

## Bonus Chapter 3

---

# Ten Really Cool Things You Can Do with VBA

---

### *In This Chapter*

- ▶ Writing programs that meet your personal needs
  - ▶ Opening and using files as needed
  - ▶ Working with Access to create complex documents
  - ▶ Creating PowerPoint presentations automatically
  - ▶ Using Object Linking and Embedding (OLE) to link and embed data
  - ▶ Working with other controls on forms
  - ▶ Developing programs that use non-Office components
  - ▶ Using Microsoft's online documentation for your programming needs
  - ▶ Designing jigs and templates for your own use
  - ▶ Creating reusable code libraries
- 

**Y**ou can write an impressive number of programs with VBA and not even begin to use all the features that it provides. That's the thing that I find most interesting about working with VBA. When you need to perform a task, such as automating your worksheet, there's usually at least one way to accomplish it and generally more than one way. As discussed in Chapter 12, you need some additional help when you're working with the Ribbon in Office 2007, but you generally use VBA alone to perform most application tasks.

## *Personalizing Your VBA Applications*

If you've already read most of the chapters in this book, you're well on your way to becoming a hotshot VBA programmer. You have all the skills required to write any program that your mind can think to create and many programs that you can't even imagine at the moment. That's the goal of this chapter: After you master the programming techniques, it's time to start considering what you can do with them.

The coolest thing that you can ever do with VBA is to make it your personal tool for accomplishing tasks that you want to do. It doesn't matter how you use the program — personal programs are just as worthwhile as those that you create for your business or fellow workers. The fact that you did something for your personal satisfaction and to meet your personal needs is the best feeling that there is. You now have the power to write any program that you need. Make sure that you take time to play.

## Using Files with the Open Command



Many of the Microsoft and third-party online VBA references mention the `Open` function, which is an older method of working with files that you won't use for opening standard application files. However, you might need to open an INI (initialization settings) file or a file containing data produced in another application that Office doesn't support directly. This function requires a number of arguments, including the name of the file and the kind of access that you require. Listing BC3-1 is a simple example of how you can use the `Open` function to open a text file on the local drive. This text file contains data that could have resided on a mainframe or in another PC program. The example translates this data and places it in Excel. (You can find the source code for this example on the Dummies.com site at <http://www.dummies.com/go/vbafd5e>.)

### Listing BC3-1 Opening and Translating a Text File into an Excel Worksheet

```
Public Sub OldOpen()  
    ' Create the input strings.  
    Dim DataLine As String  
    Dim ThisCell As String  
  
    ' Used for looping.  
    Dim Counter As Integer  
    Counter = 1  
    Dim CellCounter As Integer  
  
    ' Open the file.  
    Open ThisWorkbook.Path + "\Temp.txt" For Input As #1  
  
    ' Read the data one line at a time.  
    While Not EOF(1)  
        Line Input #1, DataLine  
  
        ' Verify whether there is a tab in the text. If so,  
        ' place each text element in a separate column.  
        If InStr(1, DataLine, vbTab) Then  
  
            ' Set the cell counter.
```

```
CellCounter = 1

' Keep processing the line of text until finished.
While (Len(DataLine) > 0)

    ' Verify that the text still has tabs in it.
    If InStr(1, DataLine, vbTab) Then

        ' Get the text to the left of the tab and place it in the cell.
        ThisCell = Left(DataLine, InStr(1, DataLine, vbTab) - 1)
        Sheet1.Cells(Counter, CellCounter) = ThisCell

        ' Make the current text the remaining tabbed text element.
        DataLine = Mid(DataLine, InStr(1, DataLine, vbTab) + 1)

        ' Go to the next cell.
        CellCounter = CellCounter + 1
    Else
        ' The text is free of tabs. Make the cell equal to this last
        ' line and then clear the data line.
        Sheet1.Cells(Counter, CellCounter) = DataLine
        DataLine = ""
    End If
Wend

Else

    ' Place the data in the worksheet.
    Sheet1.Cells(Counter, 1) = DataLine
End If

' Update the counter.
Counter = Counter + 1
Wend

' Close the file.
Close #1
End Sub
```

This code reads a text file into a worksheet. Each line of text appears in a separate worksheet row. When the text file contains tabs, it places each tabbed element in a separate cell. The effect is precisely the same as shown in this code:

```
Public Sub UseOpen()
    ' Open a text file as a workbook.
    Dim TempBook As Workbook
    Set TempBook = Workbooks.Open(ThisWorkbook.Path + "\Temp.txt")
End Sub
```

When you're working with native application files, such as the .doc or .docm extension for Word, the `FileSystemObject` method of working with files that is demonstrated in Chapter 10 is normally superior because it provides better access to the drive system and works more consistently than the `Open` function. In addition, you need to think only about one or two objects and not a host of individual functions.

You still need the `Open` function to perform certain tasks. The most important is the use of *binary* (non-text) files. If you decide to work with binary files outside of the objects that the host application provides, you must use the `Open` function. The Erlandsen Data Consulting Web site at <http://www.erlandsendata.no/english/index.php?d=envbafileaccessbinary> provides a good example of using binary data records. You can find an interesting alternative to using the `Open` function for binary data at the DeveloperFusion.com Web site at <http://www.developerfusion.com/show/2542/>. This method relies on using the `ADODB` object.

Record-based access also requires the `Open` function. You can create a file with fixed-length records. You can use these text-based files to exchange information between databases. The Microsoft Knowledge Base article at <http://support.microsoft.com/?kbid=209231> provides a good example of how to use the `Open` function for random file access.

The most important consideration for using the `Open` function is that it provides better control than some of the newer methods that VBA provides. The first example in this section uses a lot of code precisely because you have to do everything, but that means that you get to decide how things will work. Using a newer function requires less code, but it also means that you have to do things the Microsoft way, which means a loss of flexibility and, in some cases, functionality.

## Defining Database Connections



When you develop complex programs, you might find that you want to use data directly from a database, such as Access. In Chapter 15, you discover how to use Data Access Objects (DAO) to accomplish this task. DAO works fine for many tasks, but you gain additional flexibility by using ActiveX Data Objects (ADO) to access databases, including Access, from other programs. This example relies on the Microsoft ActiveX Data Objects 2.7 Library. You can add this library by using the `Tools`⇨`References` command. The code in Listing BC3-2 places the contents of the `Word List` table of the `AccessObjects` database into an Excel worksheet. (You can find the source code for this example on the Dummies.com site at <http://www.dummies.com/go/vbafd5e>.)

**Listing BC3-2 Exporting an Access Table to an Excel Worksheet**

```

Public Sub GetData()
    ' Create the database connection.
    Dim DBConn As ADODB.Connection
    Set DBConn = New ADODB.Connection
    DBConn.ConnectionString = _
        "Driver={Microsoft Access Driver (*.mdb)};" + _
        "Dbq=" + ThisWorkbook.Path + "\AccessObjects.MDB;" + _
        "Uid=admin;Pwd="

    ' Open the database.
    DBConn.Open

    ' Create a recordset.
    Dim DBRec As ADODB.Recordset
    Set DBRec = New ADODB.Recordset

    ' Set the recordset parameters.
    DBRec.ActiveConnection = DBConn
    DBRec.Source = _
        "Select * From [Word List] Where IsAcronym=True"

    ' Open the recordset.
    DBRec.Open

    ' Display the fields as headers.
    Dim CurrentCell As Integer
    CurrentCell = 1
    Dim AField As ADODB.Field
    For Each AField In DBRec.Fields
        Sheet2.Cells(1, CurrentCell) = AField.Name
        CurrentCell = CurrentCell + 1
    Next

    ' Display the data.
    Dim RowCount As Integer
    RowCount = 3
    While Not DBRec.EOF
        CurrentCell = 1
        For Each AField In DBRec.Fields
            Sheet2.Cells(RowCount, CurrentCell) =
                AField.Value
            CurrentCell = CurrentCell + 1
        Next
        DBRec.MoveNext
        RowCount = RowCount + 1
    Wend

    ' Close the database.
    DBConn.Close
End Sub

```

This program resides in Excel, yet it uses Access data. You can use a combination of sorting and filtering and different source statements to create the data set that you need for a particular program. For that matter, you pick the fields that you want or manipulate the data in various ways. The point is that you can perform various kinds of data merges within your programs. Make sure that you close the database connection to ensure that you don't lose data or corrupt the database file in some way.

## *Automating PowerPoint Presentations*

PowerPoint, more than any other application, can benefit from the cross-application benefits demonstrated in Chapter 16. A single VBA program could rely on Word for the text in a PowerPoint presentation, Access for the presentation data, and Excel for presentation graphs and statistics. The important idea is to use each of the applications for their intended purpose. PowerPoint is a consolidation tool: It's best used to consolidate into a presentation the data that you create in other applications. Think of this program as a container for the other kinds of data that you create, and the requirements for creating programs for it become a little easier to understand.

## *Creating Data Connections with OLE*



Most of the examples in this book concentrate on creating new data, moving data from one application to another, or manipulating data in some way. An application might contain data in precisely the format that you need, so all you really need is a reference to it. That's where Object Linking and Embedding (OLE) comes into play. You can use this technology to simply place a copy of the data in another document. This technique is equivalent to using the Edit⇨Paste Special or the Insert⇨Object command. Listing BC3-3 shows an example of adding a data connection by using OLE. (You can find the source code for this example on the Dummies.com site at <http://www.dummies.com/go/vbafd5e>.)

### **Listing BC3-3 Using OLE with VBA**

```
Public Sub AddOLE()  
    ' Create a reference to the current objects.  
    Dim Obj As OLEObjects  
    Set Obj = Sheet3.OLEObjects  
  
    ' Add a new object.  
    Obj.Add Filename:=ThisWorkbook.Path + "\Cuckoo.wav", _  
        Link:=False, Top:=20, Left:=40, _  
        IconLabel:="The Cuckoo Sound"  
End Sub
```

The biggest reason to use this technique is to automate data additions to a document. For example, you might want to construct a letter template that automatically adds graphics based on the customer or the type of account. In other cases, you could use this technique to automate the object-insertion process for less-skilled users.

## *Adding Functionality with Controls*

Chapter 7 provides a wealth of tips and hints about using standard VBA controls. You also discover how to add new controls to your VBA palette in that chapter's "Adding controls to the Toolbox" section. However, that chapter doesn't really convey the power that controls can have over your development efforts. A *control* is code that someone else has already debugged and tested for you. It includes some type of graphical element (unlike components, which include only code). In short, controls represent the fastest and easiest method of adding functionality to your program.

You get controls from a variety of sources. For example, if you want to add the ability to display PDFs to Word, just add an Acrobat Reader control to your palette. Adobe installs this control whenever you install Acrobat Reader (<http://www.adobe.com/products/acrobat/readermain.html>), so you don't even have to do anything special to receive the functionality.

Other programmers make their controls available for use. You have to exercise care in downloading controls from the Internet, but some sites are quite reliable. Here's a list of some sites where you can look for controls:

- ✓ **c|net Download.com:** <http://www.download.com/>
- ✓ **Tucows:** <http://www.tucows.com/>
- ✓ **Sofotex.com:** <http://www.sofotex.com/download/Programming/ActiveX/>
- ✓ **freeDownloads Center:**  
<http://www.freedomdownloadscenter.com/Programming/ActiveX/>
- ✓ **SHAREWAREORDER:** <http://www.sharewareorder.com/>

After you download a control, you have to install it on your system. I set aside a special folder for controls that I download so that I can find and remove them later. To add a control to your system, open a command prompt by choosing Start⇨Programs⇨Accessories⇨Command Prompt. Type **RegSvr32 <Name of Control>** at the command prompt and then press Enter. Windows displays a success message when the control is registered for use. Now you can add it to your VBA project.

When you find that you no longer need a control, you have to unregister it first so that the Windows Registry remains clean. To remove a control, open a command prompt with the Start→Programs→Accessories→Command Prompt command, type **RegSvr32 -u <Name of Control>**, and then press Enter. Windows displays a success message. At this point, you can erase the file.

## *Getting and Using Components*

You have no idea how many useful components reside on your machine. I know that I've delved into the components on my machine for years and have yet to try them all. The undiscovered country of code that you can use to enhance your programs extends to many areas. You can use .NET in your programs because many features of the .NET Framework appear as standard components. The same concept extends to Java applets, utilities that Windows provides, or just about any other piece of code that you can imagine.

The magic word to access any component on your machine is `CreateObject`. This function appears in many examples in this book. However, before you can use a component, you need to know that it exists.

The *OLE/COM Object Viewer utility* is a special program that displays all the components on your machine. This tool comes as part of Visual Studio and a few other Microsoft products, but you can also download it from the Microsoft Web site at <http://www.microsoft.com/downloads/details.aspx?FamilyID=5233b70d-d9b2-4cb5-aeb6-45664be858b6>. The purpose of this program is to show all the components and controls on your machine. Simply double-click the file that you download to install the utility on your machine.

When you open the OLE/COM Object Viewer, you see categories of objects that you can use. For example, all the controls on your machine appear in the `Controls` folder. These categories help VBA know which items to display in the dialog boxes that let you add new controls to the Toolbox or add new component references.

Using the OLE/COM Object Viewer can provide you with names of new components to try. It can also tell you about the component. For example, look at the `InprocServer32` entry on the Registry tab, and you see the location of the file that supports the component. In many cases, this location information tells you who created the control and also provides clues to how you can use the component to improve your code.

## Using Microsoft's Online Documentation

In the Registry example in Chapter 13, you discover that you can use Windows 32-bit Application Programming Interface (Win32 API) calls directly in VBA. You can also access the .NET Framework by using new features in Office 2003. Microsoft also provides a vast array of components and controls that you can use. All this free code sounds great until you realize that you don't have any documentation for it.

Fortunately, Microsoft provides a wealth of online documentation to go with the free code that it provides. The Microsoft Developer Network (MSDN) Library at <http://msdn2.microsoft.com/en-us/library/default.aspx> lets you search for just about any current Microsoft product. The table of contents on the left side of the site helps you refine your search after you locate something interesting. In addition to using the MSDN Library, you also want to check the Microsoft Knowledge Base at <http://support.microsoft.com/default.aspx> for updates, programming tips, and additional information.

## Creating Your Own Jigs and Templates



You might think that I create every module, form, or code module by hand. In some respects, this is true. I do write any custom code by hand. However, there's no reason for me to write my name and create basic functions or sub-procedures that I know I'm going to use every time. I don't feel any obligation to write the code to create basic structures. VBA is your ally; use it to make your workload just a little lighter. Listing BC3-4 shows an example of a template that automatically creates a module and fills it with standard data. Note that you must add a reference to the Microsoft Visual Basic for Applications Extensibility component for this example. (You can find the source code for this example on the Dummies.com site at <http://www.dummies.com/go/vbafd5e>.)

### Listing BC3-4 Automating the Module Creation Process

```
Public Sub ModuleStart()  
    ' Create the new module.  
    Dim NewMod As Module  
    Set NewMod = Modules.Add  
  
    ' Get the module name.
```

*(continued)*

**Listing BC3-4 (continued)**

```
Dim ModName As String
ModName = InputBox("Type the Module Name", "Name")
NewMod.Name = ModName

' Get the new projects
Dim MyProj As CodeModule
Set MyProj = Application.VBE.VBProjects(1).VBComponents(ModName).CodeModule

' Open the file.
MyProj.CodePane.Show

' Add the required option statement, module header, and opening Sub.
MyProj.InsertLines 1, "Option Explicit" + vbCrLf + vbCrLf + _
    "' Module Name: " + ModName + vbCrLf + "' Author: " + _
    Application.UserName + vbCrLf + "' Date: " + CStr(DateTime.Now) + _
    vbCrLf + vbCrLf + "Public Sub Main()" + vbCrLf + vbCrLf + "End Sub"
End Sub
```

## *Developing Reusable Libraries*

When you build new programs with VBA, keep reusability in mind. Many examples in this book appear as a single, large piece of code to make them easier to explain. My own code contains smaller modules that accept one or more arguments as input. Each module performs one task exceptionally well. Using this technique means that code that I write today can also perform work tomorrow.

All my tested code goes into a utility module (named `Utility.bas`). When I begin a new project, I import `Utility.bas` and instantly have a lot of the code that I need written. The functions and sub-procedures in the main module for my program call each of these utility modules and ask it to perform its special task. Some programmers call this the *Lego approach* to writing code because of the similarity to the toy's modularity of interchangeable pieces. Reusable libraries can save you a great deal of time and make mundane programming tasks a lot easier and fun.