

Bonus Chapter 1

Interacting with Other Office Applications

In This Chapter

- ▶ Starting or activating another application from Excel
 - ▶ Controlling Word from Excel and vice versa
 - ▶ Sending personalized e-mail from Excel
-

If you use Excel, you likely use other applications that comprise Microsoft Office. Just about everyone uses Word, and you're probably familiar with PowerPoint or Access.

In this bonus chapter, I present some simple examples that demonstrate how to use Excel VBA to interact with other Microsoft Office applications.

Starting Another Application from Excel

Starting another application from Excel is often useful. For example, you might want to launch another Microsoft Office application or even a DOS batch file from an Excel VBA macro.

Using the VBA Shell function

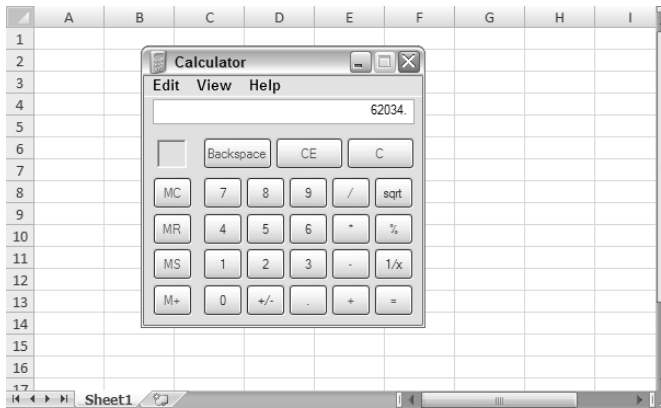
The VBA Shell function makes launching another program relatively easy. The following example starts the Windows Calculator program, which is named CALC.EXE:

```
Sub StartCalculator()  
    Dim Program As String  
    Dim TaskID As Double  
    On Error Resume Next  
    Program = "calc.exe"
```

```
TaskID = Shell(Program, 1)
If Err <> 0 Then
    MsgBox "Can't start " & Program
End If
End Sub
```

Figure BC01-1 shows the Windows calculator displayed as a result of running this procedure.

Figure BC01-1:
The Windows Calculator program.



The Shell function returns a task identification number for the application. You can use this number later to activate the task (which is why I declared the variable above the procedure: It will keep its value). The second argument for the Shell function determines how the application is displayed. (1 is the code for a normal-size window, with the focus.) Refer to the Help system for other argument values.

If the Shell function is unsuccessful, it generates an error. Therefore, this procedure uses an On Error statement to display a message if the executable file cannot be found or if some other error occurs.

But what if the Calculator program is already running? The StartCalculator procedure simply opens another instance of the program. In most cases, you want to activate the existing instance. The following modified code solves this problem:

```
Public TaskIDSub StartCalculator2()
    Dim Program As String
    Dim TaskID As Double
    Program = "calc.exe"
    On Error Resume Next
    AppActivate "Calculator"
    If Err <> 0 Then
        Err = 0
    End If
End Sub
```

```

        TaskID = Shell(Program, 1)
        If Err <> 0 Then MsgBox "Can't start " & Program
    End If
End Sub

```

This modified procedure uses an AppActivate statement to activate the application (Windows Calculator in this case) if it's already running. The argument for AppActivate is the Caption of the application's title bar. If the AppActivate statement generates an error, it means the Calculator isn't running. If it's not running, the routine starts the application with the Shell function.

Here's another example of using the Shell function. The OpenFolder procedure displays the folder that holds the workbook:

```

Sub OpenFolder()
    Dim Program As String
    Dim Folder As String
    Program = "explorer.exe"
    Folder = ThisWorkbook.Path
    Shell Program & " " & Folder, 1
End Sub

```

In this case, the program is explorer.exe, and the folder is specified by the Path property of the Workbook object. If you specify a path that doesn't exist, you see an error message from Windows (not from Excel).

Activating a Microsoft Office application

If the application that you want to start is one of several Microsoft applications, use the Application object's ActivateMicrosoftApp method. For example, the following procedure starts Word:

```

Sub StartWord()
    Application.ActivateMicrosoftApp xlMicrosoftWord
End Sub

```

If Word is already running when the preceding procedure is executed, it is activated. Other constants are available for this method:

- ✓ xlMicrosoftPowerPoint (PowerPoint)
- ✓ xlMicrosoftMail (Outlook)
- ✓ xlMicrosoftAccess (Access)
- ✓ xlMicrosoftFoxPro (FoxPro)
- ✓ xlMicrosoftProject (Project)
- ✓ xlMicrosoftSchedulePlus (SchedulePlus)

Using Automation in Excel

You can write an Excel macro to control other applications, such as Microsoft Word. More accurately, Excel macros control the most important component of Word: the so-called automation server. In such circumstances, Excel is called the *client application*, and Word is the *server application*.

The concept behind automation is quite appealing. A developer who needs to generate a chart, for example, can reach into another application's grab bag of objects, fetch a Chart object, and then manipulate its properties and use its methods. Automation, in a sense, blurs the boundaries between applications. For example, using automation, an end user might be working with a Word document inside Excel and not even realize it.



Some applications, such as Excel, can function as either a client application or a server application. Other applications can function only as client applications or only as server applications.

In the following sections, I demonstrate how to use VBA to access and manipulate the objects exposed by other applications. The examples use Microsoft Word, but the concepts apply to any application that exposes its objects for automation.

Getting Word's version number

The following example demonstrates how to create a Word object to provide access to the objects in Word's object model. This procedure creates the object, displays the version number, closes the Word application, and then destroys the object, freeing up the memory that it used:

```
Sub GetWordVersion()  
    Dim WordApp As Object  
    Set WordApp = CreateObject("Word.Application")  
    MsgBox WordApp.Version  
    WordApp.Quit  
    Set WordApp = Nothing  
End Sub
```

The Word object that's created in this procedure is invisible. If you want to see the object while it's being manipulated, set its `Visible` property to `True`, as follows:

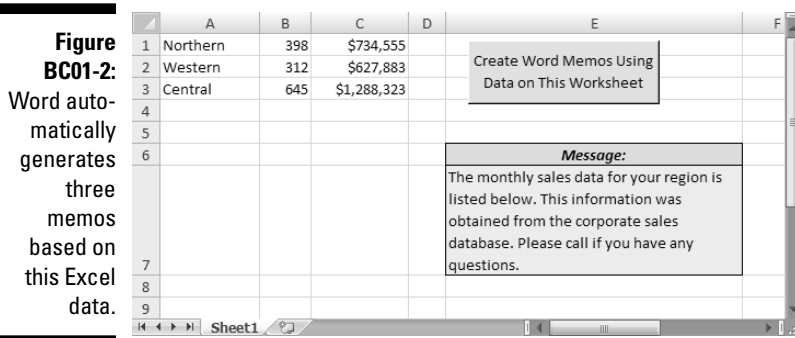
```
WordApp.Visible = True
```



Most of the automation examples in this chapter use late binding as opposed to early binding. What's the difference? When you use *early binding*, you must establish a reference to a version-specific object library, using Tools⇨References in the VBE. When you use *late binding*, setting that reference is not required. Both approaches have pros and cons.

Controlling Word from Excel

The example in Figure BC01-2 demonstrates an automation session by using Word. The MakeMemos procedure creates three customized memos in Word and then saves each memo to a separate file. The information used to create the memos is stored in a worksheet.



The code for the MakeMemos procedure is too lengthy to list here, but you can go to this book's Web site to check it out.

The MakeMemos procedure starts by creating an object called WordApp. The routine cycles through the three rows of data in Sheet1 and uses Word's properties and methods to create each memo and save it to disk. A range named Message (in cell E7) contains the text used in the memo. All the action occurs behind the scenes (Word is not visible). Figure BC01-3 shows a document created by the MakeMemos procedure.

Controlling Excel from Word

As you might expect, you can also control Excel from another application (such as another programming language or a Word VBA procedure). For example, you might want to perform some calculations in Excel and return the result to a Word document.

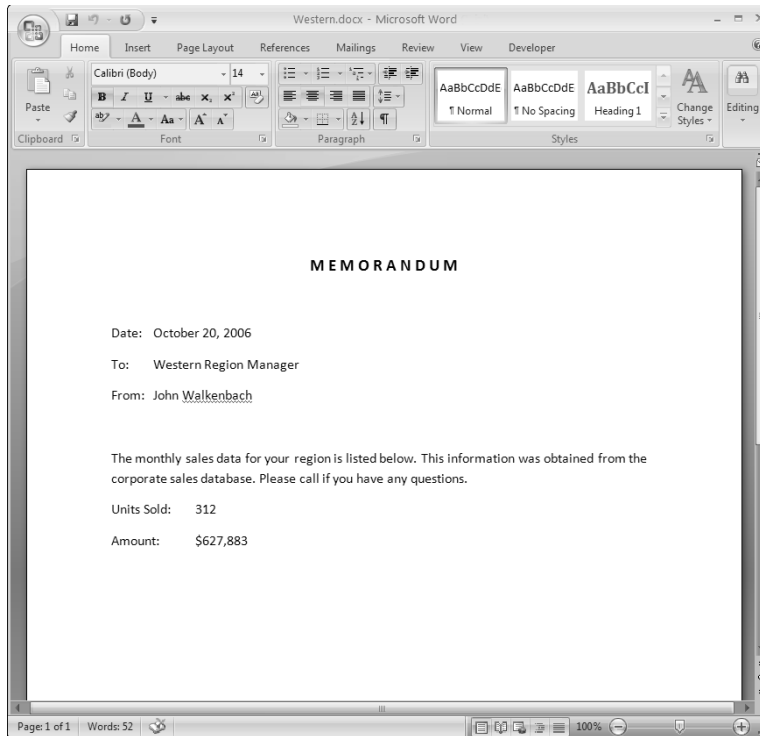


Figure BC01-3:
An Excel VBA procedure created this Word document.

You can create any of the following Excel objects with the adjacent functions:

- **Application object:** `CreateObject("Excel.Application")`
- **Workbook object:** `CreateObject("Excel.Workbook")`
- **Chart object:** `CreateObject("Excel.Chart")`

The example described in this section is a Word macro that creates an Excel Workbook object (whose moniker is `Excel.Workbook`) from an existing workbook named `projections.xlsx`. The Word macro prompts the user for two values and then creates a data table and chart, which are stored in the Word document.

The initial workbook is shown in Figure BC01-4. The `MakeExcelChart` procedure (in the Word document) prompts the user for two values and inserts the values into the worksheet.

Recalculating the worksheet updates a chart. The data and the chart are then copied from the Excel object and pasted into a new document. The results are shown in Figure BC01-5.

Figure BC01-4:
A VBA procedure in Word uses this worksheet.

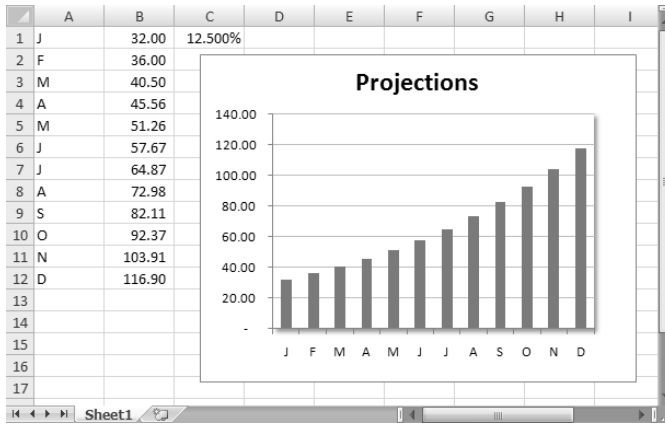
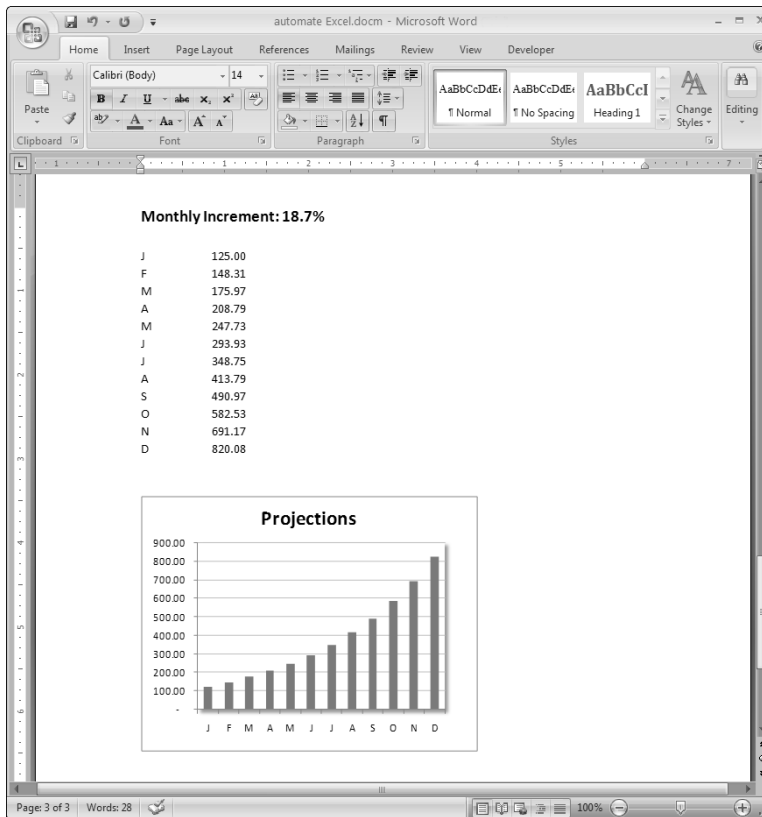


Figure BC01-5:
The Word VBA procedure uses Excel to create this document.



The code for the MakeExcelChart procedure follows:

```
Option Explicit

Sub MakeExcelChart()
    Dim XLSheet As Object
    Dim StartVal, PctChange
    Dim Wbook As String

    ' Insert a new page
    Selection.EndKey Unit:=wdStory
    Selection.InsertBreak Type:=wdPageBreak

    ' Prompt for values
    StartVal = InputBox("Starting Value?")
    PctChange = InputBox("Percent Change? For example,
        '5.2%'")

    ' Create Sheet object
    Wbook = ThisDocument.Path & "\projections.xlsx"
    Set XLSheet = GetObject(Wbook,
        "Excel.Sheet").Activatesheet

    ' Put values in sheet
    XLSheet.Range("StartingValue") = StartVal
    XLSheet.Range("PctChange") = PctChange
    XLSheet.Calculate

    ' Insert page heading
    Selection.Font.Size = 14
    Selection.Font.Bold = True
    Selection.TypeText "Monthly Increment: " & _
        Format(PctChange, "0.0%")
    Selection.TypeParagraph
    Selection.TypeParagraph

    ' Copy data from sheet & paste to document
    XLSheet.Range("data").Copy
    Selection.Paste

    ' Copy chart and paste to document
    XLSheet.Chartobjects(1).CopyPicture
    Selection.Paste

    ' Kill the object
    Set XLSheet = Nothing
End Sub
```



This example is available at the book's Web site.

Sending Personalized E-Mail by Using Outlook

The example in this section demonstrates automation with Microsoft Outlook. The code creates personalized e-mail messages by using data stored in an Excel worksheet.

Figure BC01-6 shows a worksheet that contains data used in e-mail messages: name, e-mail address, and bonus amount. This procedure loops through the rows in the worksheet, retrieves the data, and creates an individualized message (stored in the Msg variable).

Figure BC01-6: This information is used in the Outlook e-mail messages.

	A	B	C	D
1	Name	Email	Bonus	
2	John Jones	jjones@anydomain.com	\$2,000	
3	Bob Smith	bsmith@anydomain.com	\$3,500	
4	Fred Simpson	fsimpson@anydomain.com	\$1,250	
5				
6				
7				

```

Sub SendEmail()
    Dim OutlookApp As Object
    Dim MItem As Object
    Dim cell As Range
    Dim Subj As String
    Dim EmailAddr As String
    Dim Recipient As String
    Dim Bonus As String
    Dim Msg As String

    'Create Outlook object
    Set OutlookApp = CreateObject("Outlook.Application")

    'Loop through the rows
    For Each cell In _
        Columns("B").Cells.SpecialCells(xlCellTypeConstants)
        If cell.Value Like "*@*" Then
            'Get the data
            Subj = "Your Annual Bonus"
            Recipient = cell.Offset(0, -1).Value
            EmailAddr = cell.Value
            Bonus = Format(cell.Offset(0, 1).Value, "$0,000.")

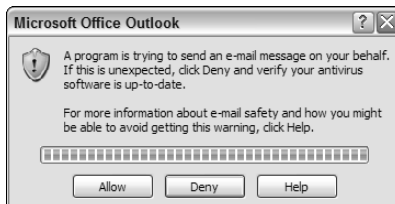
            'Compose message
            Msg = "Dear " & Recipient & vbCrLf & vbCrLf
        End If
    Next cell
End Sub

```

```
Msg = Msg & "I am pleased to inform you that "  
Msg = Msg & "your annual bonus is "  
Msg = Msg & Bonus & vbCrLf & vbCrLf  
Msg = Msg & "William Rose" & vbCrLf  
Msg = Msg & "President"  
  
'Create Mail Item and send it  
Set MItem = OutlookApp.CreateItem(0)  
With MItem  
    .To = EmailAddr  
    .Subject = Subj  
    .Body = Msg  
    .Display  
End With  
End If  
Next  
End Sub
```

This example uses the Display method, which simply displays the e-mail messages. To actually send the messages, use the Send method instead. Note however, that due to security measures, Outlook asks you for permission to actually issue the send command. See Figure BC01-7.

Figure BC01-7: Outlook asks you for permission to send e-mails through VBA code.



Notice that two objects are involved: Outlook and MailItem. The Outlook object is created with this statement:

```
Set OutlookApp = CreateObject("Outlook.Application")
```

The MailItem object is created with this statement:

```
Set MItem = OutlookApp.CreateItem(0)
```