

# Document Tricks

# 3

If you're a regular Word user, then you probably work with many different Word documents during an average work day, and you've probably got hundreds or even thousands of Word documents stored on your hard drive. The document is the basic Word container, and knowing how to work with documents is a fundamental Word skill. However, working with documents doesn't just mean using the commands on the Office menu (Open, Save, Close, and so on). Word is chock full of tricks, shortcuts, and settings that can make working with documents faster, easier, more powerful, and more efficient. This chapter introduces you to many of the techniques that cover everything, from easier ways to save and protect your documents to scripts that calculate editing time and billable time.

## Recovering More of Your Work with a Shorter AutoRecover Interval

No one who has used Word for any length of time needs to be convinced of the importance of saving a document regularly. We've all experienced that moment of horror (sometimes called the *obnosecond*) when we realize that Word has locked solid and we haven't saved for awhile, so all our recent work is toast. Most of us have become friendly with the Ctrl+S shortcut for the Save command and use it as often as possible. However, it's easy to forget to save when you're busy and a deadline looms large. As a safety net, Word has an AutoRecover feature that automatically stores information about the changes you've made to your document since the last save. If Word goes down for the count, it can use the AutoRecover data to help you recover some or all of your work.

## IN THIS CHAPTER

Recovering More of Your Work with a Shorter AutoRecover Interval .....	61
Automatically Saving Your Work Frequently .....	62
Closing a Document Without Saving .....	64
Closing All Your Open Documents .....	65
Making Backups as You Work .....	66
Showing More Items on the Recent Documents List .....	68
Opening the Most Recently Used Document at Startup .....	68
Clearing the Recent Documents List .....	69
Creating and Opening Document Workspaces .....	70
Automatically Prompting for Document Properties .....	72
Creating a Trusted Location for Documents .....	73
Viewing Total Editing Time Updated in Real-Time .....	75
Calculating Billable Time Charges .....	77
Locking Document Formatting .....	77
Preventing Untracked Changes .....	79
Setting Up a Document for Structured Editing .....	80
Inspecting a Document for Personal Information .....	84
Viewing Two Documents Side by Side .....	85
Updating All Fields Automatically .....	86

AutoRecover has saved me on a number of occasions, so I'm a big fan of this feature. If it has a downside, however, it's that the default interval that Word uses to save the AutoRecover data is too long: 10 minutes. You can lose a lot of work in 10 minutes, so it's a good idea to shorten the interval:

- If you work with only small- or medium-sized documents, shorten the interval to 1 minute.
- If you're working with a large document (several dozen pages or more), the AutoRecover process can take quite awhile. Therefore, if you occasionally work with large documents, shorten the interval to 3 or 4 minutes.
- If you work with large documents only, you might not want Word to use AutoRecover. In this case, you can turn off the AutoRecover feature.

To work with AutoRecover, choose Office, Word Options to open the Word Option dialog box, click Save, and then use the Save AutoRecover Information Every *X Minutes* spin box to set the interval you prefer. If you want to turn off AutoRecover, deactivate the AutoRecover Information check box. Click OK to put the new setting into effect.

## Automatically Saving Your Work Frequently

Setting the AutoRecover interval to 1 minute, as described in the previous section, is probably good enough for most people. However, a fast typist can easily write dozens of words in a minute, so you can still lose a fair amount of work even when using the shortest AutoRecover interval.

To get an even faster save interval, you can use VBA, specifically, Word's `OnTime` method, which runs a procedure at a specified time. The `OnTime` method uses the following syntax:

```
Application.OnTime(When, Name [, Tolerance])
```

<i>When</i>	The time (and date, if needed) you want the procedure to run.
<i>Name</i>	The name of the procedure to run when the time given by <i>When</i> arrives.
<i>Tolerance</i>	If Word isn't ready to run the procedure at <i>When</i> , it will keep trying for the number of seconds specified by <i>Tolerance</i> . If you omit <i>Tolerance</i> , VBA waits until Word is ready.

You must enter the *When* argument as a date/time serial number. The easiest way to do this is to use the `TimeValue` function:

```
TimeValue(Time)
```

<i>Time</i>	A string representing the time you want to use (such as "8:00 PM" or "20:00").
-------------	--------------------------------------------------------------------------------

For example, the following code runs a procedure called `MakeBackup` at 8:00 PM:

```
Application.OnTime _
    When:=TimeValue("8:00 PM"), _
    Name:="MakeBackup"
```

What we really want to do here is run a macro that saves the current document, and we want to run that macro at a specified time interval (for example, every 30 seconds). If you want the `OnTime` method to run after a specified time interval, use the expression `Now + TimeValue(Time)` for *When* (where *Time* is the interval you want to use). For example, if you want to save your work every 30 seconds, use the following expression for *When*:

```
Now + TimeValue("00:00:30"),
```

Listing 3.1 shows a macro that does this.

NOTE

You'll find the Word and Excel files used as examples in this chapter on my website at [www.mcfedries.com/Office2007Gurus](http://www.mcfedries.com/Office2007Gurus).

3

### Listing 3.1 A Macro That Saves the Active Document Every 20 Seconds

```
Public Sub FileSave()
    ActiveDocument.Save
    DoEvents
    Application.OnTime _
        When:=Now + TimeValue("00:00:20"), _
        Name:="FileSave"
    Application.StatusBar = "Saved: " & ActiveDocument.Name
End Sub
```

The `FileSave` macro begins by saving the current document using the `ActiveDocument` object's `Save` method. The `DoEvents` method processes any keystrokes that occurred during the save, and then the `OnTime` method sets up the `FileSave` procedure to run again in 20 seconds. The macro ends by displaying a status bar message to let you know the document was saved.

TIP

It's important that the macro in Listing 3.1 is named `FileSave` because this is also the internal name of Word's Save command (the one you run when you choose Office, Save). By giving your procedure the same name (and preceding the `Sub` keyword with `Public` to make it available to all documents), you intercept any calls to the Save command and replace Word's internal procedure with your own. This isn't strictly necessary, but it's handy because it means that your procedure will run as soon as the Office, Save command is chosen.

**TIP**

To find out the internal names of Word's commands, choose Office, Word Options, click Customize, and then click the Customize button to display the Customize Keyboard dialog box. Select an item in the Categories list and then look up the internal command name in the Commands list. Alternatively, hold down Ctrl and Alt, press the + key on your numeric keypad, and then click the Ribbon command associated with the command you want to use. Word displays the Customize Keyboard dialog box with only the selected command displayed.

## Closing a Document Without Saving

It's occasionally convenient to close a document without saving your changes. For example, you might create a new document to use as a scratch pad or to test out a new feature. Similarly, you might open an existing document, make some temporary changes, and then close the document without saving those changes. In each case, when you choose Office, Close, Word asks if you want to save the document. This not only slows you down by requiring an extra step that you don't need, but there's also the real danger that you might click Yes by accident.

To work around these problems, I use a macro that automatically closes a document without saving any changes and without prompting to save changes. Listing 3.2 shows the macro.

### Listing 3.2 A Macro That Closes a Document Without Saving Changes

```
Sub CloseWithoutSaving()
    ActiveDocument.Close SaveChanges:=wdDoNotSaveChanges
End Sub
```

As you can see, this is a simple procedure that runs only the `ActiveDocument` object's `Close` method with the `SaveChanges` argument set to the constant value `wdDoNotSaveChanges`.

A more involved example is shown in Listing 3.3.

### Listing 3.3 A Macro That Closes a Document Without Saving Changes and Then Reopens the Document

```
Sub CloseAndReopen()
    Dim currDoc As String, nState As Integer
    Dim nLeft As Integer, nTop As Integer
    Dim nWidth As Integer, nHeight As Integer
    '
    ' Preserve the window position and dimensions
    '
    With ActiveWindow
        nState = .WindowState
        nLeft = .Left
        nTop = .Top
```

```

        nWidth = .Width
        nHeight = .Height
    End With
    '
    ' Close it without saving changes
    '
    With ActiveDocument
        currDoc = .FullName
        .Close SaveChanges:=wdDoNotSaveChanges
    End With
    '
    ' Reopen the document
    '
    Documents.Open currDoc
    '
    ' Restore the window
    '
    With ActiveWindow
        .WindowState = nState
        If Not .WindowState = wdWindowStateMaximize Then
            .Left = nLeft
            .Top = nTop
            .Width = nWidth
            .Height = nHeight
        End If
    End With
End Sub

```

This macro first uses various properties of the `ActiveWindow` object to save the document's current window state, position, and dimensions. Then the code saves the document's full path name (as given by the `FullName` property), and the document is closed without saving changes. The `Open` method reopens the document, and then the window is restored to its former state, position, and dimensions.

### CAUTION

Be sure to move the `CloseAndReopen` macro to a document other than the one that you'll be closing and reopening. (For example, you could store it in the Normal template.) Otherwise, when the document is closed, the code module is closed along with it, so the document can't reopen.

## Closing All Your Open Documents

Later in this chapter, you learn how to create *workspaces*—collections of Word documents that you open as a unit. Before you open a workspace, it's a good idea to close all your open documents. You might also want to close all your open documents to get a fresh start with Word. In previous versions of Word, you can do this by holding down the Shift key, pulling down the File menu, and then selecting the Close All command. Unfortunately, that trick does not work in Word 2007. The other alternative is to shut down and restart Word, but that's often time-consuming. A faster method is to use the macro in Listing 3.4.

→ To learn how to create Word workspaces, see "Creating and Opening Document Workspaces," p. 70

**TIP**

The Close All command still exists in Word, but it's not part of the Ribbon. To add it to the Quick Access Toolbar, pull down the Customize Quick Access toolbar list and click More Commands. In the Choose Commands From list, click All Commands, scroll down the list of commands, and click Close All. Click Add and then click OK.

### Listing 3.4 A Macro That Closes All Open Documents

```
Sub CloseAllOpenDocuments()
    Dim doc As Document
    For Each doc In Documents
        doc.Close
    Next 'doc
End Sub
```

This macro runs through the Document objects that are in the Documents collection, which holds all the documents that are currently open in Word. For each document, the macro runs the Close method without any arguments, which means that Word will prompt you to save changes.

## Making Backups as You Work

Even if you're using the macros earlier in the chapter to save your work frequently, you can still lose data if your hard drive crashes. So we've all learned from hard experience not only to save our work regularly, but also to make periodic backup copies. The macro I use most often in Word is one that does both in a single procedure! That is, the macro not only saves your work, but it also makes a backup copy on another drive, such as a removable disk, a second hard drive, or a network folder. Listing 3.5 shows the code.

### Listing 3.5 A Procedure That Creates a Backup Copy of the Active Document on Another Drive

```
Sub MakeBackup()
    Dim currFile As String
    Dim backupFile As String
    Const BACKUP_FOLDER = "G:\Backups\"
    With ActiveDocument
        ' Don't bother if the document is unchanged or new
        '
        If .Saved Or .Path = "" Then Exit Sub
        '
        ' Mark current position in document
        '
        .Bookmarks.Add Name:="LastPosition"
        '
        ' Turn off screen updating
        '
    End With
```

```

        Application.ScreenUpdating = False
    '
    ' Save the file
    '
    .Save
    '
    ' Store the current file path, construct the path for the
    ' backup file, and then save it to the backup drive
    '
    currFile = .FullName
    backupFile = BACKUP_FOLDER + .Name
    .SaveAs FileName:=backupFile
End With
'
' Close the backup copy (which is now active)
'
ActiveDocument.Close
'
' Reopen the current file
'
Documents.Open FileName:=currFile
'
' Return to the pre-backup position
'
Selection.GoTo What:=wdGoToBookmark, Name:="LastPosition"
End Sub

```

The `backupFile` and `currFile` variables are strings that store the full pathnames for the active document and the backup version of the document. Use the `BACKUP_FOLDER` constant to specify the folder in which you want the backup stored.

The procedures first check to see if the backup operation is necessary. In other words, if the document has no unsaved changes (the `Saved` property returns `True`) or if it's a new, unsaved document (the `Path` property returns `" "`), bail out of the procedure (by running `Exit Sub`).

Otherwise, a new `Bookmark` object is created to save the current position in the document, screen updating is turned off, and the file is saved.

We're now ready to perform the backup. First, the `currFile` variable is used to store the full pathname of the document, and the pathname of the backup file is built with the following statement:

```
backupFile = BACKUP_FOLDER + .Name
```

This is used to save the file to the specified folder. The actual backup takes place via the `SaveAs` method, which saves the document to the path given by `backupFile`. From there, the procedure closes the backup file, reopens the original file, and uses the `GoTo` method to return to the original position within the document.

## TIP

Using a Bookmark object to reset the insertion point is useful because it takes you back to the exact point in the document where you were before the backup started. However, you may be interested only in returning to the last position within the document where an edit occurred. If that's the case, use the following statement in place of the `Selection.GoTo` statement:

```
Application.GoBack
```

Note, however, that there's a bug in the `GoBack` method, whereby Word doesn't save the last edit position (technically, it's a hidden bookmark named `\PrevSel1`) in some cases. Specifically, when you exit Word, if you elect to save changes in the last document that gets closed, Word doesn't save the last edit position in that document.

## 3

## Showing More Items on the Recent Documents List

When you pull down the Office menu in Word, you see the various “File” commands on the left and a list of your most recently used documents on the right. This list is called the Recent Documents list, and most Word installations display up to 17 documents on this list. If you regularly deal with many different Word documents, you might want to see even more files on this list. Follow these steps to increase the size of the Recent Document list:

1. Choose Office, Word Options to open the Word Options dialog box.
2. Click Advanced.
3. In the Display section, use the Show This Number of Recent Documents spin box to set the number of documents you want to see.

## NOTE

The maximum number of recent documents you can display is 50. If you specify more than your screen height can handle, Word just displays as many as it can.

4. Click OK.

## Opening the Most Recently Used Document at Startup

Word's `RecentFiles` object represents the collection of most recently used files displayed in the Recent Documents list. Each item on this list is a `RecentFile` object. You specify a `RecentFile` object by using `RecentFiles(Index)`, where *Index* is an integer that specifies the file you want to work with. The most recently used file is 1, the second most recently used file is 2, and so on.

When you close a Word session, the document you last made changes to becomes the most recently used document. There's a good chance that you'll want to continue working on that document the next time you start Word. Therefore, it would be handy to have Word open the most recently used file each time you start the program.



If you want Word to run some code each time it's started, open the Visual Basic Editor (either by choosing Developer, Visual Basic or by pressing Alt+F11), click the Normal project in the Visual Basic Editor's Project Explorer, and then create a new module (by choosing Insert, Module) named AutoExec. In this module, create a Sub procedure named Main and enter your code in that procedure. Listing 3.6 shows such a procedure:

NOTE

If you don't see the Developer tab in Word's Ribbon, choose Office, Word Options, click Popular, click to activate the Show Developer Tab in the Ribbon check box, and then click OK.

### Listing 3.6 A Procedure That Opens the Most Recently Used Document

```
Sub Main()  
    Application.RecentFiles(1).Open  
End Sub
```

3

## Clearing the Recent Documents List

The Recent Documents list is great for quickly reopening files you've worked with recently, particularly because you can increase the size of the list, as described earlier. It's not so great if other people have access to your computer because then they can easily see what you've been working on, and if you've been working on documents that contain sensitive data, the Recent Document list just makes life a bit *too* easy for a snoop. To increase security, you should remove sensitive items from the Recent Documents list. You can do this either by removing specific documents or by clearing the entire list.

To remove a single item from the Recent Documents list, follow these steps:

1. Press Ctrl+Alt+-. (Use the dash on the upper row of the keyboard, not the one on the numeric keypad.) The mouse pointer changes to a horizontal line.
2. Click the Office button.
3. In the Recent Documents list, click the document you want to clear.

For maximum privacy and security, you might prefer to clear the entire Recent Documents list (say, at the end of the day when you're done working or when you are leaving your desk for an extended time). You can do this by running the procedure shown in Listing 3.7.

### Listing 3.7 A Procedure That Removes All Items From the Recent Documents List

```
Sub ClearRecentFiles()  
    Dim rf As RecentFile  
    For Each rf In Application.RecentFiles  
        rf.Delete  
    Next 'rf  
End Sub
```

## TIP

If you don't want Word to store *any* items on the Recent Documents list, you can disable it. Choose Office, Word Options, click Advanced, and then set the Show This Number of Recent Documents spin box to 0.

## Creating and Opening Document Workspaces

When I work in Word (which I do pretty much all day long), I usually work with distinct groups of documents:

- If I'm working on a book, I open the current chapter, the outline, a notes document, a document to record screen shots, and so on.
- If I'm working on an article, I open the article, the proposal, my research notes, and so on.
- If I'm working on a blog post, I open the post and perhaps a few supporting documents.

It's not unusual for me to work on two, three, or even four such projects during the day. I like the variety, but it's a major pain to close all the documents for one project and then open all the documents I need for the next project. Perhaps that's why the Word macros I'm going to show you in this chapter are my favorites. Their purpose, as you'll see, is to create *workspaces* for Word projects that you can easily and quickly open.

→ Excel has a built-in command for creating workspaces; see “Creating a Workspace of Workbooks,” p. 153

A workspace is just a collection of related documents. Unfortunately, Word doesn't come with this functionality, but you can use VBA to create your own workspaces, as shown in Listing 3.8.

### Listing 3.8 Procedures That Create and Open a Workspace of Files

```
' CreateWorkspace()
' Saves the path and filename data of all the
' open files to the Windows Registry. Before
' running this procedure, make sure only the
' files you want in the workspace are open.
'
Sub CreateWorkspace(strWorkspaceName As String)
    Dim total As Integer
    Dim doc As Document
    Dim i As Integer
    '
    ' Delete the old workspace Registry settings
    ' First, get the total number of files
    '
    total = GetSetting("Word", strWorkspaceName, "TotalFiles", 0)
    For i = 1 To total
        '
        ' Delete each Registry setting
        '
    
```

```

        DeleteSetting "Word", strWorkspaceName, "Document" & i
    Next 'i
    '
    ' Create the new workspace
    '
    i = 0
    For Each doc In Documents
        '
        ' Make sure it's not a new, unsaved file
        '
        If doc.Path <> "" then
            '
            ' Use i to create unique Registry setting names
            '
            i = i + 1
            '
            ' Save the FullName (path and filename) to the Registry
            '
            SaveSetting "Word", strWorkspaceName, "Document" & i, doc.FullName
        End If
    Next 'doc
    '
    ' Save the total number of files to the Registry
    '
    SaveSetting "Word", strWorkspaceName, "TotalFiles", i
    Application.StatusBar = i & " documents saved to workspace."
End Sub
'
' OpenWorkspace()
' Accesses the Registry's workspace settings
' and then opens each workspace file.
'
Sub OpenWorkspace(strWorkspaceName As String)
    Dim total As Integer
    Dim i As Integer
    Dim filePath As String
    Dim doc As Document
    Dim fileAlreadyOpen As Boolean
    '
    ' See if we should first close all the open documents
    '
    If MsgBox("Close all the open documents first?", vbYesNo) = vbYes Then
        CloseAllOpenDocuments
    End If
    '
    ' Get the total number of files from the Registry
    '
    total = GetSetting("Word", strWorkspaceName, "TotalFiles", 0)
    For i = 1 To total
        '
        ' Get the path and filename
        '
        filePath = GetSetting("Word", strWorkspaceName, "Document" & i)
        '
        ' Make sure the file isn't already open
        '

```

*continues*

**Listing 3.8 Continued**

```

        fileAlreadyOpen = False
    For Each doc In Documents
        If filePath = doc.FullName Then
            fileAlreadyOpen = True
            Exit For
        End If
    Next 'doc
    ' Open it
    '
    If Not fileAlreadyOpen Then
        Documents.Open filePath
    End If
Next 'i
End Sub

```

Listing 3.8 shows two procedures that create the workspace functionality for Word:

- **CreateWorkspace**—This procedure uses the Windows Registry to store a list of open documents. Before running this procedure, make sure that only those files you want to include in the workspace are currently open.
- **OpenWorkspace**—This procedure first asks if you want to close all the open documents. (If you click Yes, the procedure runs the `CloseAllOpenDocuments` macro from Listing 3.4.) The procedure then accesses the Registry and runs through the list of saved files. For each setting, the procedure checks to see if the file is already open. If it's not, the procedure runs the `Documents.Open` method to open the file.

Notice that both procedures take `strWorkspaceName` as an argument. This is a string value that specifies the name of the workspace you want to create or open. This enables you to create as many different workspaces as you need. Here are two simple procedures that demonstrate how you'd use the macros in Listing 3.8 to create and open a workspace:

```

Sub CreateWorkspaceTest()
    CreateWorkspace "My Project"
End Sub

Sub OpenWorkspaceTest()
    OpenWorkspace "My Project"
End Sub

```

## Automatically Prompting for Document Properties

One of the interesting innovations in Windows Vista is the placement of document meta-data—properties such as the document author, subject, and status, as well as one or more tags that describe the document—at the heart of the operating system. In Vista, you can use metadata to sort documents, organize documents in to groups, filter documents, and search for documents. This is great in theory, but in practice the utility of metadata depends

entirely on how much of it there is. Some metadata is generated automatically by the program you use to create a document: the last modified date, the size of the file, and so on. However, most metadata is user-generated, so to get the most out of metadata, you need to remember to enter the data when you create new documents.

Older versions of Word had a setting that, when activated, configured Word to display the Properties dialog box each time you saved a new document for the first time. This was a great way to remember to enter metadata, but Word 2007 goes one better: It will also prompt you for metadata (by displaying the Document Information Panel) each time you open a document. Here's how to enable this setting in Word 2007:

1. Choose Developer, Document Panel to open the Document Information Panel dialog box.
2. Click to activate the Always Show Document Information panel On Document Open and Initial Save check box.
3. Click OK.

If you want an easier way to toggle this setting on and off, use the simple macro shown in Listing 3.9.

### Listing 3.9 A Macro That Toggles the Automatic Display of the Document Information Panel

```
Sub ToggleDocumentPanel()  
    With Application  
        .DisplayDocumentInformationPanel = Not .DisplayDocumentInformationPanel  
    End With  
End Sub
```

#### TIP

If you want Word to display the Properties dialog box instead of the Document Information Panel, use the following statement to activate this setting:

```
Application.Options.SavePropertiesPrompt = True
```

If you created an AutoExec module earlier (see "Opening the Most Recently Used Document at Startup"), insert the `Application.Options.SavePropertiesPrompt` statement into the `Main` procedure so that it runs each time you start Word.

## Creating a Trusted Location for Documents

Macro security is quite stringent in Office 2007, which shouldn't come as a surprise given the number of nasty VBA-based viruses and other malware that have spawned in the past few years. For a macro to work in Office 2007, the macro must come with a digital signature that is both valid and current, and the macro developer must be set up on your computer as a trusted publisher. To set up a developer as a trusted publisher, click Options when you see the Security Warning information bar, click to activate the Trust All Documents From This Publisher option, and then click OK.

What if you just want to run your own macros? In that case, you need to sign your own projects and then set yourself up as a trusted publisher.

**NOTE**

To digitally sign your own VBA projects, choose Start, All Programs, Microsoft Office, Microsoft Office Tools, Digital Certificate for VBA Projects. In the Create Digital Certificate dialog box, type a name (such as your own name) in the Your Certificate's Name text box and then click OK. When the SelfCert Success dialog box appears, click OK. Now open the Visual Basic Editor, select your project, and then choose Tools, Digital Signature. Click Choose, click your certificate, and then click OK.

If that all sounds like too much work, but you don't want to enable all macros, Word gives you a third choice: Store your macro-enabled documents in a trusted location. A *trusted location* is a folder that Word assumes contains only trustworthy documents, so it automatically enables any macros contained in those documents. By default, Word comes with three trusted locations:

%ProgramFiles%\Microsoft Office\Templates—This folder holds the application templates for all of Office.

%UserProfile%\Application Data\Microsoft\Templates—This folder holds the Office templates you create yourself.

%UserProfile%\Application Data\Microsoft\Word\Startup—This folder holds the global templates that you have created yourself and that are loaded into Word automatically at startup.

None of these locations are particularly convenient, but it's possible to set up a more suitable folder as a trusted location. Here are the steps to follow:

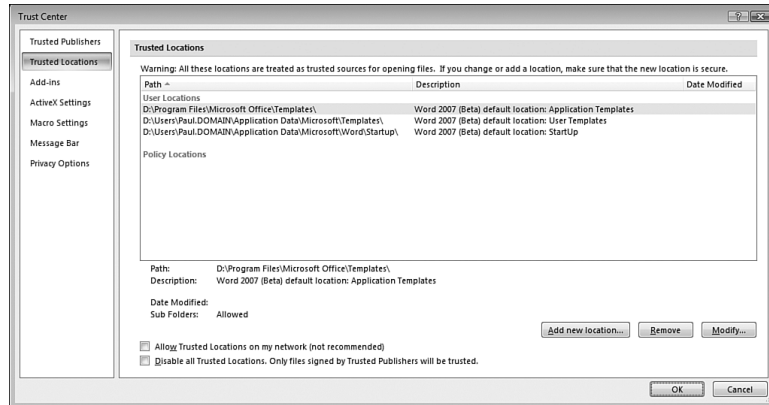
1. Choose Office, Word Options to open the Word Option dialog box.
2. Click Trust Center.
3. Click Trust Center Settings. The Trust Center dialog box displays.
4. Click Trusted Locations.

**NOTE**

While you're in the Trust Center, you might want to adjust your macro security setting. Click Macro Settings and then click the security option you prefer. If you're going to put all your macro-enabled documents in the trusted location that you're creating in these steps, you can click either Disable All Macros With Notification or Disable All Macros Without Notification.

5. If the trusted location is on your network, click to activate the Allow Trusted Locations On My Network check box.
6. Click Add New Location. The Microsoft Office Trusted Location dialog box displays, as shown in Figure 3.1.

**Figure 3.1**  
Use the Trusted Locations  
page to set up new  
trusted locations in Word.



7. Use the Path text box to type the path of the trusted location.
8. If you want Word to also trust this location's subfolders, click to activate the Subfolders of This Location Are Also Trusted check box.
9. Click OK.
10. Click OK to return to the Word Options dialog box.
11. Click OK.

## Viewing Total Editing Time Updated in Real-Time

The total amount of time that a document has been edited is useful for freelancers, lawyers, consultants, and other professionals who bill for their time. Knowing how long you have spent working on a document enables you to provide a more accurate accounting of your time.

If you have a time budget that you're trying to stick to, you may find yourself constantly checking the document's Properties dialog box to view the `Total editing time` value. Rather than wasting time performing that chore, add the `EditTime` field to your document. Word's `EditTime` field displays the total time, in minutes, that the document has had the system focus (and, presumably, has been edited) since the time at which the document was created. This is the same as the `Total editing time` value displayed in the Statistics tab of the document's Properties dialog box.

### NOTE

To display the Properties dialog box for the current document, choose **Developer, Document Panel** and click **OK** (or choose **Office, Prepare, Properties**). Pull down the Document Properties list and then click **Advanced Properties**.

Follow these steps to add some code that displays the total editing time with the word minutes added for clarity:

1. Type some descriptive text (such as **Total Editing Time:**).
2. Press Ctrl+F9 to start a new Word field (signified by the { and } braces).
3. Between the field braces, type the following:  
EDITTIME \# "0 minutes" \\* mergeformat
4. Press F9 to complete and update the field.

Now all you have to do is update the field to check the total editing time.

If even that sounds like a hassle, you can add a relatively simple macro to the document that will update the EditTime field in real-time. Open the Visual Basic Editor and insert a module in the document's project. Add the code in Listing 3.10 to the module.

### Listing 3.10 Updating the EditTime Field in Real-Time

```
Public Sub UpdateEditTime()
    Dim f As Field
    For Each f In ThisDocument.Fields
        If f.Type = wdFieldEditTime Then
            f.Update
        End If
    Next f
    Application.OnTime Now + TimeValue("00:01:00"), "UpdateEditTime"
End Sub
```

This procedure runs through all the fields in the document looking for one where the Type property is wdFieldEditTime, which corresponds to the EditTime field. When the property is found, the Update method refreshes the field. The key to the real-time updating is the Application.OnTime statement, which sets up the UpdateEditTime procedure to run again in one minute. Note that you'll probably need to adjust the project name ("Chapter03") and the module name ("Chapter3") to match the name of your project and module.

#### TIP

To constantly monitor the EditTime field as it's updated in real-time, you can place the field at the top or bottom of the document and then split the window (choose View, Split), so that the field remains visible in one pane. If that's not convenient, you can display the latest EditTime result in the status bar. In Listing 3.10, insert the following statement after the f.Update method:

```
Application.StatusBar = f.Result
```

The Field object's Result property returns the current result of the specified field.



## NOTE

If you're a professional writer or editor, you may be more concerned with the number of words in your documents. This value is available in the document's Properties dialog box, in the Statistics tab. However, you can also track this value outside of the dialog box by using a field. Insert the NumWords field and use the macro techniques in this section to update the field in real-time.

## Calculating Billable Time Charges

You saw in the previous section that you use the `EditTime` field to return the total editing time for a document. If you bill by the hour based on the amount of time you have worked on a document, you might want to keep track of how much money you've earned so far. If you earn, for instance, \$40 per hour, the following formula displays your current billable earnings:

```
{ = { EDITTIME } / 60 * 40 }
```

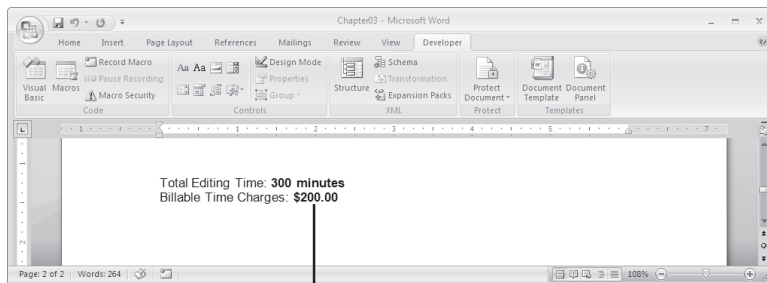
The `EditTime` result is given in minutes, so you have to divide by 60 to get the number of hours. You then multiply that result by 40 to get the earnings.

To ensure accurate billing, you may want to use the `ROUND` function to round the result to the nearest dollar:

```
{ = ROUND( { EDITTIME } / 60 * 40, 0) \#$0.00 }
```

This formula field also uses a numeric format to display the result with a dollar sign and two decimal places, as shown in Figure 3.2.

**Figure 3.2**  
A formatted formula field that calculates the billable charge based on the current *EditTime* result.



Formula field

## Locking Document Formatting

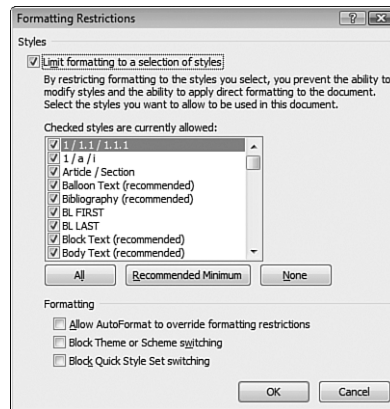
Like most modern word processors, Word fits into the category of *fritterware*—programs with so many formatting bells and whistles that you can end up frittering away hours and hours by tweaking fonts, colors, alignments, and so on. Whether you think of such activity as “frittering” depends on your point of view, but we all agree that a well-formatted document makes a better impression than a plain or sloppy-looking one. So no matter how much time you’ve devoted to getting your document just so, the last thing you want is another person running roughshod over your careful look and layout.

Fortunately, Word offers the capability to lock your document's formatting, which prevents others from changing the formatting unless they know the password. Here are the steps to follow:

1. Choose Review, Protect Document (or choose Developer, Protect Document) and then click Restrict Formatting and Editing. Word displays the Restrict Formatting and Editing task pane.
2. Click to activate the Limit Formatting to a Selection of Styles check box.
3. Click Settings to display the Formatting Restrictions dialog box, shown in Figure 3.3.

**Figure 3.3**

Use the Formatting Restrictions dialog box to restrict the formatting another user can apply to a document.



4. In the Checked Styles Are Currently Allowed list, deactivate the check box next to each style that you want to disallow. Alternatively, use the following buttons to set the check boxes:
  - **All**—Click this button to activate all the check boxes and thus enable unauthorized users to apply formatting using only the existing styles; these users cannot modify the existing styles or create new styles.
  - **Recommended Minimum**—Click this button to activate the check boxes for only those styles that Word determines are necessary for the document.
  - **None**—Click this button to deactivate all the check boxes and thus prevent unauthorized users from changing any document formatting.
5. Choose your formatting options:
  - **Allow AutoFormat to Override Formatting Restrictions**—Click to activate this check box if you want any AutoFormats that the user applies to affect restricted styles.
  - **Block Theme or Scheme Switching**—Click to activate this check box to prevent the user from changing formatting by applying a formatting theme or scheme.
  - **Block Quick Style Set Switching**—Click to activate this check box to prevent the user from changing formatting by applying a Quick Style.

6. Click OK.
7. If Word warns you that the document contains disallowed styles, click Yes to remove them or click No to keep them.
8. In the Restrict Formatting and Editing task pane, click Yes, Start Enforcing Protection. The Start Enforcing Protection dialog box displays.
9. Type the password twice and then click OK. Word disables all the formatting commands on the Ribbon.

If you or another authorized user need to change the document formatting, choose Review, Protect Document (or choose Developer, Protect Document), click Restrict Formatting and Editing, and then click Stop Protection. Type the password, click OK, and then deactivate the Limit Formatting to a Selection of Styles check box.

## Preventing Untracked Changes

When you share a Word document, it's common to turn on the Track Changes feature (choose Review, Track Changes) so that you can see all the edits made by the other person. Unfortunately, it's easy to turn off Track Changes (either accidentally or on purpose) so you can never be sure whether your document contains any untracked changes.

If it's important that you track all edits, it's possible to set up Word to prevent untracked changes and even to enforce this option with a password. Here are the steps to follow:

1. Choose Review, Protect Document (or choose Developer, Protect Document) and then click Restrict Formatting and Editing. Word displays the Restrict Formatting and Editing task pane.
2. Click to activate the Allow Only This Type of Editing in the Document check box.
3. In the Editing Restrictions list, click Tracked Changes.
4. Click Yes, Start Enforcing Protection. The Start Enforcing Protection dialog box displays.
5. Type the password twice and then click OK.

### TIP

If you're using Word in a Windows domain or an Exchange shop, you can set up portions of a document as read-only, and you can make exceptions for certain users. In the Restrict Formatting and Editing task pane, use the Editing Restrictions list to select No Changes (Read-only). If you want users to edit only a portion of the document, select that portion. In the Exceptions group, click More Users and then type the usernames or email addresses (separated by semicolons) of the users you want to be able to freely edit the selected portion. In the Individuals list, activate the check box next to each user.

If you or another authorized user need to freely edit the document text, choose Review, Protect Document (or choose Developer, Protect Document), and then Restrict Formatting and Editing to display the Restrict Formatting and Editing task pane. Click Stop Protection, type the password, click OK, and then deactivate the Allow Only This Type of Editing in the Document check box.

## Setting Up a Document for Structured Editing



A common business scenario involves creating an overall design for a document, filling in some standard text, and then leaving a few sections blank for other users to fill in. In a company report, for example, the design and most of the content might be fixed, but it might also have sections in which different departments can enter results, mission statements, goals, and so on. In these situations, it's crucial that the other users who add their content to the document do not also change the document design or any of the other content.

The previous two sections showed you how to lock document formatting and prevent untracked changes. You can also designate a document to be read-only (choose Office, Prepare, Mark as Final) so that no changes can be made. However, none of these techniques solve the problem of preventing users from editing or deleting parts of a document, while also allowing them the ability to add specific types of content. This is called *structured editing*, and Word 2007 offers a number of other tools that make it possible.

The first part of structured editing involves setting up regions of the document so that other people can't edit them or delete them. In Word 2007, you do that by forming those regions into a *group*. By default, a group cannot be edited or formatted, and you can also configure a group so that it cannot be deleted.

To convert text into a group, select the text and then choose Developer, Group, Group. Word locks the group, although you don't see anything on screen to tell you this. (This is by design: Your document looks exactly as it did before, so the quality and design of the document is not changed by creating a group.) If you also want to prevent users from deleting all or part of the group, follow these steps:

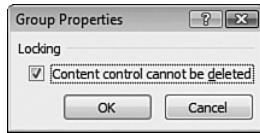
1. Click inside the group.
2. Choose Developer, Properties to open the Group Properties dialog box shown in Figure 3.4.
3. Click to activate the Content Control Cannot Be Deleted check box.
4. Click OK.

### TIP

In most cases, you probably don't want users to insert any content to the document, except in the sections that you designate (more on this later in this section). To ensure this, select the entire document and then run the Group command. This locks the whole document against editing.

**Figure 3.4**

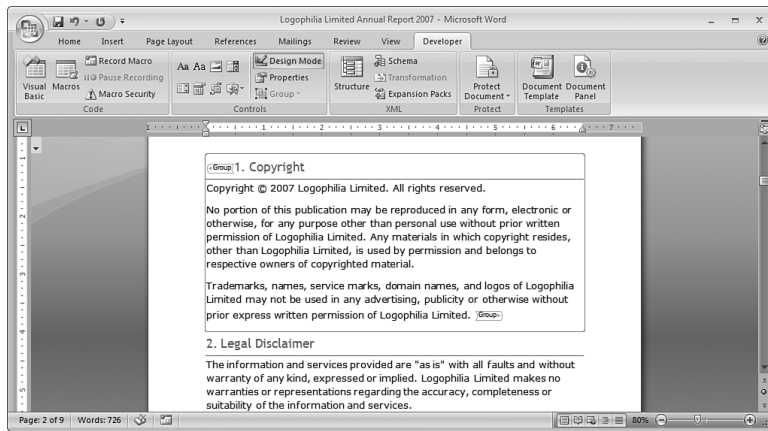
Use the Group Properties dialog box to prevent the grouped text from being deleted.



If you need to work with the grouped content or if you just want to remind yourself of the extent of the group, choose Developer, Design Mode. As you can see in Figure 3.5, Word adds “Group” icons at the beginning and end of the grouped content and also displays a box around the content when you click inside the group.

**Figure 3.5**

Choose Developer, Design Mode to see the Group icons.



**NOTE**

To remove the group, switch to Design Mode, select the entire group, and then close Developer, Group, Ungroup. If the Ungroup command is disabled, choose Developer, Properties, click to deactivate the Content Control Cannot Be Deleted check box, click OK, and then try again.

The second part of structured editing involves allowing users to add content to the document, although you don't allow just any content to be added anywhere inside the document. Instead, the editing is “structured” because you place one or more of the following restrictions on content additions:

- You designate a precise location within the location for the content.
- You designate the type of content that the user can add (such as a date or a table).
- You assist the user by offering some kind of user interface feature that leads the user to enter exactly the type of data you want.

You can set up any of these restrictions by using Word 2007's new *content controls*, which are dialog box-like controls that you can draw directly inside your document. The idea is that

you draw each control exactly where you want it, and the control dictates the type of data the user can enter and often comes with some sort of user interface that helps the user enter the data.

In the Developer tab's Controls group, Word 2007 offers seven content controls (see Figure 3.6):

**Rich Text**—Use this control for formatted text that consists of one or more paragraphs.

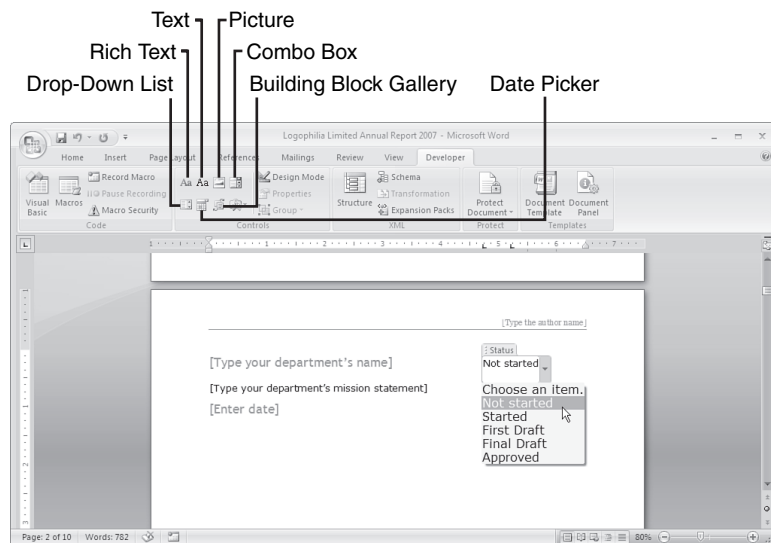
**Text**—Use this control for plain text that consists of no more than one paragraph. (If you need plain text for multiple paragraphs, click the control, choose Developer, Properties, and then click to activate the Allow Carriage Returns [Multiple Paragraphs] check box.)

**Picture**—Use this control to enable the user to insert a picture into the document. The user clicks the picture icon and then uses the Insert Picture dialog box to select the image.

**Combo Box**—Use this control to enable the user to enter text or select an item from a drop-down list. To add items to the list, click the control, choose Developer, Properties, and then click **Add**. In the Add Choice dialog box, type a Display Name and a Value, and then click OK. Repeat to add other list items. Figure 3.6 shows an example combo box control.

**Figure 3.6**

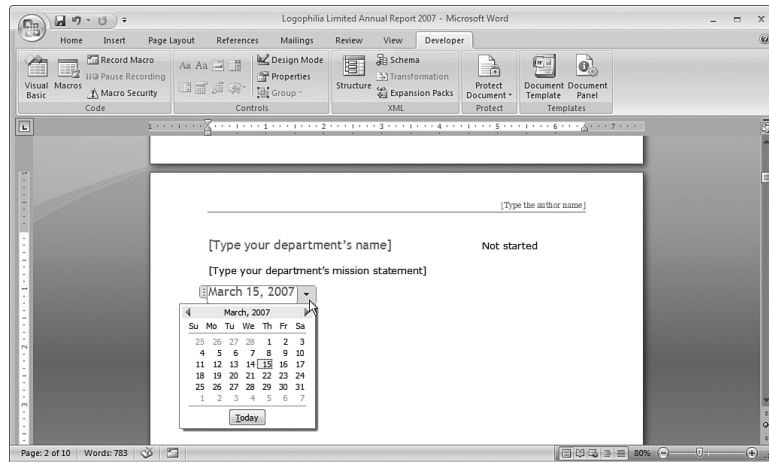
Use a Combo Box content control to enable the user to type text or select an item from a list.



**Drop-Down List**—Use this control to enable the user to select an item from a drop-down list. You add items to the list using the same method as I described for the Combo Box control.

**Date Picker**—Use this control to enable the user to enter a date by clicking the date from a calendar, as shown in Figure 3.7. To specify the date format, click the control, choose **Developer**, **Properties**, and then click the format in the **Display the Date Like This** list.

**Figure 3.7**  
Use a Date Picker content control to enable the user to specify a date by clicking it in a calendar.



**Building Block Gallery**—Use this control to enable the user to select an item from the Building Blocks Gallery. If you want the user to have a choice of a specific type of building block only, click the control, choose **Developer**, **Properties**, and then use the **Gallery** list to click the building block type (AutoText, Quick Parts, Tables, and so on).

Here are the general steps to follow to insert a content control into a document:

1. Position the insertion point where you want the content control to display.
2. Choose the **Developer** tab, and then click the content control you want. Word inserts the control.
3. Choose **Developer**, **Properties** to open the **Content Control Properties** dialog box.
4. Type a **Title** for the content control. (This text displays above the control when the user clicks it.)
5. (Optional) Type a **Tag** for the content control. (This text displays on either side of the control when you switch to **Design Mode**.)
6. If you want to apply a style to the control contents, click to activate the **Use a Style to Format Contents** check box and then use either **Style** or **New Style** to specify the style you want to use.
7. To prevent the user from deleting the control, click to activate the **Content Control Cannot Be Deleted** check box.

8. After the control has been edited, you can prevent anyone else from making changes to it by clicking to activate the Contents Cannot Be Edited check box.
9. Configure any control-specific settings, as described earlier.
10. Click OK.

## Inspecting a Document for Personal Information



Earlier in this chapter, I showed you how to get Word to automatically prompt you for document properties because saving metadata is a good idea for most documents. However, it's not such a good idea if you are sharing a document with other people, particularly people outside of your organization. That's because the metadata might contain private or sensitive data that you probably don't want outsiders to see. This also applies to other document data, such as reviewers' comments and annotations.

Removing this kind of data by hand is not only time-consuming, but it's also easy to miss a thing or two. To help out, Word (and the other main Office 2007 programs) comes with a Document Inspector that can search for potentially private data and remove it from the document automatically. The Document Inspector can remove the following document data:

- Document properties
- Headers, footers, watermarks, and hidden text
- Personal information, such as your username and your personal summary information
- Document versions
- Reviewer comments and annotations
- Custom XML data

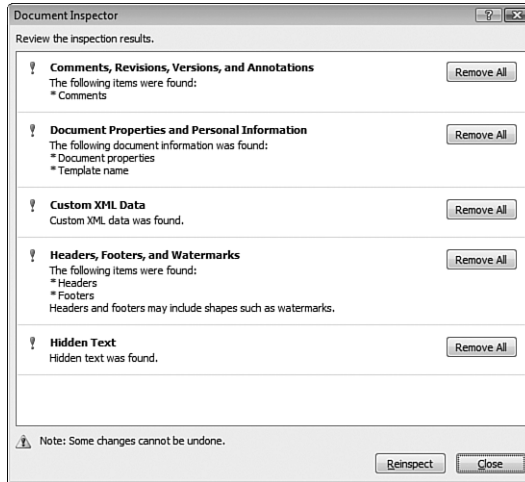
Follow these steps to use the Document Inspector:

1. Save the document. If you want to keep an internal version that maintains the personal information, choose Office, **S**ave **A**s and save a copy of the document that you can then share.
2. Choose Office, **P**repare, **I**nspect Document. Word opens the Document Inspector.
3. Click to deactivate the check box next to any content types you don't want to check.
4. Click **I**nspect. The Document Inspector checks the document, and then it displays the results of the inspection, as shown in Figure 3.8.
5. For each type of data you want to delete from the document, click the Remove All button.
6. When you're done, click **C**lose.



**Figure 3.8**

Use the Document Inspector to look for potentially private or sensitive data before sharing a document.



## Viewing Two Documents Side by Side



When you share a document with another person, you usually turn on revision marks, so that you can see the changes the other person makes. Unfortunately, it often happens that the other person accidentally turns off revision marks before or in the middle of editing so that you can no longer easily see the changes he made. In the past, this meant opening the two versions and then switching from one window to another or trying to arrange the documents, so that you could see both at once.

Word 2007 helps you avoid this extra work by offering a new View Side By Side feature, which arranges the document windows so that you can easily compare the two files. Here's how it works:

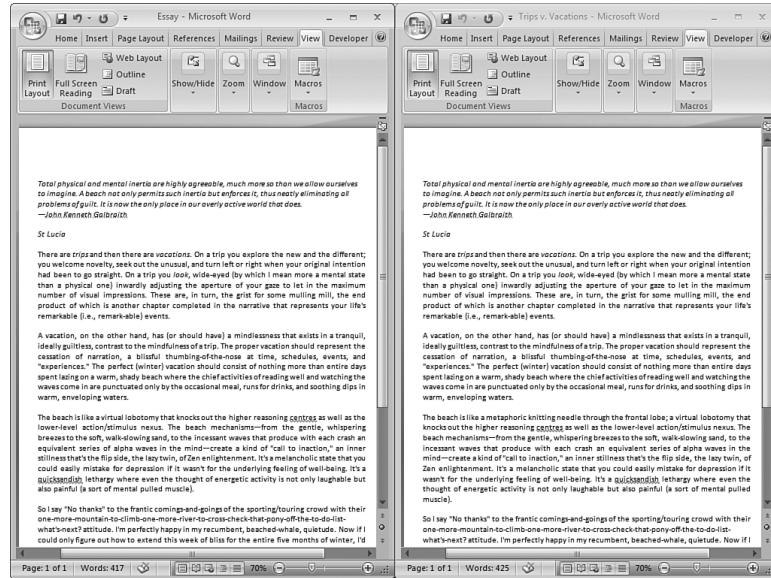
1. Open the two documents you want to compare and make one of them the active document.
2. Choose View, View Side By Side. Word displays the Compare Side By Side dialog box.
3. Click the other document you want to use and then click OK. Word arranges the two windows side by side, as shown in Figure 3.9.

### NOTE

The View Side By Side feature includes *synchronous scrolling*, which means that as you scroll up or down (or even side to side) in one document, Word scrolls in the same direction and by the same amount in the other document. If you find this distracting, you can turn it off by choosing View, Window, Synchronous Scrolling.

4. When you're done, choose View, Window, Compare Side By Side.

**Figure 3.9**  
Activate the View Side  
By Side command to  
compare two documents  
side by side.



### TIP

You can get Word to help you compare the two documents. Choose **Review, Compare, Compare to** to open the Compare Documents dialog box. Use the **Original Document** list to specify the original document and use the **Revised Document** list to select the document with changes. If you want to specify what changes to look for, click **More** and use the check boxes to modify the Comparison Settings. Click **OK**, and Word opens a Compare Result window, which includes the Original Document pane, the Revised Document pane, the Reviewing Pane with a list of the changes, and a Compared Document pane that combines the changes into a single document.

## Updating All Fields Automatically

When you're working with fields, it's common to need to update all of a document's fields at one time. One way to do this is to select the entire document and press F9. This works, but it's a hassle because not only must you perform the extra step of selecting the entire document, but that extra step also means that you lose your current cursor position.

To avoid this problem, use the VBA macro in Listing 3.11 to update all the document's fields.

### Listing 3.11 A Macro to Update All the Fields in the Active Document

```
Sub UpdateAllFields()  
    ActiveDocument.Fields.Update  
End Sub
```

Add this macro to a Quick Access toolbar button or assign it a keyboard shortcut (I use Ctrl+Alt+Shift+F9).

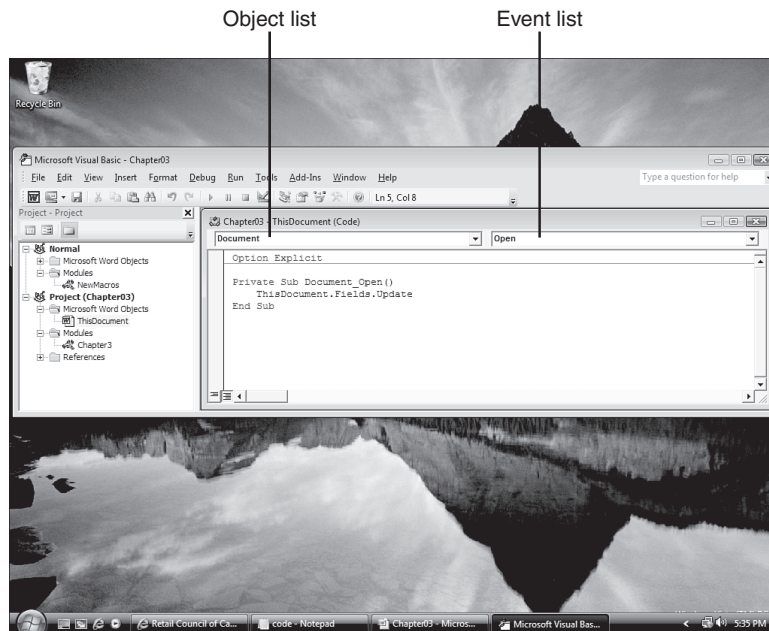
It's also often useful to update all of a document's fields when the document is opened. Word doesn't do that by default, but you can create a macro that does.

Launch the Visual Basic Editor and open the project that corresponds to your document. In the project's Microsoft Word Objects branch, double-click the `ThisDocument` object. In the code window that displays, select `Document` in the object list and select `Open` in the event list. Add the following statement to the `Document_Open()` stub that displays:

```
ThisDocument.Fields.Update
```

Figure 3.10 shows the completed code.

**Figure 3.10**  
Add this code to your document's `Open` event to update all fields each time the document is opened.



## From Here

- To learn how to add a custom watermark to a document, **see** “Creating a Custom Watermark,” **p. 93**.
- To learn how to protect Excel worksheets, **see** “Preventing Users from Changing Parts of a Worksheet,” **p. 148**.
- Excel has a built-in command for creating workspaces; **see** “Creating a Workspace of Workbooks,” **p. 153**.
- To learn how to use the Visual Basic Editor and incorporate macros into your documents, **see** “Working with VBA Macros,” **p. 429**.