on the menu bar
4. Select **Macro**
5. Click on **Record New Macro**
6. Type **BaklavaMacro** in the **Macro name:** field

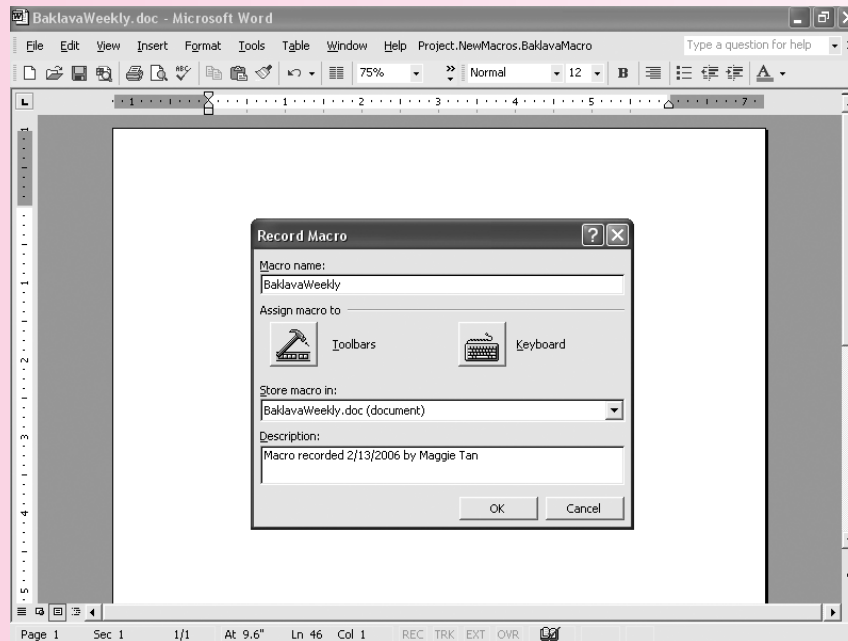
**tip:** Make sure you don't use spaces in your macro names

7. Select **BaklavaWeekly** in the **Store Macro in:** field

**tip:** Don't select the **All Documents (Normal.dot)** option. This would place your macro in all newly created Word documents. Of course, if you wanted to have this macro available for all Word documents, you would select this option

8. In the Description field place **<your name>** at the end of the sentence
9. Compare your screen with Figure 1.17
10. Click **OK**

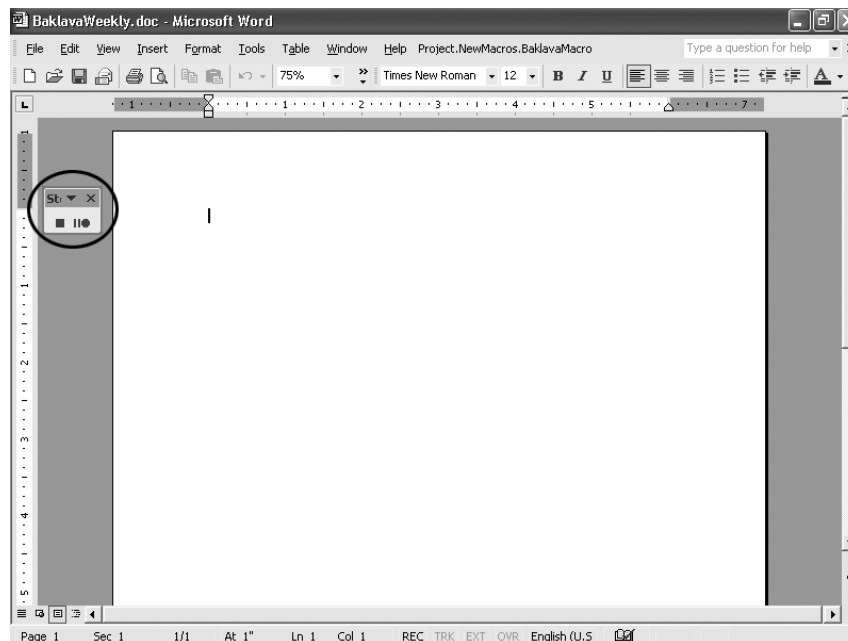
**tip:** Don't do anything else at this point. The macro recorder is active. All keystrokes and functions will be recorded. You'll record your macro in the following "Recording the Macro" steps



**FIGURE 1.17**  
Word Record Macro  
Screen

## Recording the Macro


After you clicked **OK** in the steps above, you turned on the macro recorder. You should now see a blank Word document (**BaklavaWeekly.doc**) with the macro recorder showing on the screen. Make sure your screen looks similar to Figure 1.18.



**FIGURE 1.18**  
Word Document with  
macro recorder

**tip:** If you don't see the macro recorder buttons, check in the toolbar. Depending on your Microsoft Word configuration, it might be in the toolbar. The buttons look the same

### *anotherway* ... to Record a Macro

Click the **REC**  button in the **status bar**

After you're sure the macro recorder is on, start recording your macro. Make sure to follow the steps exactly. If you make a mistake, close the document without saving it and start again.





### task reference

- Choose **Record New Macro**  under the **Tools** menu
- When finished, press the **Macro Recorder stop**  button

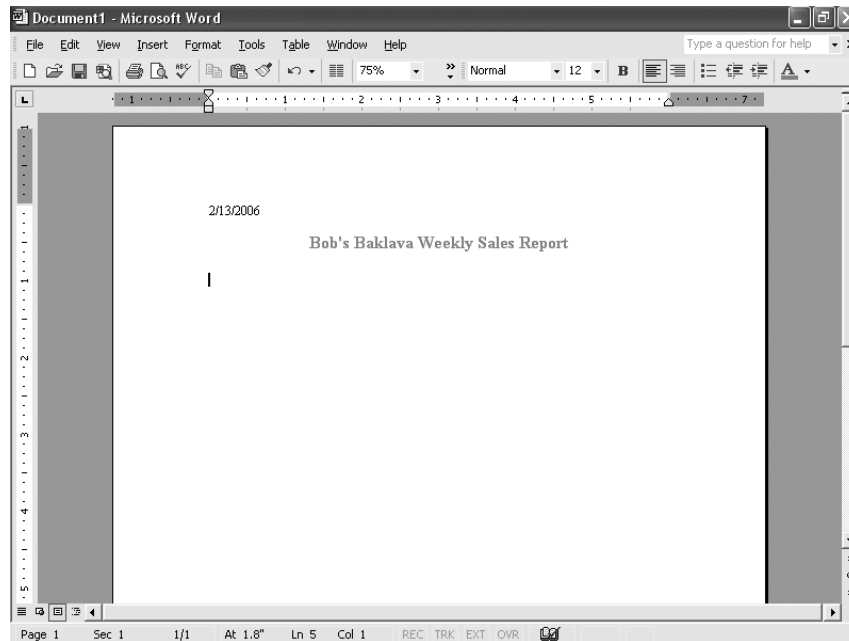
### Recording a VBA Macro:

1. Press **Ctrl+N**
2. Click **Insert** on the menu bar and then **Date and Time**
3. Select the **month/day/year** format

**tip:** The month/day/year format will show the current date. For example, if it's February 13, 2006, your system will show **2/13/2006**

4. Click on **Update Automatically**
5. Click **OK**
6. Press **Enter**
7. Press **Enter**
8. Type **Bob's Baklava Weekly Sales Report**
9. Press and hold down the **Shift** key and use the left arrow key to highlight **Bob's Baklava Weekly Sales Report**
10. Click the **Bold**  button on the formatting toolbar
11. Click the **Center Align**  button on the formatting toolbar
12. Click on the **Font Color**  button on the formatting toolbar and select **red** (first column, third item)
13. Click the **Stop Recording**  button on the macro recorder
14. Click anywhere on the Word document
15. Compare your screen with Figure 1.19
16. Close your new document without saving it
17. Press **Ctrl+S** to save your **BaklavaWeekly** file and then close it

Notice that your formatted document is a new document and not your **BaklavaWeekly.doc** file. Because you opened a new document right after you started macro recording, you can reuse the **BaklavaWeekly.doc** file to create a new report each week. And once Bob enters his weekly sales he can save the file using any name except **BaklavaWeekly.doc**.




**FIGURE 1.19**  
Completed Word  
document with macro

### Running a VBA Macro

Now that you've recorded your macro, you'll need to run it. There are a few ways to do this. Let's look at the easiest way first.

#### *task reference*

- Select **Macros** from the Tools menu bar
- Choose the macro
- Press the **Run**  button

### Running a VBA Macro:

**tip:** When you open *BaklavaWeekly.doc*, Microsoft Word will issue a security warning because your file contains a macro. Make sure to click **Enable Macros** on the dialog box

1. Open **BaklavaWeekly.doc**
2. Click **Enable Macros**
3. Click **Tools** on the menu bar
4. Select **Macro**
5. Click **Macros**
6. Make sure **BaklavaMacro** is selected
7. Press **Run**
8. Compare your result with Figure 1.19
9. Close the new file without saving it

***anotherway***  
... to Open the  
Macros Dialog Box  
Press **Alt+F8**

You've now run your `BaklavaMacro` and created a new weekly sales report.

You can also assign macros to keystrokes and menus, as well as assign a button to make them work. We'll explore all of these methods as we delve deeper into VBA programming.




## Manipulating Macros

When you record a macro, it uses VBA to accomplish the tasks. However, a macro can only record actions available within an application. A VBA program can help programmers add features and functions beyond what is available in any one application. However, you can use a macro as a building block for more extensive macros or VBA programs. We'll add some features to the `BaklavaMacro` you just recorded, but before we do, let's take a look at the macro.

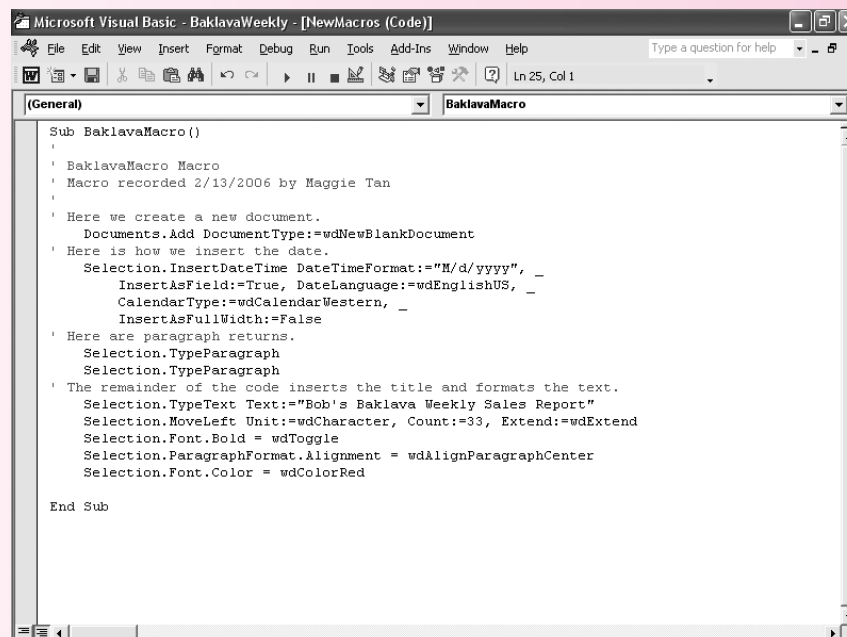
### task reference

- Open the VBA IDE
- Click on the object containing the macro
- Press **F7** to open **Code Window**

### Reading a VBA Macro:

1. Select your `BaklavaWeekly` document
2. Press **Alt+F11**
3. Click on the **Project Explorer**  button if your Project Explorer isn't open
4. Click on the **expand**  button next to the **Modules** folder
5. Double-click the **NewMacros**  object
6. Compare your screen with Figure 1.20

**FIGURE 1.20**  
BaklavaMacro code in  
VBA IDE



```

Sub BaklavaMacro ()
' BaklavaMacro Macro
' Macro recorded 2/13/2006 by Maggie Tan
'
' Here we create a new document.
Documents.Add DocumentType:=wdNewBlankDocument
' Here is how we insert the date.
Selection.InsertDateTime DateTimeFormat:="M/d/yyyy", _
InsertAsField:=True, DateLanguage:=wdEnglishUS, _
CalendarType:=wdCalendarWestern, _
InsertAsFullWidth:=False
' Here are paragraph returns.
Selection.TypeParagraph
Selection.TypeParagraph
' The remainder of the code inserts the title and formats the text.
Selection.TypeText Text:="Bob's Baklava Weekly Sales Report"
Selection.MoveLeft Unit:=wdCharacter, Count:=33, Extend:=wdExtend
Selection.Font.Bold = wdToggle
Selection.ParagraphFormat.Alignment = wdAlignParagraphCenter
Selection.Font.Color = wdColorRed

End Sub

```

**tip:** We added comments to the VBA macro to explain the code. Your recorded macro won't contain comments

When you look at your VBA IDE screen you should notice a new folder in the Project Explorer called **Modules**. We haven't seen this folder or the NewMacros object stored in it yet, but the name gives it away. This macro module contains your BaklavaMacro macro. The **macro module** contains all the macro code in a particular project.

Although at first the VBA code in this macro can be hard to read, think about the steps you took to create the macro. Look at the code in Figure 1.20. We've added comments to help explain the code. You'll need to match each piece of code with an action as you work in the VBA IDE design mode so you don't always need to depend on macros. The **design mode** is the VBA area where you'll work with code and use VBA programming tools. Let's look at some code that isn't included in the macro steps, but is just as important.

The Sub `BaklavaMacro()` starts the macro code. If you look at the bottom of the code, you'll see an `End Sub`. These lines surround the procedure. A **procedure** is a section of a program that performs a particular task. In this case, the procedure is your macro. You'll also notice the green text marked with an apostrophe beginning each line. These are the program comments you learned about in Session 1.1. You can add as many comments as you like, and we'll discuss how best to do this in Chapter 2.

## Changing the Code

Right now your BaklavaMacro opens a new document, inserts the current date, and then places the words **Bob's Baklava Weekly Sales Report** centered on the page in a bold, red font. Let's change it so the macro places the date aligned right and makes the text bold, italicized, and green.

Before you begin, make sure you've turned on an important feature called IntelliSense. **IntelliSense** is a Microsoft feature that completes pieces of programming code with the correct syntax. If you have IntelliSense turned on, your VBA IDE will suggest ways to complete your programming statements.

### task reference

- Open the VBA IDE
- Turn on **IntelliSense**

### Using IntelliSense:

1. Click on **Tools** on the menu bar
2. Click on **Options**
3. Select the **Editor** tab
4. Check the **Auto List Members** box
5. Click **OK**


Now that you've turned on IntelliSense, let's manipulate some of the VBA code.

**task reference**

- Open the VBA IDE
- Select the object
- Press **F7** to open **Code Window**

**Revising a VBA Macro:**

1. Find the line `Selection.ParagraphFormat.Alignment = wdAlignParagraphCenter`
2. Find the first line `Selection.TypeParagraph` and place your cursor at the beginning of the line
3. Press the **Enter** key to add a space
4. Place your cursor on the empty line and refer back to the line noted in step 1
5. Slowly type the code from step 1

**tip:** Type slowly. Every time you enter a " ." you will see the IntelliSense technology give you a list of options. You can use the down arrow  to move down the list. Or you can type the word you want and the list will change the closer you get to the word. To select an item use your **Tab** key

6. Instead of choosing the `wdAlignParagraphCenter`, choose the option that will align text to the right (`wdAlignParagraphRight`)
7. Move to the line `Selection.Font.Bold = wdToggle` and insert a line underneath it
8. Type the text `Selection.Font.Italic = wdToggle`
9. Place your cursor to the right of the line `Selection.Font.Color = wdColorRed`
10. Delete `=wdColorRed`
11. Place your cursor to the right of `Selection.Font.Color`
12. Type `=`
13. Select `wdColorDarkGreen` from the **IntelliSense** box
14. Press the **Enter** key
15. Type `Selection.Font.Size = 16`
16. Press **Ctrl+S**
17. Compare your code to the code in Figure 1.21

After you're sure your code matches, run your macro and see your changes.



```


Microsoft Visual Basic - BaklavaWeekly - [NewMacros (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Type a question for help
Ln 27, Col 1
(General) BaklavaMacro

Sub BaklavaMacro()
    ' BaklavaMacro Macro
    ' Macro recorded 2/13/2006 by Maggie Tan
    '
    ' Here we create a new document.
    Documents.Add DocumentType:=wdNewBlankDocument
    ' Here is how we insert the date.
    Selection.InsertDateTime DateTimeFormat:="M/d/yyyy", _
        InsertAsField:=True, _
        DateLanguage:=wdEnglishUS, CalendarType:=wdCalendarWestern, _
        InsertAsFullWidth:=False
    ' Here is a paragraph alignment and paragraph returns.
    Selection.ParagraphFormat.Alignment = wdAlignParagraphRight
    Selection.TypeParagraph
    Selection.TypeParagraph
    ' The remainder of the code inserts the title and formats the text.
    Selection.TypeText Text:="Bob's Baklava Weekly Sales Report"
    Selection.MoveLeft Unit:=wdCharacter, Count:=33, Extend:=wdExtend
    Selection.Font.Bold = wdToggle
    Selection.Font.Italic = wdToggle
    Selection.ParagraphFormat.Alignment = wdAlignParagraphCenter
    Selection.Font.Color = wdColorDarkGreen
    Selection.Font.Size = 16
End Sub

```

**FIGURE 1.21**  
Revised BaklavaMacro  
code in VBA IDE


### task reference

- Select **Macros** from the Tools menu bar
- Choose the macro
- Press the **Run**  button

### Running a VBA Macro:

1. Click anywhere between the Sub BaklavaMacro() and End Sub in the VBA code

**tip:** If you don't click within the macro's code, the VBA IDE macro window will appear and you'll need to select a macro to run

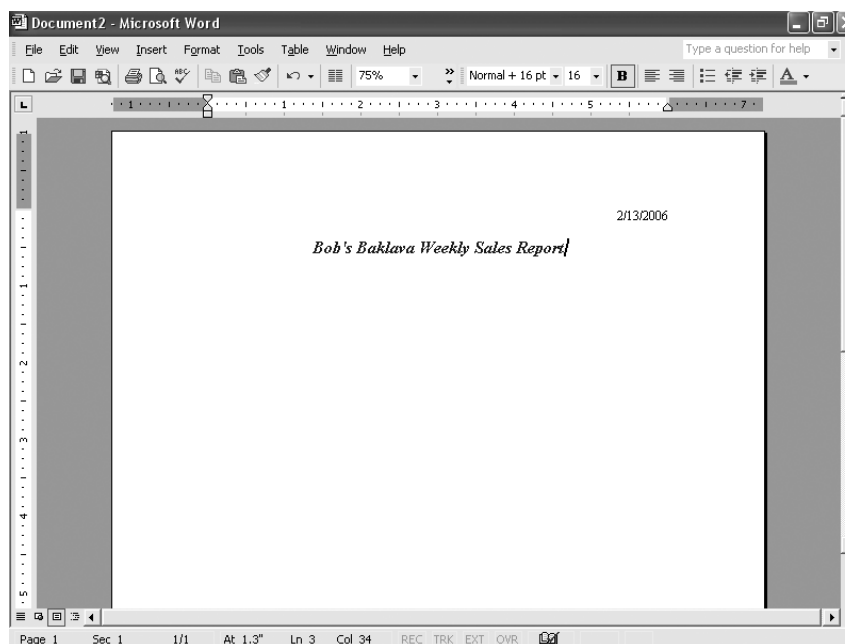
2. Press the **Run Sub/UserForm**  button
3. Press **Alt+Q**
4. Click on the new document screen
5. Compare your screen with Figure 1.22 on page 1.32

**tip:** You can view the results of your macro without closing the VBA IDE. Press **Alt+ Tab** to toggle between the VBA IDE and Microsoft Word document

### Assigning Macros

Now that your macro is working as you'd like, you'll want to make it easy for another user to run. You could provide detailed steps on how to run a macro from the Tools menu. However, part of a programmer's job is to make sure end users can use the

**FIGURE 1.22**  
Revised BaklavaMacro  
results



application. Asking end users to navigate menu options to run your macro affects the program's usability. **Usability** refers to how easy or how difficult it is to use a program.

Instead of relying on the menu, you can create a button that will allow users to run your macro by clicking it. You'll learn how to customize these command buttons later in the text. For now, let's use a VBA macro button. The VBA **macro button** provides users with a one-click option to run a macro.

### task reference

- Click on **Tools** in the menu bar
- Select **Customize . . .**
- Assign a macro to a button

### anotherway

#### . . . to Customize your Application

1. Click on the **Toolbar Options** button on the toolbar
2. Click on **Add or Remove Buttons**
3. Click on **Customize . . .**

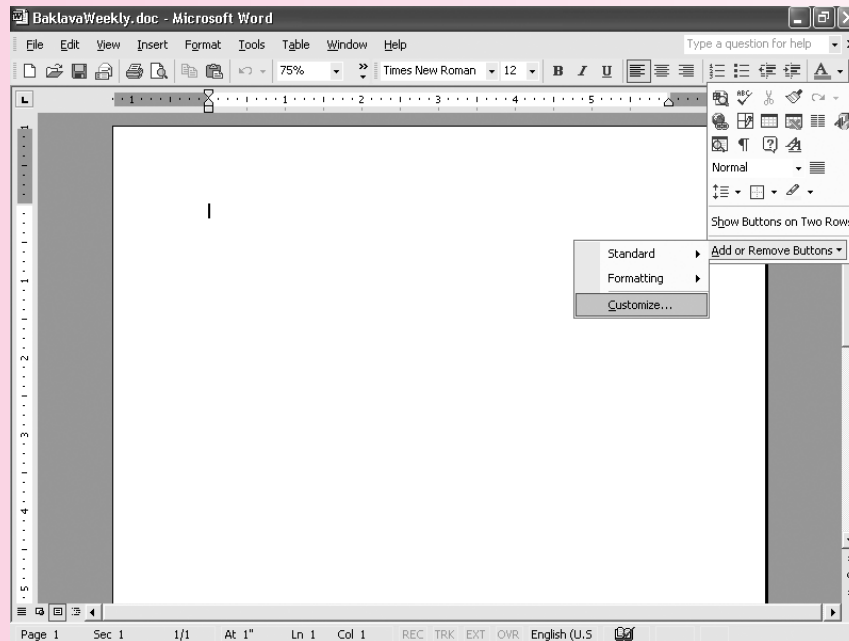
### Assigning a VBA Macro:

1. Toggle to your **BaklavaWeekly** Word document
2. Click on **Tools** on the menu bar
3. Click on **Customize . . .**

**tip:** If you're having trouble finding the **Toolbar Options** button, look on the far right of the screen. Figure 1.23 shows the options to look for



4. In the **Customize** dialog box click on the **Commands** tab
5. In the **Customize** dialog box, scroll down the **Categories:** on the left-hand side and click on **Macros**
6. Under **Commands**, click on **Project.NewMacros.BaklavaMacro**

**tip:** If you see **Normal.NewMacros.BaklavaMacro** make sure to select the **BaklavaWeekly.doc** in the **Save In** area at the bottom of the dialog



**FIGURE 1.23**  
Customize toolbar options

7. Drag the macro button up next to the **Help** menu on the menu bar

**tip:** If you have trouble placing the macro button, make sure your icon looks like this . If your icon looks like this , you cannot place the macro button

8. Release the macro button
9. In the **Customize** dialog box click on the **Modify Selection** button
10. Click **Name** in the list
11. Type **BacklavaMacro**
12. Click **Image and Text** in the list
13. Click **Close** on the **Customize** dialog
14. Press **Ctrl+S**
15. Compare your screen with Figure 1.24 on page 1.34

## Printing Macros

Cody asks Maggie to print her macro so he can go over it with her. You should print yours out for future use as well. Printing a macro code module is as easy as printing a Word document.

### task reference

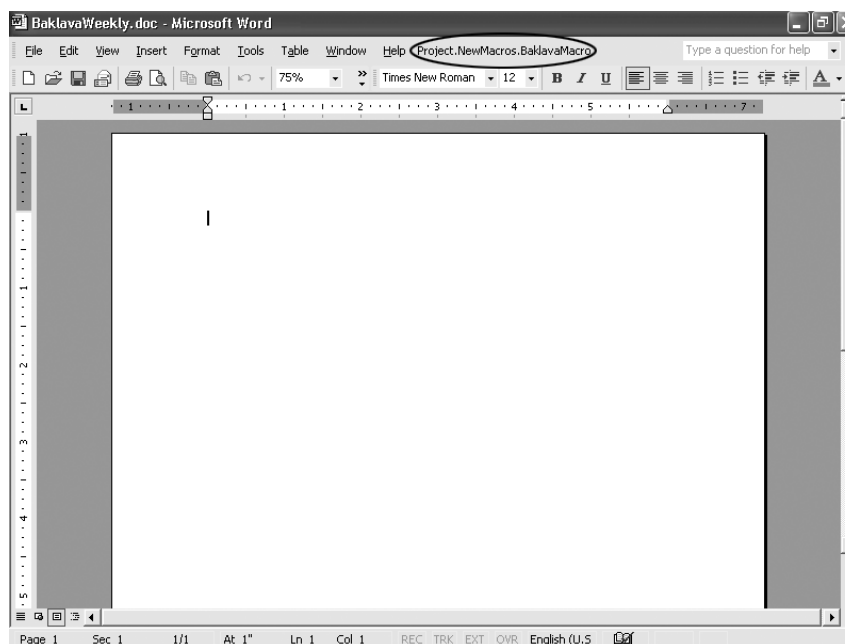
- Open the VBA IDE
- Select **File**
- Click **Print**

## VBA 1.34

## CHAPTER 1 VBA

## 1.3 Recording and Manipulating Macros

**FIGURE 1.24**  
New Baklava document  
with macro button



### Printing a VBA Macro:

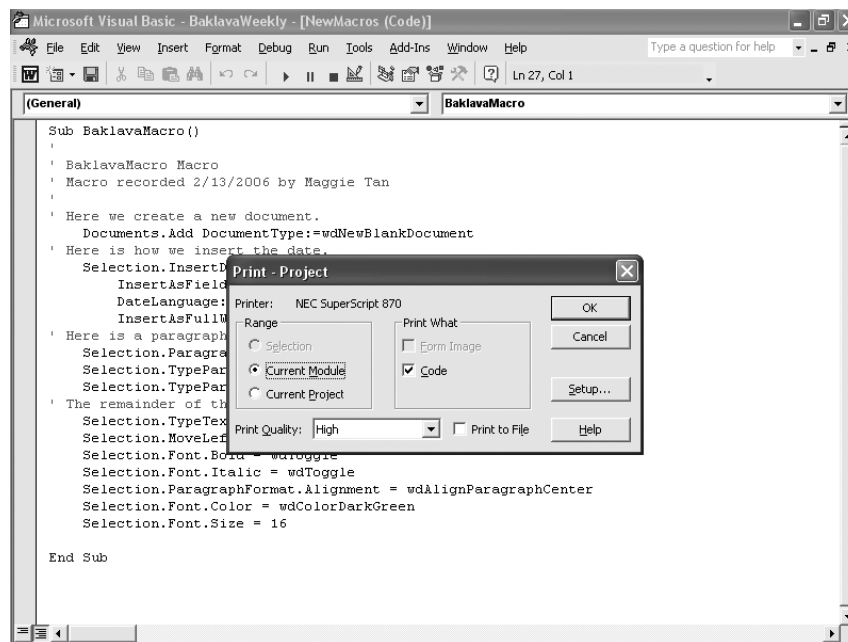
1. Press **Alt+F11** to return to the VBA IDE
2. Click anywhere in your **BaklavaMacro** code
3. Click **File** on the menu bar
4. Click **Print**
5. Select **Current Module** under **Range**
6. Select **Code** under **Print What**
7. Compare your print dialog box with Figure 1.25
8. Click **OK**

Now that you're done with your first macro, close the **BaklavaWeekly.doc**. If you didn't save your file, make sure you do it now. Close Microsoft Word by pressing **Alt+F4**.

## SESSION 1.3

### *making the grade*

1. A(n) \_\_\_\_\_ records a sequence of actions in an Office application that you can then repeat.
2. The \_\_\_\_\_ contains all the macro code in a particular project.
3. \_\_\_\_\_ is a Microsoft feature that completes pieces of programming code with the correct syntax.
4. \_\_\_\_\_ refers to how easy or how difficult it is to use a program.
5. The VBA \_\_\_\_\_ provides users with a one-click option to run a macro.



**FIGURE 1.25**  
BaklavaMacro print dialog box

## SESSION 1.4 SUMMARY

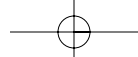
**VBA**, or *Visual Basic for Applications*, is a programming language that works within certain software applications. A **programming language** contains specific rules and words that explain the logical steps to solve a problem. You can use VBA to customize Microsoft Office applications. It's the programming language most often used for **end user development**.

VBA is a descendant of **BASIC** and comes directly from *Visual Basic*. **VBA programmers** rely on the **VBA integrated development environment** to write their programs. With VBA, programmers are able to convert **horizontal market software** into **vertical market software**.

Before programmers create programs they must first develop an **algorithm**. Programmers usually use **pseudocode** or a **program flowchart** to explain their algorithm. Once programmers are satisfied with their algorithm, they **code** the application following a strict **syntax**. If there are errors in their program, they must **debug** it.

You've spent time using the VBA IDE, or **VBA Editor**. The **Project Explorer**, **Properties Window**, and **Code Window** are useful VBA tools. You can customize the VBA IDE using **docked** and **undocked windows**. You can also use the **macro recorder** to save the steps needed to complete a task. You know how to save **macros** in a **macro module**, as well as manipulate and print them.

You're well on your way to becoming a VBA programmer. You'll learn more about harnessing the power of VBA in the coming chapters.

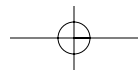
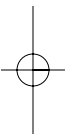
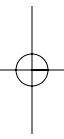


**NOTES**

**CHAPTER ONE**

[www.mhhe.com/plusseries](http://www.mhhe.com/plusseries)

**VBA 1.36**



**task reference roundup**

<b>Task</b>	<b>Page #</b>	<b>Preferred Method</b>
Setting the Macro Security Level	VBA 1.9, VBA 1.9	Select <b>Security</b> in the <b>Tools</b> menu under <b>Macro</b>
Word VBA IDE, open	VBA 1.10	Press <b>Alt+F11</b>
VBA IDE Help, opening	VBA 1.11	Press <b>F1</b>
Project Explorer, opening	VBA 1.12	Press <b>Ctrl+R</b>
Objects, viewing	VBA 1.13	Click on the <b>expand</b>  button in the <b>Project Explorer</b>
Objects, saving	VBA 1.13	Click on the <b>Word</b>  button on the toolbar and press <b>Ctrl+S</b>
Object Properties, changing	VBA 1.15, 1.16	Press <b>F4</b> to open the <b>Properties Window</b> , click on the object in the <b>Project Explorer</b> , and change object properties in the <b>Properties Window</b>
Code Window, opening	VBA 1.17	In the VBA IDE, press <b>F7</b>
VBA IDE Windows, docking and undocking	VBA 1.19	Right-click on either the <b>Property Window</b> , <b>Project Explorer</b> , or <b>Code Window</b> , and click on <b>Dockable</b>
Excel VBA IDE, open	VBA 1.21	Press <b>Alt+F11</b>
Excel VBA IDE, inserting sheet	VBA 1.22	Click on the <b>Excel</b>  button and then click on <b>Insert</b> in the <b>Menu</b> bar and then <b>Worksheet</b>
Macro Recorder, using	VBA 1.24	Click <b>Tools</b> on the <b>Menu</b> bar, select <b>Macro</b> , and click on <b>Record New Macro</b>
Macro, recording	VBA 1.26	Press the <b>Macro Recorder record</b>  button; when finished press the <b>Macro Recorder stop</b>  button
Macro, running	VBA 1.27, 1.31	Choose <b>Macro</b> , press the <b>Run</b>  button
Macro, reading	VBA 1.28	Select <b>object</b> in VBA IDE, press <b>F7</b> to open the <b>Code Window</b>
IntelliSense, using	VBA 1.29	Select <b>options</b> in the <b>Tools</b> menu, click on the <b>Editor</b> tab, and check the <b>Auto List Members</b> box
Macro, revising	VBA 1.30	Select <b>object</b> in VBA IDE, press <b>F7</b> to open the <b>Code Window</b>
Macro, assigning	VBA 1.32	Click <b>Tools</b> , then <b>Customize</b> , and assign the macro to a button
Macro, printing	VBA 1.34	Open the VBA IDE, click <b>File</b> , and then <b>Print</b>



## review of concepts

LEVEL TWO

### FILL-IN

1. A(n) \_\_\_\_\_ contains specific rules and words that explain the logical steps to solve a problem.
2. When a computer language is \_\_\_\_\_ it has the ability to work on a variety of computers.
3. A(n) \_\_\_\_\_ programming language responds to a user clicking on an icon or typing.
4. When you create a graphical depiction of the detailed steps you'll use to solve a problem, you've got a \_\_\_\_\_.
5. A(n) \_\_\_\_\_ is a characteristic of an object.
6. You can manipulate how windows are arranged in the VBA IDE. You can either \_\_\_\_\_ or \_\_\_\_\_ them.

CHAPTER ONE

### REVIEW QUESTIONS

1. VBA offers you many benefits and advantages to customizing Office applications. Explain three reasons why a business might use VBA with Microsoft Office.
2. In this chapter we've discussed how to think like a programmer when you approach a problem. Explain how you might approach a problem like a programmer.
3. You can customize the look of your VBA IDE in a variety of ways. Explain how you would dock and undock the Project Explorer and Properties Window.
4. Compare and contrast the Word and Excel VBA IDEs. How are these two VBA IDEs similar? How are they different?

### CREATE THE QUESTION

Read the following statements. Then create a short question that each statement would answer.

#### ANSWER

1. A programming language that works within another application
2. Using English statements to create an outline of the necessary steps for a piece of software to operate
3. It's another name for the VBA IDE
4. The VBA IDE window that contains the program code
5. All of a project's macros are contained here
6. Use this to create a one-click solution for running macros

#### QUESTION

---



---



---



---



---



---





### 1. Creating Business Letterhead

June's Bridal Shoppe, a store specializing in bridal gowns and wedding planning, sends a Congratulations letter to each customer when she gets married. Previously, June had purchased letterhead from the local store for writing her letters. However, she recently bought a new computer equipped with Microsoft Office and a new color printer. She asks you to create

some letterhead containing her business information and the current date. She also wants to be able to click a button and have a page ready to print on the day of each wedding. You decide to create a macro that will open a new Word document with the desired information. You'll also put a macro button in the document for June.

#### 1. Open Microsoft Word

**tip:** Press **Ctrl+N** to create a new Word document if one isn't automatically created

2. Press **Ctrl+S**
3. Save the file as **BridalLetterhead.doc**
4. Press the **Enter** key
5. Turn on the macro recorder
6. Record a macro called **Letterhead** in the **BridalLetterhead** document
7. Press **Ctrl+N**
8. Insert the date in the format **Thursday, June 19, 2006** and align it to the right
9. Press the **Enter** key twice and click the **Center**  button
10. Type **Congratulations from June's Bridal Shoppe**
11. Highlight **Congratulations from June's Bridal Shoppe** and click the **Bold**  button
12. Use the **Font Color**  button to make the text blue
13. Click the **Stop Recording**  button on the Macro Recorder
14. Click anywhere on the Word document
15. Press **Alt+F11**
16. Press **Ctrl+S**
17. Test the macro
18. Print the macro code
19. Quit Microsoft Word

## 2. Creating a Sales Report

You have a job at Albert's Antique Emporium, an antique and secondhand shop near the university. You work on some evenings and weekends. Albert has run the shop for more than 25 years and has always kept track of sales in a handwritten ledger. Albert Jr. has recently taken over the shop and wants to use Microsoft Excel to track sales so he can calculate profits and create reports in various formats.

Albert Jr. asks you to help him by making an opening page in Microsoft Excel. You decide to use a macro to create this new Excel workbook. You record a macro and show Albert Jr. the results. He's satisfied, but would like you to make a few changes in the macro. He wants **Prepared by:** and **Prepared on:** to be bold and red, and the name and date to be bold.

1. Open the workbook **vba01sales.xls**
2. Click **File**
3. Click **Save As**
4. Save the workbook as **AntiqueSales.xls**
5. Press **Alt+F11**
6. Place the cursor at the end of the line `ActiveCell.FormulaR1C1 = "Prepared by:"`
7. Press the **Enter** key
8. Type `ActiveCell.Font.Bold = True`
9. Press the **Enter** key
10. Type `ActiveCell.Font.ColorIndex = 3`
11. Place the cursor at the end of the line `ActiveCell.FormulaR1C1 = "Prepared on:"`
12. Press the **Enter** key
13. Type `ActiveCell.Font.Bold = True`
14. Change the 4 in `ActiveCell.Font.ColorIndex = 4` to a 3
15. Place the cursor at the end of the line `ActiveCell.FormulaR11C1 = "=TODAY()"`
16. Press the **Enter** key
17. Type `ActiveCell.Font.Bold = True`
18. Press **Ctrl+S**
19. Test the macro
20. Print the macro code
21. Quit Microsoft Excel

## 1. Solving Computer Problems

You are a Help Desk consultant at your university. You answer faculty, staff, and student computing questions over the phone or e-mail. You routinely answer questions about setting up e-mail accounts, connecting to the Internet, and using Microsoft Office.

Stu, the help desk supervisor, has asked you to help him keep track of the problems the Help Desk solves on a daily basis. Since every Help Desk consultant has Microsoft Word installed on his or her computer, Stu decides to use a macro to generate a Trouble Ticket for each problem the Help Desk solves. It's important to know who had the problem, what the problem was, and which Help Desk consultant solved the problem.

Stu knows you are good at macros so he asks you to create a macro that will produce a new document with the information shown in Figure 1.26. Stu also wants you to place a macro button in Microsoft Word that will execute the macro when someone clicks it.

Open a new Word document and record a macro that will generate a Word document like Figure 1.26. Make sure that the macro creates a new document and places the current date in each. Leave the **Name**, **Problem**, and **Issue Resolved by:** areas blank so each Help Desk consultant can fill these in.

Make sure to place your name and any other information your instructor requires in the VBA code comment area. When you're done, save the file as **TroubleTicket.doc** and print the macro code module. Be prepared to turn it in to your instructor.

## 2. Tracking Trees

Maynard's Tree Farm (MTF) sells a variety of trees to landscapers and individuals to plant in parks and yards. MTF also sells fresh cut pine trees at Christmas. Maynard recently bought a computer with Microsoft Office. His wife, Millie, is learning how to track tree inventory using Microsoft Excel. They've asked you to help them create a macro to automate parts of the spreadsheet.

Every time Maynard plants a new crop of trees, Millie wants to create a new worksheet to track the type of tree, how many Maynard planted, and where he planted them. Millie will need a new sheet each time Maynard plants a new crop. Over time Millie will accumulate many worksheets in her tree inventory workbook. Both Maynard and Millie know they will also need to account for trees sold in the workbook, but they'll worry about this later. The trees they enter in this workbook will be a few years old before they sell them.

Millie asks you to create a macro that will create one sheet named **WhitePine** to start. Remember to include the following items in the worksheet: **Number Planted**, **Condition**, and **Area Planted**. Before she leaves, Millie asks you to use your best judgment to format and make the worksheet look professional. You've already created an opening page. Now it's time to record the macro, test it, and then show it to Millie.

Open the worksheet **vba01trees.xls**. Take a moment to look at the opening page you've created. Millie didn't want any new buttons added to the menu or the

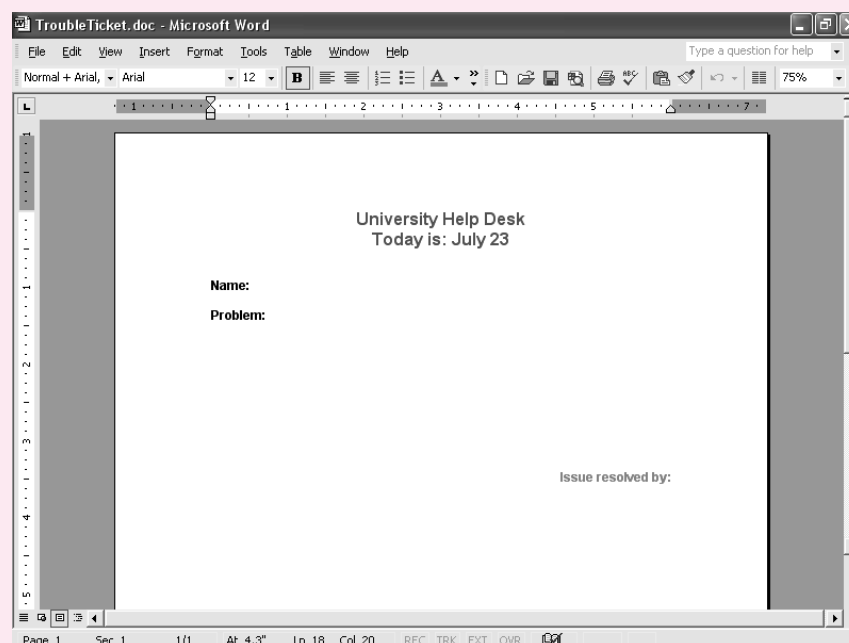
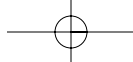


FIGURE 1.26  
TroubleTicket document



## hands-on projects

### challenge!

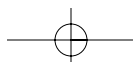
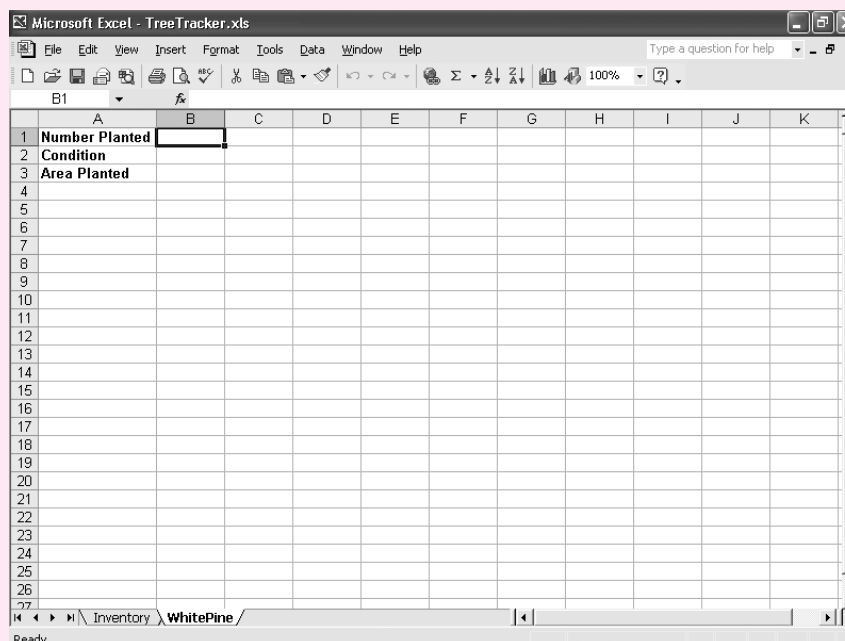
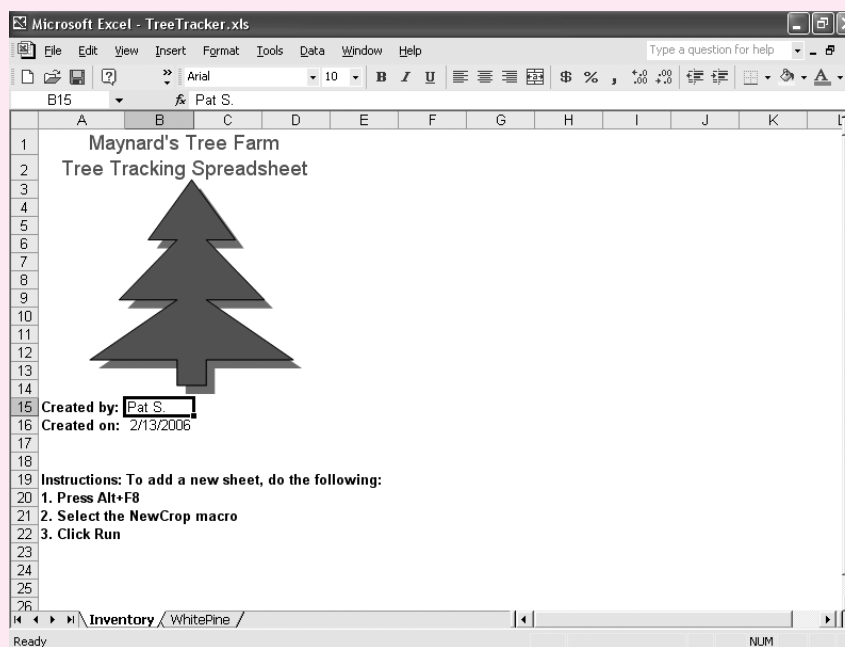
toolbar so you've placed instructions on the opening spreadsheet. Notice the name of the macro. You'll need to name your recorded macro the same. Take a moment to place your name after the **Created by:** then save this file as **TreeTracker.xls**.

When you record your macro, make sure to open a new worksheet. Name the worksheet **WhitePine** as you record the macro. As you format the various

items, make them bold so they will stand out from the text Millie enters. Before you work on your macro, look at Figure 1.27 so you have a good idea of the steps you need to record. Your **WhitePine** worksheet should look similar.

When you're done testing your macro, save the file and print the macro code module. Be prepared to turn it in to your instructor.

**FIGURE 1.27**  
Tree Farm Inventory and  
WhitePine worksheets



## running project

**la llama cycle**

La Llama Cycle (LLC) started as a small business run in Jesus Rodriguez's garage. For five years during the evenings and on weekends Jesus converted stock motorcycles into personalized machines. As more people drove his customized motorcycles around town, more people wanted their own customized motorcycles. Soon Jesus had more business than he could handle working part time. He quit his day job and began customizing motorcycles full time. In the past month, LLC has opened its third motorcycle shop and now employs 150 people at three locations. Customers come from the tri-county area to have their motorcycles customized at LLC.

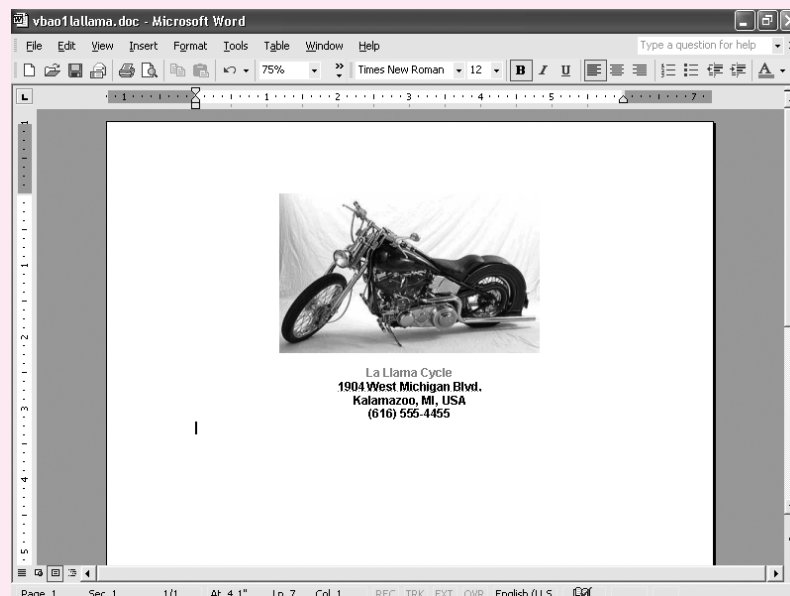
With the latest expansion, Jesus has also purchased new computer systems running Microsoft Windows with Office. Previously Jesus did much of the inventory, purchasing, payroll, and other business processes by hand. Jesus realizes that with the latest expansion he needs to create computerized solutions to his business processes. Jesus knows how to customize motorcycles and run his company well. He is not proficient in computers, but he knows what he needs. So when he ordered the computers he also hired you as the IT

support staff. You have been working for the past month installing the new systems and familiarizing yourself with the business processes. Now you'll customize Office applications to help LLC operate more efficiently.

Your first assignment comes from Jennifer, LLC's general manager. She'd like you to create a process that will allow all LLC employees to create a Word document with LLC's letterhead. You mention that you could create a macro that will run on all of the company's computers. Users would just need to click a button on the menu bar in Microsoft Word to create the letterhead. She agrees that this would be the best solution. She'll stop by later in the week to see what you've done.

It's up to you to design the letterhead. You might use Figure 1.28 to help you create it. Make sure to plan out the steps to create the letterhead before starting your macro recording. Remember, you can make changes to your macro in the VBA IDE as well.

When you finish with your macro, make sure to save your file as **LLCLetterhead.doc** and print out a copy of the macro code module. Be prepared to turn it in to your instructor.



**FIGURE 1.28**  
La Llama letterhead

