Solve Real World Problems with Excel

Programming

Excel *with* VBA and .NET

O'REILLY®

Jeff Webb &
Steve Saunders

Programming Excel with VBA and .NET
by Jeff Webb and Steve Saunders

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Editors: Simon St.Laurent, John Osborn
Production Editor: Sanders Kleinfeld
Copyeditor: Norma Emory
Indexer: Ellen Troutman-Zaig

Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrators: Robert Romano, Jessamyn Read, and Lesley Borash

Printing History:

April 2006: First Edition.

ISBN: 0-596-00766-3
[M]

CHAPTER 7

Controlling Excel

I talked a little about the `Application` object back in Chapter 4. `Application` is where everything starts in Excel: it's the grandma of all the other objects. You use the `Application` object to:

- Perform top-level actions, such as quitting Excel, showing dialog boxes, or recalculating all workbooks
- Control the Excel options, such as the settings on the Tools → Options dialog box
- Get references to the other objects in Excel

In this chapter, you will learn about those tasks in detail. This chapter includes task-oriented reference information for the following objects: `Application`, `AutoCorrect`, `AutoRecover`, `ErrorChecking`, `Windows`, and `Panes`.

Code used in this chapter and additional samples are available in *ch07.xls*.

Perform Tasks

Use `Application` object to perform top-level tasks in Excel. The following sections describe how to:

- Quit the Excel application from code
- Turn user interaction and screen updates off and on
- Open, close, and arrange Excel windows
- Display Excel dialog boxes

These are the most common tasks for the `Application` object.

Quit Excel

Use the `Quit` method to quit Excel. If there are any workbooks with unsaved changes, Excel displays a dialog box asking the user if those changes should be saved. There are several ways to change that behavior:

- Save all workbooks before quitting.
- Set the all workbooks `Saved` property to True.
- Set `DisplayAlerts` to False.

The following code shows how to save all open workbooks before closing without prompting the user:

```
Sub QuitSaveAll( )
    Dim wb As Workbook
    For Each wb In Workbooks
        wb.Save
    Next
    Application.Quit
End Sub
```

Conversely, this code quits Excel without saving any of the workbooks:

```
Sub QuitSaveNone( )
    Dim wb As Workbook
    For Each wb In Workbooks
        ' Mark workbook as saved.
        wb.Saved = True
    Next
    Application.Quit
End Sub
```

Setting the `Saved` property fools Excel into thinking that it doesn't need to save changes and they are lost when Excel quits.

There's one other handy member to know about when quitting Excel: the `SaveWorkspace` method lets you save an *.xlw* file that you can use to restore the workbooks and windows currently in use. The following code saves those settings as *Resume.xlw*:

```
Sub QuitWithResume( )
    Application.SaveWorkspace "Resume.xlw"
    Application.Quit
End Sub
```

Lock Out User Actions

Sometimes you want to prevent users from interrupting Excel while you perform some time-consuming task in code. The `Application` object provides these ways to limit user interaction:

- Set `DisplayAlerts` to False to hide standard Excel dialogs while code runs

- Set `Interactive` to False to lock users out of Excel completely
- Set `ScreenUpdating` to False to hide changes as they are made by code

Each of these approaches should include some code at the end of the procedure to change the settings back to their defaults when your code finishes. Otherwise, you might lock a user out permanently!

The following code demonstrates how to lock out user actions temporarily while a long task executes:

```
Sub LockOutUser( )
    Dim cel As Range
    ' Show the hourglass cursor.
    Application.Cursor = xlWait
    ' Turn off user interaction, screen updates.
    Application.Interactive = False
    Application.ScreenUpdating = False
    ' Simulate a long task.
    For Each cel In [a1:iv999]
        cel.Select
    Next
    ' Restore default settings.
    Application.Interactive = True
    Application.ScreenUpdating = True
    Application.Cursor = xlDefault
    [a1].Select
End Sub
```

One of the side benefits of setting `ScreenUpdating` to False is that the preceding code executes more quickly since Excel doesn't have to update the screen or scroll the worksheet as cells are selected. Again, just be sure to turn screen updates back on when done.

Open and Close Excel Windows

The `Application` object provides a `Windows` collection that lets you open, arrange, resize, and close Excel's child windows. For example, the following code opens a new child window and then cascades the open windows for the active workbook:

```
Sub OpenCascadeWindows( )
    ActiveWindow.NewWindow
    Application.Windows.Arrange xlArrangeStyleCascade, True
End Sub
```

You close and maximize child windows using methods on the `Window` object. For example, the following code closes the window opened in the preceding code and restores the original window to a maximized state in Excel:

```
Sub CloseMaximize( )
    ActiveWindow.Close
    ActiveWindow.WindowState = xlMaximized
End Sub
```

Closing the last child window for a workbook also closes the workbook.

Finally, you can control the Excel parent window using the `Application` object's `WindowState` and `DisplayFullScreen` properties:

```
Sub ChangeExcelWindowState( )
    Application.WindowState = xlMaximized
    API.Sleep 1000
    Application.WindowState = xlMinimized
    API.Sleep 1000
    Application.WindowState = xlNormal
    API.Sleep 1000
    Application.DisplayFullScreen = True
    API.Sleep 1000
    Application.DisplayFullScreen = False
End Sub
```

Display Dialogs

The three different sorts of dialog boxes in Excel are built-in dialogs that perform actions, built-in dialogs that return information, and custom dialogs you build from Visual Basic forms. The `Application` object gives you several ways to display the first two types:

- Use the `FindFile` method to let the user select a file to open in Excel.
- Use the `Dialogs` collection to display Excel's other built-in dialog boxes to perform those specific actions.
- Use `FileDialog` method to get file and folder names from the user.
- Use the `InputBox` method to get ranges or formulas.

For example, the following code displays Excel's built-in Open dialog box and then opens the file selected by the user:

```
Sub OpenFile1( )
    On Error Resume Next
    Application.FindFile
    If Err Then Debug.Print "User cancelled import."
End Sub
```

You can do the same thing using the `Dialogs` collection:

```
Sub OpenFile2( )
    On Error Resume Next
    Application.Dialogs(XlBuiltInDialog.xlDialogOpen).Show
    If Err Then Debug.Print "User cancelled import."
End Sub
```

Both of the preceding samples display the Open dialog box and open the file in Excel. You have to include error-handling statements in case the user chooses a non-Excel file then cancels importing the file—otherwise that action halts your code with an application error.

The Dialogs collection can display any of the Excel dialog boxes. See Appendix A for a list of those dialogs—about 250 of them! Displaying a dialog that way is just like displaying it through the user interface: Excel uses its current settings and takes whatever actions the user chooses from the dialog.

Sometimes you *don't* want Excel to perform its standard action after the user closes the dialog; instead, you'd rather get the information from the dialog and take your own actions in code. The most common example of this is when you want to get a file or folder name. In that case, use the FileDialog method.

FileDialog displays the built-in Excel Open dialog box, but doesn't open the file. You can change the caption, file filter, and other settings as well. The following code uses the FileDialog to open a web file in the browser:

```
Sub OpenWebFile( )
    With Application.FileDialog(msoFileDialogFilePicker)
        ' Set dialog box options
        .Title = "Show web file"
        .Filters.Add "Web files (*.htm)", "*.htm;*.html;*.xml", 1
        .FilterIndex = 1
        .AllowMultiSelect = False
        ' If the user chose a file, open it in the browser.
        If .Show = True Then _
          ThisWorkbook.FollowHyperlink .SelectedItems(1)
    End With
End Sub
```

Finally, the Application object's InputBox method lets you get Excel ranges and formulas from the user. This method is otherwise identical to the Visual Basic InputBox. Figure 7-1 shows the Excel InputBox in action.

The *Type* argument of InputBox determines the kind of data the user can enter. The most common settings are 0 for a formula, 1 for a number, or 8 for a range. The following code displays the input box shown in Figure 7-1:

```
Sub GetRange( )
    Dim rng As Range
    Set rng = Application.InputBox("Select a range", _
      "Application InputBox", , , , , , 8)
    rng.Select
End Sub
```

Control Excel Options

All of the Excel settings and options can be controlled in code through Application object properties. Quite a few of the Application properties are devoted to Excel settings and options, but you only occasionally need to change these settings in code—it is usually a better idea to let the users maintain their own settings.

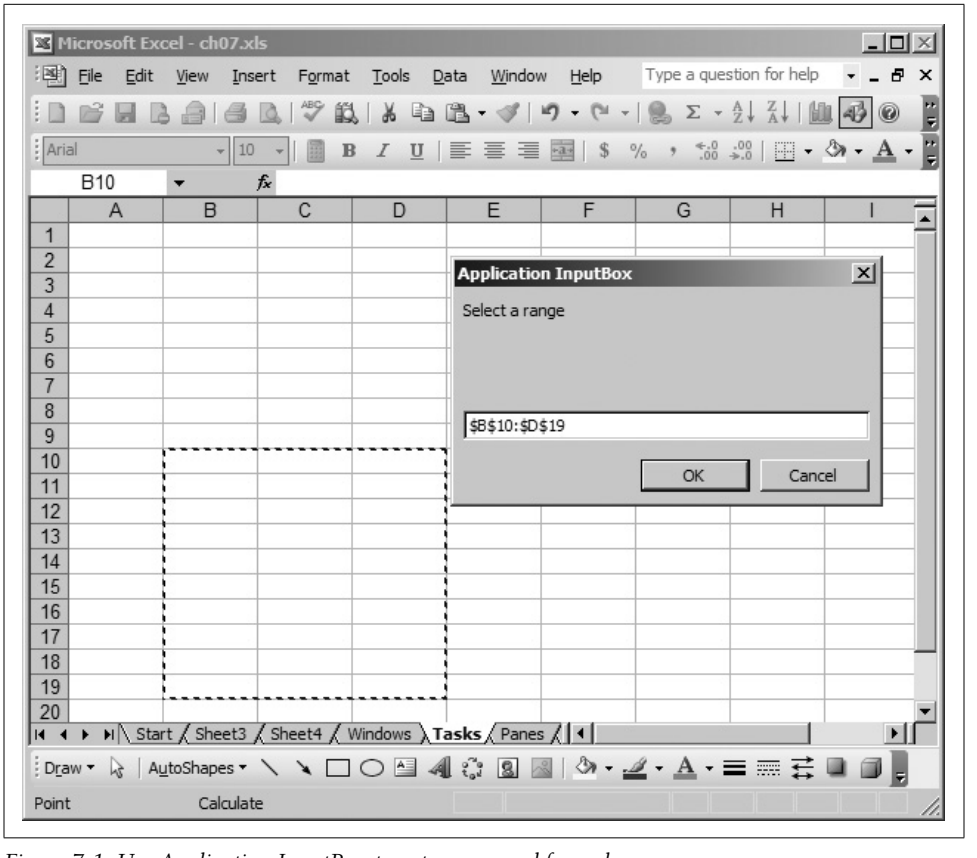

Figure 7-1. Use Application.InputBox to get ranges and formulas

If you do change Excel options in code, it is polite to restore the user's settings when you are done. To do that, save the original setting in a module-level variable and restore that setting before exiting.

Set Startup Paths

Excel uses several predefined folders to load workbooks, add-ins, and templates. You can get or set these folders from code using the properties in Table 7-1.

Table 7-1. Application properties for predefined folders

Property	Use to
`AltStartupPath`	Get or set the user folder used to load add-ins and workbooks automatically
`DefaultFilePath`	Get or set the default folder to which workbooks are saved
`LibraryPath`	Get the built-in Excel add-in library folder
`NetworkTemplatesPath`	Get the `AltStartupPath` if it is a network share

Table 7-1. Application properties for predefined folders (continued)

Property	Use to
`Path`	Get the folder where Excel is installed
`StartupPath`	Get the built-in folder Excel uses to load add-ins and workbooks automatically (*XLSTART*)
`TemplatesPath`	Get the user folder Excel from which loads templates

You use these properties when installing templates and add-ins, as covered in Chapter 6, and when your code relies on specific locations. For example you might want to change the `DefaultFilePath` to a specific folder while your application runs:

```
Dim m_originalPath As String
Const APP_PATH = "c:\ExcelDocs"

Sub SetPath( )
    ' Store the user settng.
    m_oringalPath = Application.DefaultFilePath
    ' Use this setting while application runs.
    Application.DefaultFilePath = APP_PATH
End Sub

Sub RestorePath( )
    ' Restore the user setting before exit.
    Application.DefaultFilePath = m_originalPath
End Sub
```

View System Settings

There are a great many other settings and options in Excel. Chapter 6 showed how to find operating system and version information from the `Application` object. You can also get and set the options set through the Excel Options dialog box (Figure 7-2) using individual `Application` properties.

For example, to select the R1C1 reference style in Figure 7-2, use this code:

```
Sub SetReferenceStyle( )
    Application.ReferenceStyle = xlR1C1
End Sub
```

Get References

As the top-level object in Excel, `Application` is the source of all other object references. However, the object name `Application` isn't always used in code because Excel includes shortcuts (called *global members*) that let you omit it. For instance, the following two lines are equivalent:

```
Application.Selection.Clear ' Clear selected cells.
Selection.Clear             ' Same thing!
```

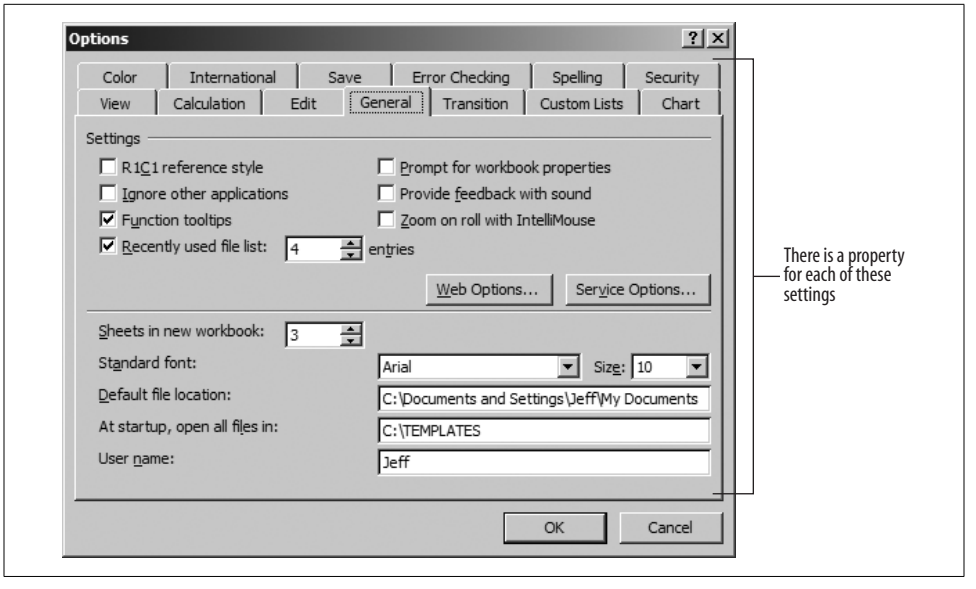

Figure 7-2. Use Application properties to get or set these options

In this case, `Selection` returns the selected cells on the active worksheet as a `Range` object. Table 7-2 lists the Application members that return references to other objects.

Table 7-2. Application object members that return object references

`ActiveCell`	`ActiveChart`	`ActivePrinter`
`ActiveSheet`	`ActiveWindow`	`ActiveWorkbook`
`AddIns`	`Assistant`	`AutoCorrect`
`AutoRecover`	`Cells`	`Charts`
`Columns`	`COMAddIns`	`CommandBars`
`Dialogs`	`ErrorCheckingOptions`	`FileDialog`
`FileFind`	`FileSearch`	`FindFile`
`FindFormat`	`International`	`Intersect`
`LanguageSettings`	`Names`	`NewWorkbook`
`ODBCErrors`	`OLEDBErrors`	`PreviousSelections`
`Range`	`RecentFiles`	`Rows`
`RTD`	`Selection`	`Sheets`
`SmartTagRecognizers`	`Speech`	`SpellingOptions`
`ThisCell`	`ThisWorkbook`	`Union`
`UsedObjects`	`Watches`	`Windows`
`Workbooks`	`WorksheetFunction`	`Worksheets`

Most of the names of the members in Table 7-2 are descriptive of the objects they return. The exceptions to that rule are the members that can return a mixed collec-

tion of objects, such as Selection, and members that return Range objects: ActiveCell, Cells, Columns, Range, Rows, and ThisCell.

Application Members

The Application object has the following members. Key members (shown in **bold**) are covered in the following reference section:

ActivateMicrosoftApp	ActiveCell	ActiveChart
ActivePrinter	ActiveSheet	ActiveWindow
ActiveWorkbook	**AddChartAutoFormat**	**AddCustomList**
AddIns	**AlertBeforeOverwriting**	**AltStartupPath**
Application	**ArbitraryXMLSupportAvailable**	**AskToUpdateLinks**
Assistant	**AutoCorrect**	**AutoFormatAsYouType ReplaceHyperlinks**
AutomationSecurity	**AutoPercentEntry**	**AutoRecover**
Build	**Calculate**	**CalculateBeforeSave**
CalculateFull	**CalculateFullRebuild**	**Calculation**
CalculationInterruptKey	**CalculationState**	**CalculationVersion**
Caller	CanPlaySounds	CanRecordSounds
Caption	**CellDragAndDrop**	**Cells**
CentimetersToPoints	**Charts**	**CheckAbort**
CheckSpelling	**ClipboardFormats**	ColorButtons
Columns	**COMAddIns**	**CommandBars**
CommandUnderlines	**ConstrainNumeric**	**ControlCharacters**
ConvertFormula	**CopyObjectsWithCells**	Creator
Cursor	**CursorMovement**	**CustomListCount**
CutCopyMode	**DataEntryMode**	**DecimalSeparator**
DefaultFilePath	**DefaultSaveFormat**	**DefaultSheetDirection**
DefaultWebOptions	**DeleteChartAutoFormat**	**DeleteCustomList**
Dialogs	**DisplayAlerts**	**DisplayClipboardWindow**
DisplayCommentIndicator	**DisplayDocumentActionTaskPane**	**DisplayExcel4Menus**
DisplayFormulaBar	**DisplayFullScreen**	**DisplayFunctionToolTips**
DisplayInsertOptions	**DisplayNoteIndicator**	**DisplayPasteOptions**
DisplayRecentFiles	**DisplayScrollBars**	**DisplayStatusBar**
DisplayXMLSourcePane	**DoubleClick**	**EditDirectlyInCell**
EnableAnimations	**EnableAutoComplete**	**EnableCancelKey**
EnableEvents	**EnableSound**	**ErrorCheckingOptions**
Evaluate	**ExtendList**	**FeatureInstall**
FileConverters	**FileDialog**	**FileFind**
FileSearch	**FindFile**	**FindFormat**
FixedDecimal	**FixedDecimalPlaces**	**GenerateGetPivotData**
GetCustomListContents	**GetCustomListNum**	**GetOpenFilename**
GetPhonetic	**GetSaveAsFilename**	**Goto**
Height	**Help**	**Hinstance**
Hwnd	**InchesToPoints**	**InputBox**
Interactive	**International**	**Intersect**

Iteration **LanguageSettings** **LargeButtons**
Left **LibraryPath** **MacroOptions**
MailLogoff **MailLogon** **MailSession**
MailSystem **MapPaperSize** **MaxChange**
MaxIterations **MoveAfterReturn** **MoveAfterReturnDirection**
Name **Names** **NetworkTemplatesPath**
NewWorkbook **NextLetter** **ODBCErrors**
ODBCTimeout **OLEDBErrors** **OnKey**
OnRepeat **OnTime** **OnUndo**
OnWindow **OperatingSystem** **OrganizationName**
Parent **Path** **PathSeparator**
PivotTableSelection **PreviousSelections** **ProductCode**
PromptForSummaryInfo **Quit** **Range**
Ready **RecentFiles** **RecordMacro**
RecordRelative **ReferenceStyle** **RegisteredFunctions**
RegisterXLL **Repeat** **ReplaceFormat**
RollZoom **Rows** **RTD**
Run **SaveWorkspace** **ScreenUpdating**
Selection **SendKeys** **SetDefaultChart**
Sheets **SheetsInNewWorkbook** **ShowChartTipNames**
ShowChartTipValues **ShowStartupDialog** **ShowToolTips**
ShowWindowsInTaskbar **SmartTagRecognizers** **Speech**
SpellingOptions **StandardFont** **StandardFontSize**
StartupPath **StatusBar** **TemplatesPath**
ThisCell **ThisWorkbook** **ThousandsSeparator**
Top TransitionMenuKey TransitionMenuKeyAction
TransitionNavigKeys **Undo** **Union**
UsableHeight **UsableWidth** **UsedObjects**
UserControl **UserLibraryPath** **UserName**
UseSystemSeparators Value **VBE**
Version **Visible** **Volatile**
Wait **Watches** **Width**
Windows **WindowsForPens** **WindowState**
Workbooks **WorksheetFunction** **Worksheets**

[Application.]ActivateMicrosoftApp(*XlMSApplication*)

Starts or activates another Microsoft Office application. `XlMSApplication` can be one of the following settings:

```
xlMicrosoftWord
xlMicrosoftPowerPoint
xlMicrosoftMail
xlMicrosoftAccess
xlMicrosoftFoxPro
xlMicrosoftProject
xlMicrosoftSchedulePlus
```

This method causes an error if the requested application is not installed. `xlMicrosoftMail` activates the user's default mail application.

[Application.]ActivePrinter [= *setting*]

Sets or returns the printer that Excel will use. When setting this property, the printer name must include the port number, for example:

```
Sub SetPrinter( )
    ActivePrinter = "\\wombat2\Lexmark Z52 Color Jetprinter on Ne04:"
End Sub
```

The preceding code tells Excel to use a shared printer over the network. The port number used by Excel is `Ne`*nn*`:` for virtual ports but is `LPT`*n*`:` or `COM`*n*`:` for physical ports. The following code gets an array of the available printers in a format that can be used by Excel:

```
Function GetPrinters() As String( )
    ' Use a suitably large array (supports up to 100 printers).
    ReDim result(100) As String
    Dim wshNetwork As Object, oPrinters As Object, temp As String
    ' Get the network object
    Set wshNetwork = CreateObject("WScript.Network")
    Set oPrinters = wshNetwork.EnumPrinterConnections
    ' Get the current active printer
    temp = ActivePrinter
    ' Printers collection has two elements for each printer.
    For i = 0 To oPrinters.Count - 1 Step 2
        ' Set the default printer.
        wshNetwork.SetDefaultPrinter oPrinters.Item(i + 1)
        ' Get what Excel sees.
        result(i \ 2) = ActivePrinter
        ' For debug purposes, show printer.
        Debug.Print ActivePrinter
    Next
    ' Trim empty elements off the array.
    ReDim Preserve result(i \ 2)
    ' Change back to original printer
    ActivePrinter = temp
    ' Return the result.
    GetPrinters = result
End Function
```

Application.AddChartAutoFormat(*Chart*, *Name*, [*Description*])

Creates a new chart type based on an existing chart.

Argument	Setting
Chart	A chart object to get formatting from
Name	The name to add to the chart autoformat list
Description	A description of the chart type

The following code adds a custom chart type to Excel based on an existing chart in the current workbook:

```
Sub TestAddChartType( )
    Application.AddChartAutoFormat Charts(1), _
      "new custom", "my description"
End Sub
```

To see the new chart type, select some data on a worksheet and choose Insert → Chart → Custom Types → User Defined.

Application.AddCustomList(*ListArray*, [*ByRow*])

Creates a new automatic list based on an array or a range of cells.

Argument	Setting
ListArray	The array or range of cells containing the items for the list.
ByRow	True creates the list from rows in a range; False creates the list from columns in the range. Ignored if *ListArray* is a single row or column. Causes an error if *ListArray* is not a range.

The first item in each list must be unique. An error occurs if a list with an identical first item already exists. The following code creates a new custom list from a range on the active worksheet:

```
Sub TestCustomList( )
    Application.AddCustomList [a1:a10]
End Sub
```

To see the new list, choose Tools → Options → Custom Lists.

Application.AlertBeforeOverwriting [= *setting*]

True displays an alert if a drag-and-drop changes cells that contain data; False does not. The default is True.

Application.AltStartupPath

Sets or returns the folder from which to automatically load templates and add-ins.

Application.ArbitraryXMLSupportAvailable

Returns True if Excel accepts custom XML schemas. This property is available only in Excel 2003.

Application.AskToUpdateLinks [= *setting*]

True asks prompts before updating external links when a workbook is opened; False does not prompt before updating. The default is True.

Application.Assistant

Returns a reference to the annoying Office Assistant character. For example, the following code displays the assistant and then animates its departure:

```
Sub TestAssistant( )
     Application.Assistant.Visible = True
     With Application.Assistant.NewBalloon
        .Text = "Ciao for now!"
        .Show
     End With
     Application.Assistant.Animation = msoAnimationGetArtsy
     Application.Assistant.Animation = msoAnimationGoodbye
End Sub
```

As of Office 2003, the assistant is no longer installed by default.

Application.AutoCorrect

Returns a reference to the `AutoCorrect` object. That object determines how Excel makes automatic corrections to user data entry.

Application.AutoFormatAsYouTypeReplaceHyperlinks [= *setting*]

True automatically reformats entries that begin with `http://`, `ftp://`, `mailto:`, and other protocols as hyperlinks; False does not. The default is True.

Application.AutomationSecurity [=*MsoAutomationSecurity*]

Set or returns the macro security setting used when opening Office documents in code. Possible settings are:

`msoAutomationSecurityLow`
: Enable all macros. This is the default.

`msoAutomationSecurityByUI`
: Use the security setting specified in the Security dialog box.

`msoAutomationSecurityForceDisable`
: Disable all without showing any security alerts.

These settings apply only to files opened in code. Files opened by the user apply the settings in the Security dialog box.

The default setting for this property is a security hole created to provide backward compatibility with multifile macros written for earlier versions of Excel. You should close this hole in your own code by setting the property to `msoAutomationSecurityByUI` before opening files, as shown here:

```
Sub TestMacroSecurity( )
    ' Enable macro security on file to open
    Application.AutomationSecurity = msoAutomationSecurityByUI
    With Application.FileDialog(msoFileDialogOpen)
```

```
            .AllowMultiSelect = False
            ' Get a file
            .Show
            ' Open it.
            Application.Workbooks.Open .SelectedItems(1)
        End With
    End Sub
```

Application.AutoPercentEntry [= *setting*]

True multiplies values formatted as percentage by 100 when displayed (e.g., entering 99 displays 9900%); False does not. Default is True.

Application.AutoRecover

Returns the `AutoRecover` object, which controls Excel's automatic file recovery features.

Application.Build

Returns the Excel build number. The following code displays Excel's version, build number, and calculation engine version:

```
Sub ShowVersion( )
    Debug.Print Application.Version; Application.Build; _
      Application.CalculationVersion
End Sub
```

[Application.]Calculate( )

Recalculates the formulas in all open workbooks.

Application.CalculateBeforeSave [= *setting*]

True recalculates workbooks before they are saved; False does not. Default is True.

Application.CalculateFull( )

Forces a full recalculation of all formulas in all workbooks.

Application.CalculateFullRebuild( )

Forces a full recalculation of all formulas and rebuilds dependencies in all workbooks.

Application.Calculation [= *XlCalculation*]

Sets or returns the calculation mode. Can be one of the following settings:

`xlCalculationAutomatic`
: Recalculates cells as data is entered (default)

`xlCalculationManual`
: Recalculates only when the user chooses Calculate Now (F9)

`xlCalculationSemiautomatic`
: Recalculates all cells except data tables automatically

Application.CalculationInterruptKey [= *XlCalculationInterruptKey*]

Sets or returns which key halts recalculation. Can be one of the following settings:

`xlAnyKey` (default)
`xlEscKey`
`xlNoKey`

Application.CalculationState

Sets or returns a constant indicating the state of all open workbooks. Can be one of the following:

`xlCalculating`
`xlDone`
`xlPending`

Application.CalculationVersion

Returns the version number of the calculation engine.

Application.Caller

Returns information about how the macro was called, as described in the following table:

When called from	Returns
A formula entered in a cell	A `Range` object for the cell
An array formula in a range of cells	A `Range` object for the range of cells
VBA code, the Run Macro dialog box, or anywhere else	`Error 2023`
An `Auto_Open`, `Auto_Close`, `Auto_Activate`, or `Auto_Deactivate` macro	The name of the workbook (Obsolete)
A macro set by the `OnDoubleClick` or `OnEntry` property	The name of the chart or cell to which the macro applies (Obsolete)

Application.Caption [= *setting*]

Sets or returns the text displayed in the Excel titlebar. For example, the following code replaces "Microsoft Excel" with "Funky Monkey" in the titlebar:

```
Sub TestCaption( )
    Application.Caption = "Funky Monkey"
End Sub
```

Application.CellDragAndDrop [= *setting*]

True enables drag-and-drop; False disables. Default is true.

[Application.]Cells[(*row*, *column*)]

Returns a range of cells on the active worksheet. For example, the following code selects cell B1 on the active worksheet:

```
Sub TestCells( )
     Cells(1, 2).Select
End Sub
```

Application.CentimetersToPoints(*Centimeters*)

Converts centimeters to points. This is the same as multiplying by 0.035.

[Application.]Charts([*index*])

Returns a reference to the `Charts` collection.

Application.CheckAbort([*KeepAbort*])

Aborts recalculation. The argument *`KeepAbort`* accepts a `Range` object to continue recalculating. This lets you stop recalculation for all but a specific range of cells.

Application.CheckSpelling(*Word*, [*CustomDictionary*], [*IgnoreUppercase*])

Returns True if *`Word`* is spelled correctly; False if it is not.

Argument	Setting
`Word`	The word to spellcheck.
`CustomDictionary`	The filename of the custom dictionary to use if the word isn't found in the main dictionary. Defaults to the user setting.
`IgnoreUppercase`	True excludes words that are all uppercase; False includes them. Defaults to the user setting.

Application.ClipboardFormats

Returns an array of `XlClipboardFormat` constants indicating the types of data currently on the clipboard. Possible array values are:

`xlClipboardFormatBIFF`	`xlClipboardFormatBIFF2`
`xlClipboardFormatBIFF3`	`xlClipboardFormatBIFF4`
`xlClipboardFormatBinary`	`xlClipboardFormatBitmap`
`xlClipboardFormatCGM`	`xlClipboardFormatCSV`
`xlClipboardFormatDIF`	`xlClipboardFormatDspText`
`xlClipboardFormatEmbeddedObject`	`xlClipboardFormatEmbedSource`
`xlClipboardFormatLink`	`xlClipboardFormatLinkSource`
`xlClipboardFormatLinkSourceDesc`	`xlClipboardFormatMovie`
`xlClipboardFormatNative`	`xlClipboardFormatObjectDesc`
`xlClipboardFormatObjectLink`	`xlClipboardFormatOwnerLink`
`xlClipboardFormatPICT`	`xlClipboardFormatPrintPICT`
`xlClipboardFormatRTF`	`xlClipboardFormatScreenPICT`
`xlClipboardFormatStandardFont`	`xlClipboardFormatStandardScale`
`xlClipboardFormatSYLK`	`xlClipboardFormatTable`
`xlClipboardFormatText`	`xlClipboardFormatToolFace`
`xlClipboardFormatToolFacePICT`	`xlClipboardFormatVALU`
`xlClipboardFormatWK1`	

Use `ClipboardFormats` to determine the type of data available on the clipboard before taking other actions, such as Paste. For example, this code copies a chart into the clipboard, then pastes it into Paint:

```
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Sub TestClipBoardFormats( )
    Dim fmt, chrt As Chart
    ' Copy a chart image into the clipboard.
    Set chrt = Charts(1)
    chrt.CopyPicture xlScreen, xlBitmap
    For Each fmt In Application.ClipboardFormats
        ' If the bitmap is in the clipboard
        If fmt = xlClipboardFormatBitmap Then
        ' Start Paint
        Shell "mspaint.exe", vbNormalFocus
        ' Wait a half second to catch up.
        Sleep 500
        ' and paste the Chart image.
        SendKeys "%EP", True
        Exit For
        End If
    Next
End SubEnd Sub
```

The Sleep API shown in the preceding code is required to wait for focus to change to the newly opened Paint application.

[Application].Columns([*index*])

Returns one or more columns on the active worksheet as a `Range` object. For example, the following code selects column C on the active worksheet:

```
Sub TestColumns( )
     Columns(3).Select
End Sub
```

Application.COMAddIns([*index*])

Returns a collection of the installed COM add-ins. If there are no COM add-ins installed, causes an error. The following code lists the COM add-ins:

```
Sub TestCOMAddins( )
    Dim c  As COMAddIn
    On Error Resume Next
    For Each c In Application.COMAddIns
        If Err Then Debug.Print "No COM addins."
        Debug.Print Join(Array(c.Description, c.progID, c.Application, _
          c.Connect), ", ")
    Next
End Sub
```

Application.CommandBars([*index*])

Returns one or more command bars. The following code displays a list of the command bars with their status:

```
Sub TestCommandbars( )
    Dim cb As CommandBar
    Debug.Print "Name", "Visible?", "BuiltIn?"
    For Each cb In Application.CommandBars
        Debug.Print cb.Name, cb.Visible, cb.BuiltIn
    Next
End Sub
```

Application.CommandUnderlines [= *xlCommandUnderlines*]

(Macintosh only.) Sets or returns how commands are highlighted. Can be one of the following settings:

`xlCommandUnderlinesOn`

`xlCommandUnderlinesOff`

`xlCommandUnderlinesAutomatic`

For Windows, `CommandUnderlines` always returns `xlCommandUnderlinesOn` and cannot be set.

Application.ConstrainNumeric [= *setting*]

(Windows for Pen only.) True restricts handwriting recognition to numbers and punctuation; False allows the full alphabet.

Application.ControlCharacters [= *setting*]

(Right-to-left language display only.) True displays control characters for right-to-left languages; False hides the characters.

Application.ConvertFormula(Formula, FromReferenceStyle, [ToReferenceStyle], [ToAbsolute], [RelativeTo])

Converts cell references in a formula between the A1 and R1C1 reference styles, between relative and absolute references, or both.

Argument	Description	Settings
Formula	The formula you want to convert.	Must be a valid formula beginning with an equals sign
FromReferenceStyle	The `XlReferenceStyle` of the formula.	`xlA1` `xlR1C1`
ToReferenceStyle	The `XlReferenceStyle` style you want returned. If this argument is omitted, the reference style isn't changed; the formula stays in the style specified by *FromReferenceStyle*.	`xlA1` `xlR1C1`

ToAbsolute	The converted `XlReferenceStyle`. If omitted, the reference type isn't changed. Defaults to `xlRelative`.	`xlAbsolute`
		`xlAbsRowRelColumn`
		`xlRelRowAbsColumn`
		`xlRelative`

RelativeTo	The cell that references are relative to. Defaults to active cell.	`Range` object

The following code converts a formula to R1C1 style relative to cell A1:

```
Sub TestConvertFormula( )
    Dim str As String
    str = "=Sum(A1:A20)"
    Debug.Print Application.ConvertFormula(str, xlA1, xlR1C1, _
      xlRelative, [a1])
End Sub
```

Application.CopyObjectsWithCells [= *setting*]

True copies objects, such as buttons, with selected cells; False omits objects. Default is True.

Application.Cursor [= *XlMousePointer*]

Sets or returns the mouse pointer image. Can be one of these settings:

`xlDefault`
`xlIBeam`
`xlNorthwestArrow`
`xlWait`

Application.CursorMovement [= *setting*]

Sets or returns whether a visual cursor or a logical cursor is used. Can be one of these settings:

`xlVisualCursor`
`xlLogicalCursor`

Application.CustomListCount

Returns the number of custom lists. To view custom lists, select Tools → Options → Custom Lists.

Application.CutCopyMode [= *setting*]

Sets or returns whether or not the user is currently cutting or copying cells. Return settings are:

False, Excel is not in either mode
`xlCopy`
`xlCut`

Setting `CutCopyMode` to True or False cancels the current mode.

Application.DataEntryMode [= *setting*]

Sets or returns whether or not Excel is in data-entry mode. Can be one of these settings:

`xlOn`
`xlOff`
`xlStrict`, prevents the user from exiting the mode by pressing Esc

Data-entry mode restricts users to unlocked cells. By default, cell protection is set to Locked, so you must unlock a range to demonstrate this feature. The following code

restricts data entry to range A1:D4; the user can return to regular mode by pressing Esc, as shown by the following code:

```
Sub TestDataEntryMode( )
    Range("a1:d4").Locked = False
    Application.DataEntryMode = xlOn
End Sub
```

Application.DecimalSeparator [= *setting*]

Sets or returns the character used as the decimal separator.

Application.DefaultFilePath [= *setting*]

Sets or returns the path Excel uses by default when opening files.

Application.DefaultSaveFormat [= *XlFileFormat*]

Sets or returns the file format used by Excel when saving. Can be one of these settings:

xlAddIn	xlCSV	xlCSVMac
xlCSVMSDOS	xlCSVWindows	xlCurrentPlatformText
xlDBF2	xlDBF3	xlDBF4
xlDIF	xlExcel2	xlExcel2FarEast
xlExcel3	xlExcel4	xlExcel4Workbook
xlExcel5	xlExcel7	xlExcel9795
xlHtml	xlIntlAddIn	xlIntlMacro
xlSYLK	xlTemplate	xlTextMac
xlTextMSDOS	xlTextPrinter	xlTextWindows
xlUnicodeText	xlWebArchive	xlWJ2WD1
xlWJ3	xlWJ3FJ3	xlWK1
xlWK1ALL	xlWK1FMT	xlWK3
xlWK3FM3	xlWK4	xlWKS
xlWorkbookNormal	xlWorks2FarEast	xlWQ1
xlXMLSpreadsheet		

Application.DefaultSheetDirection [= *setting*]

Sets or returns the default reading direction. Can be one of these settings:

xlRTL
xlLTR

Application.DefaultWebOptions

Returns a DefaultWebOptions object that determines how Excel saves workbooks as web pages.

Application.DeleteChartAutoFormat(*Name*)

Removes a custom chart type. The following code removes the custom chart type created earlier in AddChartAutoFormat:

```
Sub TestDeleteChartType( )
    Application.DeleteChartAutoFormat "new custom"
End Sub
```

[Application.]DeleteCustomList(ListNum)

Removes a custom list. The following code removes the list created earlier in AddCustomList:

```
Sub TestDeleteCustomList( )
    ' Delete the last list.
    Application.DeleteCustomList Application.CustomListCount
End Sub
```

Application.Dialogs(*XlBuiltInDialog*)

Returns the collection of Excel's dialog boxes. Use Dialogs to display any of the Excel dialog boxes from code. The following code displays the Activate Workbook dialog box:

```
Sub TestDialogs( )
    Application.Dialogs(XlBuiltInDialog.xlDialogActivate).Show
End Sub
```

Excel has hundreds of dialog boxes. See Appendix A for a list of them.

Application.DisplayAlerts [= *setting*]

True displays standard Excel dialogs while a macro runs; False hides those dialogs and automatically uses the default response for each. Default is True.

Set this property to False for batch operations in which you don't want user intervention; be sure to reset the property to True when done. For example, the following code closes all workbooks but the current one without saving or prompting the user:

```
Sub CloseAllNoSave( )
    Dim wb As Workbook
    ' Turn off warnings.
    Application.DisplayAlerts = False
    For Each wb In Workbooks
        ' Close all workbooks but this one.
        If Not (wb Is ThisWorkbook) Then _
          wb.Close
```

```
    Next
    ' Turn warnings back on.
    Application.DisplayAlerts = True
End Sub
```

Application.DisplayClipboardWindow [= *setting*]

True displays the Clipboard window; False hides it. For example, the following code copies a chart and displays the Clipboard window:

```
Sub TestClipBoardWindow( )
    Dim chrt As Chart
    ' Copy a Chart image into the Clipboard.
    Set chrt = Charts(1)
    chrt.CopyPicture xlScreen, xlBitmap
    Application.DisplayClipboardWindow = True
End Sub
```

Application.DisplayCommentIndicator [=*XlCommentDisplayMode*]

Sets or returns the icon displayed for comments. Can be one of the following settings:

`xlNoIndicator`

`xlCommentIndicatorOnly` (default)

`xlCommentAndIndicator`

Application.DisplayDocumentActionTaskPane [= *setting*]

For Smart documents, True displays the Document Action task pane, and False hides it. Setting this property causes an error if the workbook is not a Smart document.

Application.DisplayExcel4Menus [= *setting*]

True uses Excel Version 4.0 menus; False uses the current version menus. Default is False.

Application.DisplayFormulaBar [= *setting*]

True displays the Formula bar; False hides it. Default is True.

Application.DisplayFullScreen [= *setting*]

True displays Excel in full-screen mode; False uses the standard window mode. Default is False.

Application.DisplayFunctionToolTips [= *setting*]

True displays the function tool tips; False does not. Default is True.

Application.DisplayInsertOptions [= *setting*]

True displays a dialog with special options, such as Clear Formatting, when inserting cells; False does not display the dialog. Default is True.

Application.DisplayNoteIndicator [= *setting*]

True displays an icon indicating cells with notes; False hides the icon. Default is True.

Application.DisplayPasteOptions [= *setting*]

True displays a dialog with special options when pasting cells; False does not display the dialog. Default is True.

Application.DisplayRecentFiles [= *setting*]

True displays a list of recently opened files on the File menu; False does not. Default is True.

Application.DisplayScrollBars [= *setting*]

True displays scrollbars for workbooks; False does not. Default is True.

Application.DisplayStatusBar [= *setting*]

True displays application status bar; False does not. Default is True.

Application.DisplayXMLSourcePane([*XmlMap*])

(Excel 2003 Professional Edition only.) Displays the XML Source task pane.

Argument	Setting
XmlMap	The `XmlMap` object to display in the task pane

Application.DoubleClick( )

Double-clicks the active cell. This method emulates the user action.

Application.EditDirectlyInCell [= *setting*]

True allows editing in cells; False requires edits to be made in the Formula bar. Default is True.

Application.EnableAnimations [= *setting*]

True animates insertions and deletions; False does not animate those operations. Default is True.

[Application.]EnableAutoComplete [= *setting*]

True automatically completes words; False does not. Default is True.

Application.EnableCancelKey [= *XlEnableCancelKey*]

Sets or returns how Excel handles the Esc, Ctrl-Break, and Command-Period (Macintosh) keys. Can be one of these settings:

`xlDisabled`
: Cancel key trapping disabled.

`xlErrorHandler`
: Cancel key causes error 18, which can be trapped by an On Error statement.

`xlInterrupt`
: Cancel interrupts the current procedure, and the user can debug or end it (default).

Application.EnableEvents [= *setting*]

True turns on Excel events; False turns off Excel events. Default is True. Setting this property to False prevents code written for `Workbook`, `Worksheet`, and other object events from running.

Application.EnableSound [= *setting*]

True allows Excel to play sounds; False disables sounds. Default is True.

Application.ErrorCheckingOptions

Returns the `ErrorCheckingOptions` object, which controls Excel's settings for automatic error checking.

[Application.]Evaluate(*Name*)

Evaluates an expression and returns the result. `Evaluate` is equivalent to enclosing the expression in square brackets ([ ]).

Argument	Setting
Name	A range address, a named range, or a formula

It is common to use the bracket notation for the `Evaluate` method since it is shorter. The following code displays various values from the active sheet:

```
Sub TestEvaluate( )
    ' Show value of cell A1.
    Debug.Print [a1]
    ' Show total of A1:A3.
    Debug.Print [sum(a1:a3)]
    ' Show table of named ranges
    Dim n As Name, str As String
    Debug.Print "Name", "# w/data", "Address"
    For Each n In Names
        str = "Count(" & n.Name & ")"
        Debug.Print n.Name, Evaluate(str), [n]
    Next
End Sub
```

Using the bracket notation with a `Name` object returns the address of the name.

Application.ExtendList [= *setting*]

True extends formatting and formulas to new data added to a custom list; False does not. Default is True.

Application.FeatureInstall [= *MsoFeatureInstall*]

Determines how to handle calls to methods and properties that require features that aren't yet installed. Can be one of these settings:

`msoFeatureInstallNone`
: Doesn't install; causes an error when uninstalled features is called (default)

`msoFeatureInstallOnDemand`
: Prompts the user to install feature

`msoFeatureInstallOnDemandWithU`
: Automatically installs the feature; doesn't prompt the user

Application.FileConverters[(*Index1*, *Index2*)]

Returns an array of installed file converters.

Argument	Setting
Index1	The full name of the converter including file type
Index2	The path of the converter's DLL

If arguments are omitted, `FileConverters` returns `Null` if there are no converters or a two-dimensional array containing the name, DLL path, and extension for each converter. The following code displays a table of the installed converters:

```
Sub TestFileConverters( )
    Dim cnv As Variant, i As Integer
    cnv = Application.FileConverters
    ' Display table columns
    Debug.Print "Name", "DLL", "Extension"
    ' Check if converters are installed
    If Not IsNull(cnv) Then
        For i = 1 To UBound(cnv, 1)
            Debug.Print cnv(i, 1), cnv(i, 2), cnv(i, 3)
        Next
    Else
        Debug.Print "No converters installed."
    End If
End Sub
```

Application.FileDialog (*MsoFileDialogType*)

Returns the `FileDialog` object.

Argument	Description	Settings

MsoFileDialogType	Determines which Excel dialog to return	`msoFileDialogFilePicker`
		`msoFileDialogFolderPicker`
		`msoFileDialogOpen`
		`msoFileDialogSaveAs`

The following code displays the file picker dialog box and lets the user select a text file to open in Notepad:

```
Sub TestFileDialog( )
    Dim fname As String
    With Application.FileDialog(msoFileDialogFilePicker)
        .AllowMultiSelect = False
        .Filters.Add "Text files (*.txt)", "*.txt", 1
        .FilterIndex = 1
        .Title = "Open text file"
        If .Show = True Then _
          Shell "notepad.exe " & .SelectedItems(1)
    End With
End Sub
```

Application.FileFind

(Macintosh only.) Returns the `FileFind` object. The following code displays all of the files by Jeff:

```
Sub TestFind( ) ' Macintosh only
    Dim s
    With Application.FileFind
        .Author = "Jeff"
        .Execute
        For Each s In .Results
            Debug.Print s
        Next
    End With
End Sub
```

Application.FileSearch

(Windows only.) Returns the `FileSearch` object. The following code displays all of the text files in the current folder:

```
Sub TestSearch( ) ' Windows only
    Dim s
    With Application.FileSearch
        .LookIn = ThisWorkbook.Path
        .Filename = ".txt"
        .Execute
        For Each s In .FoundFiles
            Debug.Print s
        Next
    End With
End Sub
```

Application.FindFile( )

Displays the Open File dialog box and opens the selected file in Excel.

Application.FindFormat

Returns the `CellFormat` object used by the `Find` method. For example, the following code selects the first bold cell on the active worksheet:

```
Sub TestFindFormat( )
    With Application.FindFormat
        .Font.Bold = True
    End With
    Cells.Find("", , , , , , , , True).Select
End Sub
```

Application.FixedDecimal [= *setting*]

True assumes a fixed decimal place for data entries; False assumes each entry has a variable decimal place. Default is False.

Application.FixedDecimalPlaces [= *setting*]

Sets the placement of the decimal assumed during data entry. Default is 2. The following code configures Excel to treat the entry `1000` as `0.1`, `45000` as `4.5`, and so on:

```
Sub TestDecimal( )
    ' Turn on fixed decimal.
    Application.FixedDecimal = True
    ' Set the decimal place.
    Application.FixedDecimalPlaces = 4
End Sub
```

Application.GenerateGetPivotData [= *setting*]

True turns the `GenerateGetPivotData` command on; False turns the command off. The `GenerateGetPivotData` command substitutes cell references for GETPIVOTDATA worksheet functions in formulas.

Application.GetCustomListContents

Returns an array of items from a custom list. For example, the following code displays all of the items in each of the custom lists:

```
Sub TestListContent( )
    Dim i As Integer, lst( ), str As String, num As Integer
    Debug.Print "List Number", "Contents"
    For i = 1 To Application.CustomListCount
        lst = Application.GetCustomListContents(i)
        str = Join(lst, ", ")
        num = Application.GetCustomListNum(lst)
        Debug.Print num, str
    Next
End Sub
```

Application.GetCustomListNum(*ListArray*)

Returns the index of a custom list.

Argument	Setting
ListArray	The array of custom list items to look up.

Application.GetOpenFilename([*FileFilter*], [*FilterIndex*], [*Title*], [*ButtonText*], [*MultiSelect*])

Displays the Open File dialog box and returns a filename or False if no file is selected. Does not open the file.

Argument	Setting
FileFilter	A filter to use in the drop-down list on the dialog box. Each filter is a pair separated by a comma: *DisplayString*, *Type*. See the following example.
FilterIndex	The index of the filter to display initially.
Title	The caption for the dialog box. Default is Open.
ButtonText	(Macintosh only.) The caption to show on the action button. Default is Open.
MultiSelect	True allows the user to select multiple files.

The following code displays the File Open dialog box for web file types; if the user selects a file, the code opens the file in Notepad:

```
Sub TestGetOpen( )
    Dim fname As String, fltr As String
    fltr = "Web page (*.htm),*.htm,XML data (*.xml),*.xml," & _
      "XML Style Sheet (*.xsl),*.xsl"
    fname = Application.GetOpenFilename(fltr, _
      1, "Open web file", , False)
    If fname <> "False" Then _
        Shell "Notepad.exe " & fname
End Sub
```

Application.GetPhonetic([*Text*])

Returns the Japanese phonetic text of a string. Available only with Japanese language support.

Application.GetSaveAsFilename([*InitialFilename*], [*FileFilter*], [*FilterIndex*], [*Title*], [*ButtonText*])

Displays the Save File As dialog box and returns a filename or False if no file is selected. Does not save the file.

Argument	Setting
InitialFileName	The name to display in the File text box
Other arguments	See "Application.GetOpenFilename"

The following code saves the active workbook as a web page, closes the newly saved file, and reopens the original workbook in XLS format:

```
Sub TestGetSaveAs( )
    Dim fname1 As String, fname2 As String, fname3 As String
```

```
    Dim fltr As String
    ' Save changes
    ActiveWorkbook.Save
    ' Get current filename.
    fname1 = ActiveWorkbook.Name
    ' Get filename for web page.
    fname2 = Replace(fname1, "xls", "htm")
    fltr = "Web page (*.htm),*.htm,XML data (*.xml),*.xml," & _
      "XML Style Sheet (*.xsl),*.xsl"
    ' Show the Save As dialog.
    fname3 = Application.GetSaveAsFilename(fname2, fltr, _
      1, "Export to web")
    ' If not cancelled, save the file as a web page.
    If fname3 <> "False" Then _
        ActiveWorkbook.SaveAs fname3, xlHtml
    ' Reopen the original file.
    Workbooks.Open fname1
    ' Close the web page file.
    Workbooks(fname2).Close
End Sub
```

Application.Goto([*Reference*], [*Scroll*])

Selects a range of cells and activates the sheet containing the cells.

Argument	Setting
Reference	A range, named range, or string that evaluates to one of those.
Scroll	True scrolls the sheet so that the selection is in the upper-left corner.

`Goto` is similar to `Select`, except `Select` does not activate the sheet.

Application.Height

Returns the height of the Excel window in pixels. Use the `WindowState` property to maximize window or minimize Excel.

Application.Help([*HelpFile*], [*HelpContextID*])

Displays a help topic in Excel's Help window.

Argument	Setting
HelpFile	The file to display. Can be compiled Help (*.chm* or *.hlp*) or a web page (*.htm*). Defaults to the Excel help file.
HelpContextID	For compiled help files, the numeric ID of the topic to display. Ignored for web pages.

See Chapter 6 for details on creating and displaying Help. The following code displays an error message help page in the Help window:

```
Sub TestApplicationHelp( )
    ' Display Help in Help window.
    Application.Help ("http://excelworkshop.com/Help/error51.htm")
End Sub
```

Application.Hinstance

Returns a handle to the Excel application instance.

Application.Hwnd

Returns a handle to the top-level Excel window. You use handles with the Windows API to do low-level tasks not available through Excel objects. For example, the following code displays the Excel always on top of all other windows, even if Excel doesn't have focus:

```
Declare Function SetWindowPos Lib "user32" (ByVal hwnd As Long, _
  ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, _
  ByVal cx As Long, ByVal cy As Long, ByVal wFlags As Long) As Long
Const SWP_NOSIZE = &H1
Const SWP_NOMOVE = &H2
Const HWND_TOPMOST = -1
Const HWND_NOTOPMOST = -2

Sub TestShowXLOnTop( )
    ' Change to False to return to normal.
    ShowXLOnTop True
End Sub

Public Function ShowXLOnTop(ontop As Boolean)
    Dim hXl As Long, setting As Long
    If ontop Then setting = HWND_TOPMOST _
      Else setting = HWND_NOTOPMOST
    hXl = Application.hwnd
    SetWindowPos hXl, setting, 0, 0, _
      0, 0, SWP_NOSIZE Or SWP_NOMOVE
End Sub
```

Application.InchesToPoints(*Inches*)

Converts a measurement from inches to points. This is the same a multiplying the value by 72.

[Application.]InputBox(*Prompt*, [*Title*], [*Default*], [*Left*], [*Top*], [*HelpFile*], [*HelpContextID*], [*Type*])

This is the same as the Visual Basic `InputBox` method with one addition: `Application.InputBox` allows you to get a selected range using the *Type* argument which accepts the settings in the following table:

Setting	Input is
0	A formula
1	A number
2	Text (a string)
4	A logical value (True or False)
8	A cell reference, as a `Range` object
16	An error value, such as #N/A
64	An array of values

The following code demonstrates getting a range using `InputBox`:

```
Sub TestInputBox( )
    Dim rng As Range
    On Error Resume Next
    Set rng = Application.InputBox( _
      "Select a cell", , , , , , 8)
    If Not (rng Is Nothing) Then
        Debug.Print rng.Count & " cells selected."
    Else
        Debug.Print "Input cancelled."
    End If
End Sub
```

See Chapter 3 for details on the Visual Basic `InputBox` method.

Application.Interactive [= *setting*]

True allows users to interact with Excel; False prevents user actions. Set the `Interactive` property to False to prevent user actions while performing time-consuming operations in code. Be sure to set `Interactive` back to True when done.

Application.International(*XlApplicationInternational*)

Returns an array of locale settings. `XlApplicationInternational` can be one of the settings from the following table:

Category	Setting	Returns
Cell references	`xlLeftBrace`	Character used instead of the left brace ({) in array literals.
	`xlLeftBracket`	Character used instead of the left bracket ([) in R1C1-style relative references.
	`xlLowerCaseColumnLetter`	Lowercase column letter.
	`xlLowerCaseRowLetter`	Lowercase row letter.
	`xlRightBrace`	Character used instead of the right brace (}) in array literals.
	`xlRightBracket`	Character used instead of the right bracket (]) in R1C1-style references.

Category	Setting	Returns
	xlUpperCaseColumnLetter	Uppercase column letter.
	xlUpperCaseRowLetter	Uppercase row letter (for R1C1-style references).
Country/Region	xlCountryCode	Excel country/region version setting.
	xlCountrySetting	Windows country/region setting.
	xlGeneralFormatName	Name of the General number format.
Currency	xlCurrencyBefore	True if the currency symbol precedes the currency values; False if it follows them.
	xlCurrencyCode	Currency symbol.
	xlCurrencyDigits	Number of decimal digits to be used in currency formats.
	xlCurrencyLeadingZeros	True if leading zeros are displayed for zero currency values.
	xlCurrencyMinusSign	True if a minus sign indicates negative numbers; False if using parentheses.
	xlCurrencyNegative	Currency format for negative currency values: • 0, parentheses, ($nnn) or (nnn$) • 1, minus before, -$nnn or –nnn$ • 2, minus mid, $-nnn or nnn-$ • 3, minus after, $nnn- or nnn$-
	xlCurrencySpaceBefore	True adds a space before the currency symbol.
	xlCurrencyTrailingZeros	True displays trailing zeros for zero currency values.
	xlNoncurrencyDigits	Number of decimal digits to be used in noncurrency formats.
Date and Time	xl24HourClock	True uses 24-hour time; False uses 12-hour time.
	xl4DigitYears	True uses four-digit years; False uses two-digit years.
	xlDateOrder	Order of date elements: • 0, month-day-year • 1, day-month-year • 2, year-month-day
	xlDateSeparator	Date separator (/).
	xlDayCode	Day symbol (d).
	xlDayLeadingZero	True includes leading zero in days.
	xlHourCode	Hour symbol (h).
	xlMDY	True orders dates month-day-year in the long form; False orders dates day-month-year.
	xlMinuteCode	Minute symbol (m).
	xlMonthCode	Month symbol (m).
	xlMonthLeadingZero	True includes leading zero in months displayed as numbers.
	xlMonthNameChars	Obsolete, always returns 3.
	xlSecondCode	Second symbol (s).
	xlTimeLeadingZero	True includes leading zero in times.

Category	Setting	Returns
	xlTimeSeparator	Time separator (:)
	xlWeekdayNameChars	Obsolete, always returns 3.
	xlYearCode	Year symbol in number formats (y).
Measurement	xlMetric	True is metric system in use; False if the English measurement system is in use.
	xlNonEnglishFunctions	True if functions are not displayed in English.
Separators	xlAlternateArraySeparator	Alternate array item separator to be used if the current array separator is the same as the decimal separator.
	xlColumnSeparator	Character used to separate columns in array literals.
	xlDecimalSeparator	Decimal separator.
	xlListSeparator	List separator.
	xlRowSeparator	Character used to separate rows in array literals.
	xlThousandsSeparator	Zero or thousands separator.

[Application.]Intersect(*Arg1*, *Arg2*, [*Argn*], ...)

Returns the Range object containing the overlapping region of the ranges *Arg1* through *Argn*.

Argument	Setting
Arg1	The first Range object to intersect
Arg2	The second Range object to intersect
Argn	Any number of additional Range objects to intersect

Application.Iteration [= *setting*]

True uses iteration to calculate formulas that refer to themselves (this is called a *circular reference*); False causes an error for circular references. Default is False. Use the MaxChange and MaxIterations properties to control how many calculations are performed during iteration.

Application.LanguageSettings

Returns a LanguageSettings object containing information about the user's locale.

Application.LargeButtons [= *setting*]

True displays large toolbar buttons; False displays regular-size buttons. Default is False.

Application.Left [= *setting*]

Sets or returns the distance between the left edge of the screen and the left edge of the Excel window in pixels.

Application.LibraryPath

Returns the path to the Excel add-in library, for example *C:\Program Files\Microsoft Office\OFFICE11\LIBRARY*.

Application.MacroOptions([Macro], [Description], [HasMenu], [MenuText], [HasShortcutKey], [ShortcutKey], [Category], [StatusBar], [HelpContextId], [HelpFile])

Sets the description and help files displayed for a macro or user-defined function.

Argument	Setting
Macro	The name of the macro to set.
Description	A description that appears in the Macro or Formula dialog box.
HasMenu	Ignored.
MenuText	Ignored.
HasShortcutKey	True assigns a shortcut key to the macro.
ShortcutKey	The shortcut key to assign.
Category	The name of a category for the user-defined function. Default is User Defined.
StatusBar	Ignored.
HelpContextId	The context ID for the help topic within the compiled help file. Ignored for other help file types.
HelpFile	The name of the help file to display for user-defined functions.

The usable arguments are different for macros (`Subs`) and user-defined functions (`Functions`). The Macro dialog box doesn't use *`Category`*, *`HelpContextId`*, or *`HelpFile`* arguments. The Insert Function dialog box doesn't use *`HasShortcutKey`* or *`ShortcutKey`* arguments.

The following code sets the options for the `ShowXlOnTop` user-defined function:

```
Sub TestMacroOptions( )
    Application.MacroOptions "ShowXlOnTop", _
      "Set Excel as the top-most window.", , , , , _
      "Windows", "Excel On Top", , _
      "http:\\excelworkshop.com\Help\ch07.htm"
End Sub
```

After this code runs, Excel displays the options on the Insert Function dialog as shown in Figure 7-3.

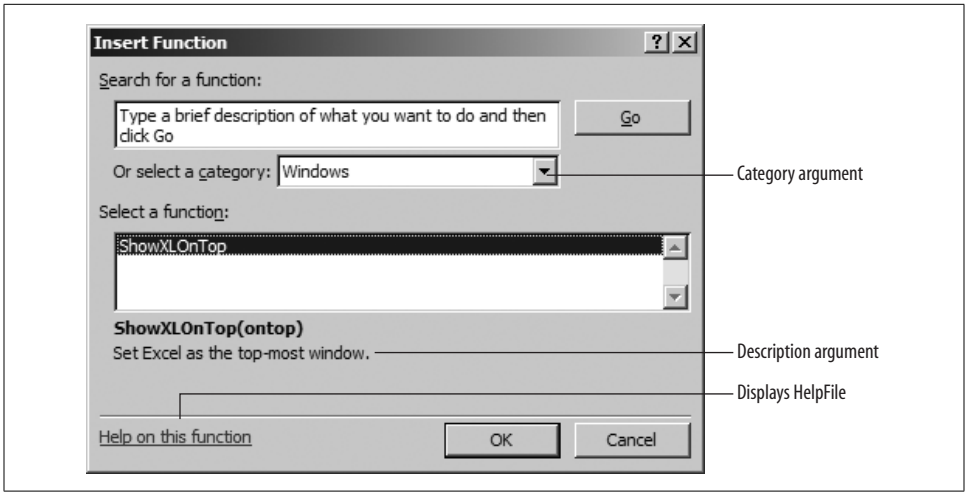

Figure 7-3. How Excel displays macro options for user-defined functions

Application.MailLogoff( )

Ends a MAPI mail session.

Application.MailLogon([*Name*], [*Password*], [*DownloadNewMail*])

Closes any existing MAPI sessions and creates a new one, starting the Microsoft Mail spooler. Returns True if Mail is started successfully, False if not.

Argument	Setting
Name	The username for the mail session.
Password	User password.
DownloadNewMail	True downloads new mail immediately. Default is False.

Application.MailSession

Returns the MAPI session number begun by Excel. Returns `Null` if there is no session.

Application.MailSystem

Returns the `XlMailSystem` setting indicating the users installed mail system. Can be one of these settings:

```
xlMAPI
xlNoMailSystem
xlPowerTalk
```

Application.MapPaperSize [= *setting*]

True adjusts printing to map from the standard paper size of one locale to another; False does not adjust.

Application.MaxChange [= *setting*]

The maximum amount of change allowed in resolving circular references using iteration. Once the change is less than this amount, iteration stops.

Application.MaxIterations [= *setting*]

The maximum number of calculations performed when resolving a circular reference.

Application.MoveAfterReturn [= *setting*]

True activates the next cell after the user presses Enter; False keeps the current cell active. Default is True.

Application.MoveAfterReturnDirection [=*XlDirection*]

Sets or returns which cell is activated after the user presses Enter. Can be one of these settings:

`xlDown` (default)
`xlToLeft`
`xlToRight`
`xlUp`

Application.Names([*index*])

Returns the collection of named ranges in the active workbook. The following code displays a table of named ranges:

```
Sub TestNames( )
    Dim n As Name
    Debug.Print "Name", "Address"
    For Each n In Names
        Debug.Print n.Name, n.RefersTo
    Next
End Sub
```

Application.NetworkTemplatesPath

Returns the `AltStartupPath` property if that setting is a network share. Otherwise, returns an empty string.

Application.NewWorkbook

Returns an Office NewFile object that represents an item on the New Workbook task pane. You can use this object to add or remove items from the task pane. For example, the following code adds the Invoice template and displays the task pane:

```
Sub TestNewWorkbook( )
    With Application.NewWorkbook
      .Add "Invoice.xlt", _
        MsoFileNewSection.msoNewfromTemplate, _
        "New Invoice", MsoFileNewAction.msoCreateNewFile
    End With
    Application.CommandBars("Task Pane").Visible = True
End Sub
```

See the Office VBA help file (*VBAOF11.CHM*) for information about the NewFile object.

Application.NextLetter( )

(Macintosh with PowerTalk mail only.) Opens the next unread mail message in the In Tray.

Application.ODBCErrors

Returns the ODBCErrors collection generated by the most recent query table or PivotTable report.

Application.ODBCTimeout [= *setting*]

Sets or returns the time limit for ODBC queries. Default is 45 seconds.

Application.OLEDBErrors

Returns the OLEDBErrors collection generated by the most recent OLE DB query.

Application.OnKey(*Key*, [*Procedure*])

Assigns a macro to run when a key is pressed. Can also be used to disable built-in Excel key combinations.

Argument	Setting
Key	The key combination to assign. The character codes are the same as for SendKeys. See Chapter 3 for the SendKeys codes.
Procedure	The name of the macro to run. Setting to "" disables any built-in action for those keys; omitting this argument restores the built-in action.

The following code demonstrates how to reassign, disable, and restore a built-in key assignment:

```
Sub TestOnKey( )
    ' Reassign Ctrl+C
   Application.OnKey "^c", "CopyMsg"
   ' Disable Ctrl+C
   'Application.OnKey "^c", ""
   ' Restore Ctrl+C
   ' Application.OnKey "^c"
End Sub

Sub CopyMsg( )
    MsgBox "You can't copy right now."
End Sub
```

Application.OnRepeat(*Text*, *Procedure*)

Reassigns the Repeat item on the Edit menu (Ctrl-Y).

Argument	Setting
Text	The text to display instead of Repeat…
Procedure	The procedure to run when the user chooses Edit → Repeat or presses Ctrl-Y.

The following code replaces the Repeat item on the Edit menu with the item Do Over and runs the DoOver procedure with the user selects the item:

```
Sub TestOnRepeat( )
    Application.OnRepeat "Do over", "DoOver"
End Sub
```

Application.OnTime(*EarliestTime*, *Procedure*, [*LatestTime*], [*Schedule*])

Sets the name of a procedure to run at a specified time.

Argument	Setting
EarliestTime	The earliest time you want to run the procedure.
Procedure	The name of the procedure to run.
LatestTime	The latest time you want to run the procedure. Default is no limit.
Schedule	True schedules the procedure to run; False removes the procedure from the schedule to run. Default is True.

Application.OnUndo(*Text*, *Procedure*)

Reassigns the Undo item on the Edit menu (Ctrl-Z). The arguments are the same as for OnRepeat.

Application.OnWindow [= *setting*]

Sets or returns a procedure to run when a window is activated.

Application.OperatingSystem

Returns the name, version, and address model of the operating system. For example, "Windows (32-bit) NT 5.01" indicates Windows XP Professional.

Application.OrganizationName

Returns the name of the user's organization as entered during installation.

Application.Path

Returns the path to the folder where Excel is installed.

Application.PathSeparator

Returns `"\"` in Windows and `":"` on the Macintosh.

Application.PivotTableSelection [= *setting*]

True enables structured selection PivotTable reports; False disables. Default is False.

Application.PreviousSelections([*index*])

Returns one of the four last-selected ranges entered in the Go To dialog box.

Application.ProductCode

Returns the programmatic ID (`ProgId`) of Excel. This value is a globally unique identifier (GUID) used in Windows programming.

Application.PromptForSummaryInfo [= *setting*]

True prompts the user for the workbook properties when files are first saved; False does not prompt. Default is False.

Application.Quit( )

Exits Excel. Excel prompts to save changes before closing unless `DisplayAlerts` is set to False or the workbook's `Saved` property is set to True.

[Application.]Range([*cell1*],[*cell2*])

Returns a range of cells.

Argument	Setting
cell1	The upper-left corner of the range
cell2	The lower-right corner of the range

The three ways to specify the `Range` method are cell references, strings, or brackets. The following three lines all select the same range:

```
Range(Cells(1, 1), Cells(3, 3)).Select
Range("A1", "C3").Select
[A1:C3].Select
```

Application.Ready

Returns True if Excel is ready for input, False otherwise. Excel is not "ready" while a user is editing a cell (edit mode) or when a dialog box is displayed. In those situations, macros must wait to run.

Application.RecentFiles([*index*])

Returns the `RecentFiles` collection. `RecentFiles` represents the list of recently used files displayed at the bottom of the File menu. For example, the following code displays the path- and filenames for each file in the Recent Files list:

```
Sub TestRecentFiles( )
    Dim f As RecentFile
    For Each f In Application.RecentFiles
        Debug.Print f.Path
    Next
End Sub
```

Application.RecordMacro([*BasicCode*], [*XlmCode*])

Sets the code for Excel to record if the user selects Tools → Macro → Record New Macro and then performs a task that runs this macro.

Argument	Setting
BasicCode	The string to record in place of the default
XlmCode	Obsolete

By default, Excel records `Application.Run "workbook!macro"` whenever a user runs a macro while recording. To prevent recording, set *BasicCode* to "" for the macro:

```
Sub SecretMacro( )
    ' Don't record this!
    Application.RecordMacro ""
    ' Secret stuff...
End Sub
```

Application.RecordRelative [= *setting*]

True uses relative references when recording; False uses absolute references. Default is False.

Application.ReferenceStyle [=*XlReferenceStyle*]

Sets or returns the style Excel uses to refer to cells. Can be one of these settings:

```
xlA1
xlR1C1
```

Application.RegisteredFunctions

Returns an array of DLL functions registered with Excel. The following code displays a list of the registered functions:

```
Sub TestRegisteredFunctions( )
    Dim i As Integer, func
    func = Application.RegisteredFunctions
    Debug.Print "DLL", "Function", "Arguments/Return type"
    If Not IsNull(func) Then
        For i = 1 To UBound(func, 1)
         Debug.Print func(i, 1), func(i, 2), func(i, 3)
        Next
    Else
     Debug.Print "No functions registered."
    End If
End Sub
```

Application.RegisterXLL(*Filename*)

Loads an Excel DLL (XLL) and registers it.

Argument	Setting
Filename	The name of the file to register

Application.Repeat( )

Repeats the last user action.

Application.ReplaceFormat [= *setting*]

Sets or returns the `CellFormat` object used when reformatting during search and replace. For example, the following code replaces all bold with italic:

```
Sub TestReplaceFormat( )
    Dim fBold As CellFormat, fItal As CellFormat
    Set fBold = Application.FindFormat
    Set fItal = Application.ReplaceFormat
    fBold.Font.Bold = True
    fItal.Font.Bold = False
    fItal.Font.Italic = True
    Cells.Replace "", "", , , , , True, True
End Sub
```

Application.RollZoom [= *setting*]

True sets the IntelliMouse wheel to zoom the display rather than scroll it; False sets the wheel to scroll. Default is False.

[Application.]Rows([*index*])

Returns a range containing the cells in a row on the active worksheet. For example, the following code selects row 3:

```
Rows(3).Select
```

Application.RTD

Returns a real-time data object.

Application.Run([*Macro*], [*Args*])

Runs a macro.

Argument	Setting
Macro	The name of the macro to run
Args	Arguments for the macro

This method is mainly used by Excel itself when recording user actions that run macros. However, you can also use it to run automated tests during development.

Application.SaveWorkspace([*Filename*])

Saves the current settings as an Excel workspace file.

Argument	Setting
Filename	The name of the file to save. Default is *RESUME.XLW*.

Excel workspace files include the open documents, window placement, and other settings that are restored when the user opens the file. Don't confuse this with shared workspaces, which is a way to share a workbook with others through SharePoint.

Application.ScreenUpdating [= *setting*]

True updates the Excel display as tasks are performed; False hides updates. Default is True. Setting `ScreenUpdating` to False speeds up lengthy operations, such as changing all the cells on a worksheet. Be sure to set this property back to True when done.

Application.Selection

Returns the currently selected objects on the active worksheet. Returns `Nothing` if no objects are selected. Use the `Select` method to set the selection, and use `TypeName` to discover the kind of object that is selected. The following code displays information about the current selection:

```
Sub TestSelection( )
    Dim str As String
    Select Case TypeName(Selection)
    Case "Nothing"
        str = "Please select something."
    Case "Range"
        str = "You selected the range: " & Selection.Address
    Case "Picture"
        str = "You selected a picture."
    Case Else
        str = "You selected a " & TypeName(Selection) & "."
    End Select
    MsgBox str
End Sub
```

Application.SendKeys(*Keys*, [*Wait*])

This method is the same as the Visual Basic `SendKeys` method. See Chapter 3 for details.

Application.SetDefaultChart([*FormatName*], [*Gallery*])

Sets the default chart type used by Excel.

Argument	Setting
FormatName	Can be one of the `XlChartType` constants, `xlBuiltIn`, or the name of a custom chart type
Gallery	Not used

For example, the following code sets the default chart type to a 3-D style:

```
Sub TestSetChartType( )
    Application.SetDefaultChart XlChartType.xl3DArea
End Sub
```

[Application.]Sheets([index])

Returns the `Worksheet` and `Chart` objects in the active workbook. `Sheets` is a mixed collection, so you can't count on every item being a specific type. Instead, you must test check the `TypeName` before calling methods on the object, as shown by the following code:

```
Sub TestSheet( )
    Dim itm As Object, ws As Worksheet, ct As Chart
    For Each itm In Sheets
        Select Case TypeName(itm)
            Case "Worksheet"
                Set ws = itm
                Debug.Print ws.UsedRange.Address
            Case "Chart"
                Set ct = itm
                If ct.HasTitle Then _
                    Debug.Print ct.ChartTitle
            Case Else
                Debug.Print TypeName(itm)
        End Select
    Next
End Sub
```

Use the `Worksheets` or `Charts` method to get those specific object types.

Application.SheetsInNewWorkbook [= *setting*]

Gets or sets the number of worksheets automatically included in new workbooks. Default is 3.

Application.ShowChartTipNames [= *setting*]

True shows the names of items on a chart as tool tips; False hides the names. Default is True.

Application.ShowChartTipValues [= *setting*]

True includes the values of series points in the tool tips displayed on a chart; False hides the values. Default is True.

Application.ShowStartupDialog [= *setting*]

True displays the New Workbook task pane when the user chooses File → New; False creates the workbook without displaying the task pane. Default is True.

Application.ShowToolTips [= *setting*]

True displays pop-up tool tips when the mouse pointer pauses over a toolbar button; False does not display tool tips. Default is True.

Application.ShowWindowsInTaskbar [= *setting*]

True displays each open workbook as a separate instance of Excel with its own item on the Windows task bar; False collects all workbooks into a single instance of Excel with only one task bar item. Default is True.

`ShowWindowsInTaskbar` affects only how Excel appears in Windows. It doesn't affect how much memory it uses or the number of processes running for Excel.

Application.SmartTagRecognizers

Returns a collection of `SmartTagRecognizer` objects.

Application.Speech

Returns a `Speech` object that can be used to say words. Using Speech causes an error if the feature is not installed. The following code tries to say "Hazelnootpasta":

```
Sub TestSpeech( )
    On Error Resume Next
    Application.Speech.Speak "Hazelnootpasta"
    If Err Then MsgBox "Speech not installed."
End Sub
```

Application.SpellingOptions

Returns a `SpellingOptions` object that you can use to control how Excel performs spellchecking. The following code displays the main spelling option settings:

```
Sub TestSpellingOptions( )
    With Application.SpellingOptions
        Debug.Print .DictLang
        Debug.Print .IgnoreCaps
        Debug.Print .IgnoreMixedDigits
        Debug.Print .IgnoreFileNames
        Debug.Print .SuggestMainOnly
    End With
End Sub
```

Application.StandardFont [= *setting*]

Sets or returns the standard font name. For Windows, the default is Arial.

Application.StandardFontSize [= *setting*]

Sets or returns the standard font point size. For Windows, the default is 10.

Application.StartupPath

Returns the path to the XLSTART directory.

Application.StatusBar [= *setting*]

Sets or returns the text in the Excel status bar.

Application.TemplatesPath

Returns the path to the user's Templates folder.

Application.ThisCell

Returns the `Range` object of the cell calling the current user-defined function.

Application.ThisWorkbook

Returns the `Workbook` object of the Excel file that contains the current procedure. `ThisWorkbook` is different from `ActiveWorkbook` in that `ActiveWorkbook` changes based on the current selection, whereas `ThisWorkbook` always refers to the file that contains the running code.

Application.ThousandsSeparator [= *setting*]

Sets or returns the character used to separate thousands.

Application.Top [= *setting*]

Sets or returns the distance between the top of the Excel window and the top of the screen.

Application.Undo( )

Cancels the last user action.

[Application.]Union(*Arg1*, *Arg2*, [*Argn*])

Joins two or more Range objects into a single Range.

Argument	Setting
Arg1	The first Range object to join
Arg2	The second Range object to join
Argn	Any number of additional Range objects to join

Application.UsableHeight

Returns the maximum height of the usable area of Excel in points. This is the Height minus the title, menu, tool, status bars, and column header.

Application.UsableWidth

Returns the maximum width of the usable area of Excel in points. This is the Width minus the scrollbar and row header.

Application.UsedObjects

Returns a collection of all the objects used in Excel. This code displays the names and types of all the objects currently in use by Excel:

```
Sub TestUsedObjects( )
    Dim o, name As String
    On Error Resume Next
    Debug.Print "Type", "Name"
    For Each o In Application.UsedObjects
        name = o.name
        Debug.Print TypeName(o), name
    Next
End Sub
```

Application.UserControl

Returns True if Excel is visible, False if Excel was started programmatically and is not visible. When UserControl is False, Excel quits if there are no references to it.

Application.UserLibraryPath

Returns the path to the user's Addins folder.

Application.UserName [= *setting*]

Sets or returns the user's name.

Application.UseSystemSeparators [= *setting*]

True uses the operating system settings for thousands and decimal separators; False uses the Excel settings. Default is True.

Application.VBE

Returns the `VBE` object that represents the Visual Basic Editor. The following code displays the Visual Basic Editor:

```
Private Sub cmdViewCode_Click( )
    On Error Resume Next
    Application.VBE.MainWindow.Visible = True
    ' An error occurs if security settings prohibit this.
    If Err Then
        MsgBox "You must change Macro security options " & _
          "before you can view code in this way. " & _
          "Choose Tools>Macro>Security>Trusted Publishers and " & _
          "select Trust access to Visual Basic Project."
    End If
End Sub
```

Application.Version

Returns the Excel version number. For example, Excel 2003 returns `11.0`.

Application.Visible [= *setting*]

True if the Excel window is visible; False if it is hidden. When Excel is not visible, it doesn't appear on the task bar, and the only way to close the application may be to use the Task Manager (Ctrl-Delete) in Windows.

Application.Volatile([*Volatile*])

Marks a user-defined function for recalculation whenever any cells on the worksheet are recalculated.

Argument	Setting
Volatile	True causes the function to recalculate when any cell on the worksheet is recalculated; False recalculates only when the input values change. Default is True.

Application.Wait(*Time*)

Pauses Excel.

Argument	Setting
Time	The time to resume Excel

You can specify an interval of time to wait by incrementing `Now`. The following code uses that technique to create a procedure that pauses for an interval specified in milliseconds (the same as the Windows API `Sleep` function):

```
Sub TestSleep( )
    ' Wait 5 seconds.
    Sleep 5000
    MsgBox "Time's up!"
End Sub

Sub Sleep(milsecs As Long)
    Dim dt As Date
    ' 0.00001 = 1 second in the Date type.
    dt = Now + (milsecs / 100000000)
    Application.Wait (dt)
End Sub
```

Application.Watches([*index*])

Returns a collection of `Watch` objects that represent items in a Watch window.

Application.Width [= *setting*]

Sets or returns the width of the Excel window in pixels.

Application.Windows([*index*])

Returns a collection of `Window` objects that represent the windows displayed by Excel.

Application.WindowsForPens

Returns True if Excel is running under Windows for Pen Computing, False otherwise.

Application.WindowState [= *XlWindowState*]

Sets or returns the state of the Excel window. Can be one of these settings:

```
xlMaximized
xlNormal
xlMinimized
```

[Application.]Workbooks([*index*])

Returns a collection of `Workbook` objects that represent workbooks that are currently open in Excel.

[Application.]WorksheetFunction

Returns the `WorksheetFunction` object, which is used to access Excel's built-in functions. See Chapter 4 for a description of the available functions.

[Application.]Worksheets([*index*])

Returns a collection containing the `Worksheet` objects in the active workbook. This is different from the `Sheets` collection, which returns `Worksheet`, `Chart`, and other types of sheet objects.

AutoCorrect Members

The `AutoCorrect` object has the following members. Key members (shown in **bold**) are covered in the following reference section:

`AddReplacement`
`AutoExpandListRange`
`CorrectCapsLock`
`Creator`
`DisplayAutoCorrectOptions`
`ReplacementList`
`TwoInitialCapitals`
`Application`
`CapitalizeNamesOfDays`
`CorrectSentenceCap`
`DeleteReplacement`
`Parent`
`ReplaceText`

The `AutoCorrect` object provides a set of properties that determine how Excel handles automatic correction. Most of the `AutoCorrect` members are True/False properties that enable or disable specific Auto Correct options. The following code displays a list of the current Auto Correct settings in Excel:

```
Sub ShowAutoCorrectSettings( )
    With Application.AutoCorrect
        Debug.Print .AutoExpandListRange
        Debug.Print .CapitalizeNamesOfDays
        Debug.Print .CorrectCapsLock
        Debug.Print .CorrectSentenceCap
        Debug.Print .DisplayAutoCorrectOptions
        Debug.Print .ReplaceText
        Debug.Print .TwoInitialCapitals
    End With
End Sub
```

These properties correspond to the settings on the AutoCorrect dialog box (Figure 7-4). To see that dialog, choose Tools → AutoCorrect Options.

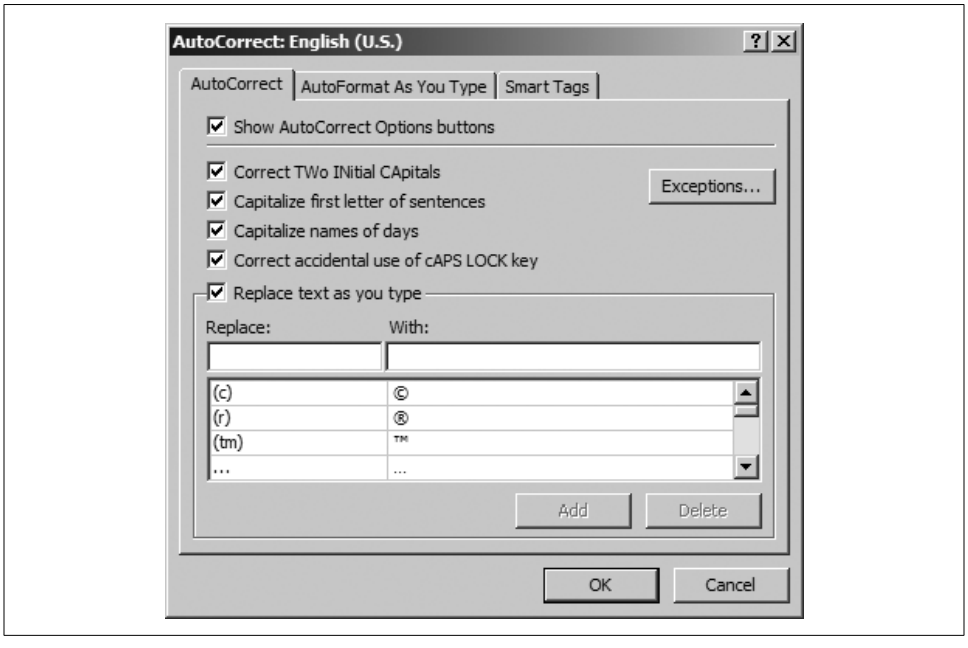

Figure 7-4. Displaying the AutoCorrect options

AutoCorrect.AddReplacement(*What*, *Replacement*)

Adds an item to the replacement list shown at the bottom of Figure 7-4.

Argument	Setting
What	The typed sequence to automatically correct
Replacement	The correction to use

AutoCorrect.DeleteReplacement(*What*)

Deletes an item from the replacement list.

Argument	Setting
What	The typed sequence to delete from the replacement list

AutoCorrect.ReplacementList

Returns the replacement list. The following code displays the list of items that Excel will automatically replace and the replacements that will be used:

```
Sub ShowReplacementList( )
    Dim i As Integer
    With Application.AutoCorrect
        Debug.Print "Replace", "With"
        For i = 1 To UBound(.ReplacementList, 1)
            Debug.Print .ReplacementList(i)(1), _
              .ReplacementList(i)(2)
        Next
    End With
End Sub
```

AutoRecover Members

The `AutoRecover` object has the following members. Key members (shown in **bold**) are covered in the following reference section:

`Application`
`Creator`
`Enabled`
`Parent`
`Path`
`Time`

AutoRecover.Enabled [= *setting*]

True enables automatic recovery; False disables it.

AutoRecover.Path [= *setting*]

Sets or returns the path where Excel stores the files used by automatic recovery.

AutoRecover.Time [= *setting*]

Sets or returns the number of minutes between when automatic recovery files are saved. Must be between 1 and 120. Default is 10.

ErrorChecking Members

The ErrorChecking object has the following members:

```
Application                    BackgroundChecking
Creator                        EmptyCellReferences
EvaluateToError                InconsistentFormula
IndicatorColorIndex            ListDataValidation
NumberAsText                   OmittedCells
Parent                         TextDate
UnlockedFormulaCells
```

Most of the ErrorChecking members are True/False properties that enable or disable specific error-checking options. The following code displays a list of the current error-checking settings in Excel:

```
Sub ShowErrorCheckingSettings( )
    With Application.ErrorCheckingOptions
        Debug.Print .BackgroundChecking
        Debug.Print .EmptyCellReferences
        Debug.Print .EvaluateToError
        Debug.Print .InconsistentFormula
        Debug.Print .IndicatorColorIndex
        Debug.Print .ListDataValidation
        Debug.Print .NumberAsText
        Debug.Print .OmittedCells
        Debug.Print .TextDate
        Debug.Print .UnlockedFormulaCells
    End With
End Sub
```

These properties correspond to the settings on the Error Checking dialog box shown in Figure 7-5. To see the dialog, choose Tools → Error Checking → Options.

SpellingOptions Members

The SpellingOptions object has the following members:

```
ArabicModes                    DictLang
GermanPostReform               HebrewModes
IgnoreCaps                     IgnoreFileNames
IgnoreMixedDigits              KoreanCombineAux
KoreanProcessCompound          KoreanUseAutoChangeList
SuggestMainOnly                UserDict
```

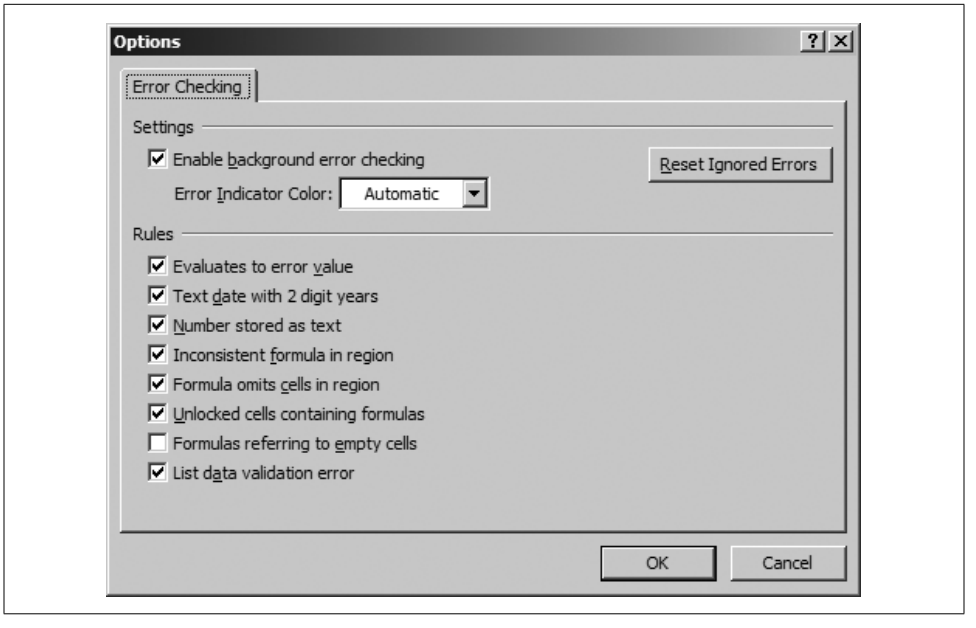

Figure 7-5. Displaying error-checking options

The `SpellingOptions` object provides a set of properties that determine how Excel handles spellchecking. All of the `Spelling` members are read/write properties that enable or disable specific options. The following code displays a list of the current spell-checking settings in Excel:

```
Sub ShowSpellCheckSettings( )
    With Application.SpellingOptions
        Debug.Print .DictLang
        Debug.Print .IgnoreCaps
        Debug.Print .IgnoreMixedDigits
        Debug.Print .SuggestMainOnly
        Debug.Print .UserDict
    End With
End Sub
```

These properties correspond to the settings on the Spelling tab of the Options dialog box (Figure 7-6). To see that dialog, choose Tools → Options → Spelling.

Language-specific settings in Figure 7-6 are disabled because my selected language is English (U.S.). You must install those language versions of Excel to use those settings.

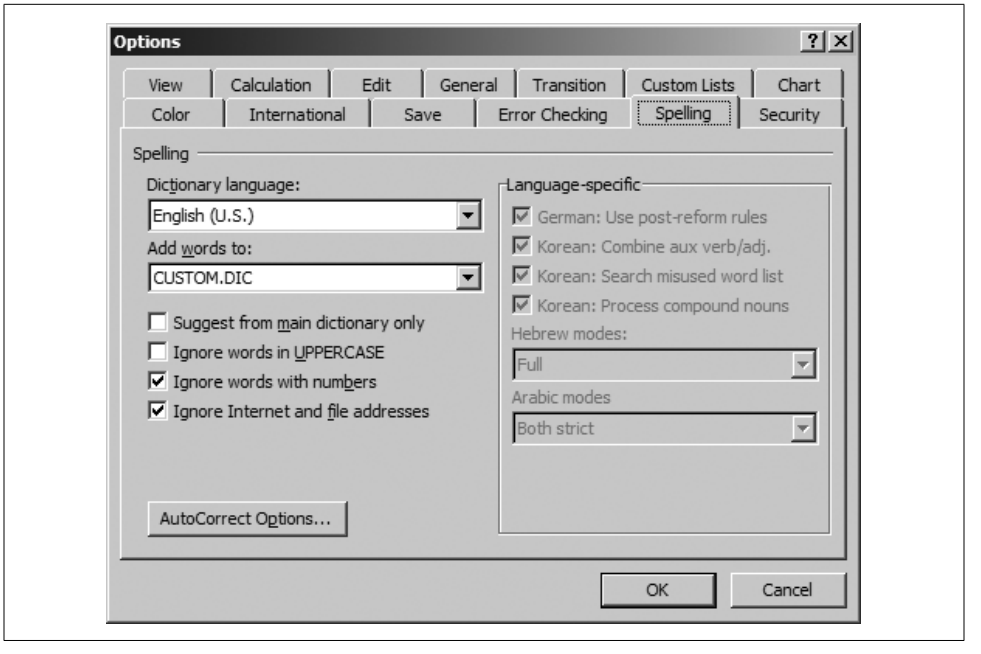

Figure 7-6. Displaying the spellchecking options

Window and Windows Members

The `Window` object and `Windows` collection have the following members. Key members (shown in **bold**) are covered in the following reference section:

Activate	**ActivateNext**	**ActivatePrevious**
ActiveCell	ActiveChart	ActivePane
ActiveSheet	Application[2]	**Arrange**[1]
BreakSideBySide[1]	Caption	**Close**
CompareSideBySideWith[1]	Count[1]	Creator[2]
DisplayFormulas	**DisplayGridlines**	**DisplayHeadings**
DisplayHorizontalScrollBar	**DisplayOutline**	**DisplayRightToLeft**
DisplayVerticalScrollBar	**DisplayWorkbookTabs**	**DisplayZeros**
EnableResize	**FreezePanes**	**GridlineColor**
GridlineColorIndex	Height	Index
LargeScroll	Left	**Panes**
Parent[1]	**PointsToScreenPixelsX**	**PointsToScreenPixelsY**
RangeFromPoint	**RangeSelection**	**ResetPositionsSideBySide**[1]
ScrollColumn	**ScrollIntoView**	**ScrollRow**
ScrollWorkbookTabs	**SelectedSheets**	**Selection**
SmallScroll	**Split**	**SplitColumn**

SplitHorizontal	**SplitRow**	**SplitVertical**
SyncScrollingSideBySide[1]	**TabRatio**	Top
Type	UsableHeight	UsableWidth
View	Visible	**VisibleRange**
Width	**WindowNumber**	**WindowState**
Zoom		

[1] Collection only
[2] Object and collection

Use the `Windows` collection and `Window` objects to control which window has focus in Excel and to open, close, arrange, and control the appearance of Excel windows. Use the `Application` object's `ActiveWindow` property to get the window that currently has focus, or use the `Windows` collection to choose a specific window.

The following code demonstrates the most common window tasks:

```
Sub TestWindows( )
    Dim i As Integer, wnd As Window
    Dim curWnd As Window, curState As XlWindowState
    ' Save the current settings
    Set curWnd = ActiveWindow
    curState = curWnd.WindowState
    ' Create four new windows.
    For i = 1 To 4
        Set wnd = curWnd.NewWindow
        wnd.Caption = "New Window: " & i
    Next
    ' Cascade the windows.
    Application.Windows.Arrange (xlArrangeStyleCascade)
    ' Activate each in turn.
    For Each wnd In Application.Windows
        wnd.Activate
        ' Wait 1 second.
        API.Sleep (1000)
    Next
    ' Close created windows
    For Each wnd In Application.Windows
       If wnd.Caption Like "New Window: ?" Then wnd.Close
    Next
    ' Restore original window and state.
    curWnd.Activate
    curWnd.WindowState = curState
End Sub
```

window.Activate( )

Sets focus on the window, bringing it to the top.

window.ActivateNext( )

Sets focus to the next window in the Excel windows list.

window.ActivatePrevious( )

Sets focus to the previous window in the Excel windows list.

windows.Arrange([*ArrangeStyle*], [*ActiveWorkbook*], [*SyncHorizontal*], [*SyncVertical*])

Arranges the Excel windows.

Argument	Setting
ArrangeStyle	Can be one of these `XlArrangeStyle` settings: `xlArrangeStyleCascade`, `xlArrangeStyleTiled` (default), `xlArrangeStyleHorizontal`, `xlArrangeStyleVertical`.
ActiveWorkbook	True arranges only the windows of the active workbook; False arranges all workbooks. Default is False.
SyncHorizontal	True links the windows so that they scroll together horizontally; False allows independent scrolling. Default is False. Ignored if `ActiveWorkbook` is not True.
SyncVertical	True links the windows so that they scroll together vertically; False allows independent scrolling. Default is False. Ignored if `ActiveWorkbook` is not True.

windows.BreakSideBySide( )

Ends the side-by-side comparison of two workbooks. See `CompareSideBySideWith` for details.

window.Close([*SaveChanges*], [*Filename*], [*RouteWorkbook*])

Close the window. Closing the last open window for a workbook closes the workbook, so `Close` has these arguments in the following table that determine what to do in that case:

Argument	Setting
SaveChanges	True saves changes to the workbook; False abandons changes. Prompts the user if omitted.
Filename	The name of the file to save the workbook as; default is the current filename.
RouteWorkbook	If the workbook has a routing slip attached, True routes the workbook to the next recipient; False does not route. Prompts the user if omitted.

windows.CompareSideBySideWith(*WindowName*)

Starts side-by-side comparison between the active window and another window. Side-by-side comparison links the scrolling of the two windows so that you can more easily compare different versions of a workbook. Use `BreakSideBySide` to turn off this comparison.

The following code demonstrates turning side-by-side comparison on and off. Ordinarily, you would open two existing versions of a workbook, but I create the second version here so that the demonstration is self-contained:

```
Sub TestBeginSideBySide( )
    Dim fpath As String, wnd As Window
    ' Get the window for active workbook.
    Set wnd = Application.ActiveWindow
    ' Get the workbook's full filename.
    fname = ActiveWorkbook.Path & "\" & ActiveWorkbook.name
    ' Change it to a new filename.
    fname = VBA.Replace(fname, ".xls", "_v2.xls")
    ' Save a copy of the workbook.
    ActiveWorkbook.SaveCopyAs fpath
    ' Open the copy (makes the copy the active window).
    Application.Workbooks.Open fname
    ' Turn on side-by-side comparision.
    Application.Windows.CompareSideBySideWith wnd.Caption
End Sub

Sub TestEndSideBySide( )
    ' Turn off side-by-side comparision.
    Application.Windows.BreakSideBySide
End Sub
```

window.DisplayFormulas [= *setting*]

True displays formulas in cells; False displays result of formulas (values). Default is False.

window.DisplayGridlines [= *setting*]

True displays gridlines showing cell boundaries; False hides gridlines. Default is True.

window.DisplayHeadings [= *setting*]

True displays column headings (A, B, C, ...); False hides headings. Default is True.

window.DisplayHorizontalScrollBar [= *setting*]

True displays the horizontal scrollbar; False hides it. Default is True.

window.DisplayOutline [= *setting*]

True displays outlining symbols; False hides them. Default is True. To outline a worksheet, choose Data → Group and Outline → Auto Outline. The outlining symbols appear to the left of the row numbers.

window.DisplayRightToLeft [= *setting*]

True displays Excel in right-to-left fashion; False displays Excel left-to-right. `DisplayRightToLeft` is used for locales with left-to-right languages, such as Saudi Arabia.

window.DisplayVerticalScrollBar [= *setting*]

True displays the vertical scrollbar; False hides it. Default is True.

window.DisplayWorkbookTabs [= *setting*]

True displays the sheet tabs at the bottom of the workbook; False hides them. Default is True.

window.DisplayZeros [= *setting*]

True displays zero values as `0` in cells; False hides zero values. Default is True.

window.EnableResize [= *setting*]

True allows the user to resize the window; False prohibits resizing. Default is True. Accessing this property causes an error if `WindowState` is not `xlNormal`. The following code prevents the user from changing the active window's size:

```
Sub TestDisableResize( )
    If ActiveWindow.WindowState = xlNormal Then _
        ActiveWindow.EnableResize = False
End Sub
```

window.FreezePanes [= *setting*]

True locks panes to prevent horizontal and vertical scrolling; False allows panes to scroll. Default is False.

window.GridlineColor [= *setting*]

Sets or returns the color of gridlines as an RGB color. RGB colors are long integers that you can create using the RGB function or (commonly) by specifying a value in hexadecimal. The following code changes the grid color to red, green, blue, and back to normal:

```
Sub TestGridlineColor( )
    ' Change grid color using hexidecimal values.
    ActiveWindow.GridlineColor = &HFF     ' Red
    ' Wait 1 second.
    API.Sleep (1000)
    ActiveWindow.GridlineColor = &HFF00   ' Green
    API.Sleep (1000)
    ActiveWindow.GridlineColor = &HFF0000 ' Blue
    API.Sleep (1000)
    ' Restore the default.
    ActiveWindow.GridlineColorIndex = xlColorIndexAutomatic
End Sub
```

window.GridlineColorIndex [=*xlColorIndexAutomatic*]

Sets or returns the color of the gridlines based on the index into the color palette. Default is xlColorIndexAutomatic.

window.LargeScroll([*Down*], [*Up*], [*ToRight*], [*ToLeft*])

Scrolls the window a number of pages in a given direction. You can combine arguments to scroll diagonally.

Argument	Setting
Down	Number of pages to scroll down
Up	Number of pages to scroll up
ToRight	Number of pages to scroll right
ToLeft	Number of pages to scroll left

window.Panes

Returns the collection of Panes objects for the window. Windows that are not split return one pane.

window.PointsToScreenPixelsX(*Points*)

Converts an application width measurement of points to a screen measurement in pixels. The following code displays the screen dimensions in pixels:

```
Sub TestPointsToPixels( )
    Application.DisplayFullScreen = True
```

```
    Debug.Print Application.Windows(1).PointsToScreenPixelsX(Application.Width)
    Debug.Print Application.Windows(1).PointsToScreenPixelsX(Application.Height)
    Application.DisplayFullScreen = False
End Sub
```

window.PointsToScreenPixelsY(*Points*)

Converts the application height measurement of points to a screen measurement in pixels.

window.RangeFromPoint(*x*, *y*)

Returns the `Range` object at the specified *x* and *y* coordinates. Coordinates are in pixels, not points.

window.RangeSelection

Returns a `Range` object containing the selected cells on the window. `RangeSelection` is slightly different from `Selection`, since `Selection` can include drawing objects as well as ranges.

windows.ResetPositionsSideBySide( )

Restores the side-by-side comparison display after one of the windows is maximized or minimized while the user is doing a comparison.

window.ScrollColumn [= *setting*]

Sets or returns the column number displayed in the leftmost side of the Excel window.

window.ScrollIntoView(*Left*, *Top*, *Width*, *Height*, [*Start*])

Scrolls the window to a rectangular region on the worksheet.

Argument	Setting
Left	The left edge of the rectangle in points.
Top	The top edge of the rectangle in points.
Width	The width of the rectangle in points.
Height	The height of the rectangle in points.
Start	True scrolls the upper-left corner of the rectangle to the upper-left corner of the window; False scrolls the lower-right corner of the rectangle to the lower-right corner of the window. Default value is True.

window.ScrollRow [= *setting*]

Sets or returns the row number displayed at the top of the Excel window.

window.ScrollWorkbookTabs([*Sheets*], [*Position*])

Scrolls the worksheet tabs displayed at the bottom of a workbook.

Argument	Setting
Sheets	The number of tabs to scroll in either direction. Positive values scroll to the right; negative values scroll left.
Position	Can be one of the following settings: `xlFirst`, `xlLast`.

window.SelectedSheets

Returns the collection of worksheets and charts selected in the window. More than one sheet can be selected by multiselecting the sheet tabs at the bottom of the window.

window.Selection

Returns the objects selected on the window.

window.SmallScroll([*Down*], [*Up*], [*ToRight*], [*ToLeft*])

Scrolls the window a number of rows or columns in a given direction. You can combine arguments to scroll diagonally.

Argument	Setting
Down	Number of rows to scroll down
Up	Number of rows to scroll up
ToRight	Number of columns to scroll right
ToLeft	Number of columns to scroll left

window.Split [= *setting*]

True splits the window into panes; False displays the window as a single pane. Default is False. Use `Split` in combination with the following properties to divide a window into panes. For example, the following code splits the active window vertically at column C:

```
Sub TestSplitVertically( )
    With ActiveWindow
        .SplitColumn = 3
        .SplitRow = 0
        .Split = True
    End With
End Sub
```

window.SplitColumn [= *setting*]

Sets or returns the column number at which to split a window vertically.

window.SplitHorizontal [= *setting*]

Sets or returns the location in points at which to split a window horizontally.

window.SplitRow [= *setting*]

Sets or returns the row number at which to split a window horizontally.

window.SplitVertical [= *setting*]

Sets or returns the location in points at which to split a window vertically.

windows.SyncScrollingSideBySide [= *setting*]

True synchronizes the two windows displayed during side-by-side comparison so that scrolling one window scrolls the other window an equal amount; False allows the windows to scroll independently.

window.TabRatio [= *setting*]

Sets or returns the ratio between the width of the tab area and the width of the window's horizontal scrollbar. Default is 0.6.

window.View [= *XlWindowView*]

Sets or returns whether page breaks are displayed. Can be one of these settings:

```
xlNormalView
xlPageBreakView
```

window.VisibleRange

Returns the `Range` object that is visible on the window.

window.*Window*Number [= *setting*]

Returns the number portion of the window caption. For example, the window captioned `ch07.xls:2` returns `2`.

window.*Window*State [= *XlWindowState*]

Sets or returns the state of the window. Can be one of these settings:

```
xlMaximized
xlNormal
xlMinimized
```

window.Zoom [= *setting*]

Sets or returns a percentage by which to magnify the window.

Pane and Panes Members

The Pane object and Panes collection have the following members. These members are the same as the Window members of the same name.

Activate	Application[2]
Count[1]	Creator[2]
Index	LargeScroll
Parent[2]	ScrollColumn
ScrollIntoView	ScrollRow
SmallScroll	VisibleRange

[1] Collection only
[2] Object and collection

Pane objects represent the regions of a window. By default, Excel windows have one pane; additional panes are created when the user or code splits the window into two or four regions.

The following code demonstrates splitting the active window into four panes, then scrolling each of those panes:

```
Sub TestPanes( )
    Dim pn As Pane, down As Integer, right As Integer
    Dim i As Integer
    With ActiveWindow
        ' Set the location for the split.
        .SplitColumn = 10
        .SplitRow = 16
        ' Split into four panes.
        .Split = True
        For i = 1 To .Panes.Count
            down = i * 2
            right = i + 3
```

```
            ' Scroll each pane.
            .Panes(i).SmallScroll down, , right
        Next
    End With
End Sub
```

The preceding code demonstrates two key things:

- The Panes collection can't be used in a For Each statement. Instead, you must use For Next.
- Scrolling is cumulative for pairs of panes. In other words, the horizontal pairs of panes are always on the same row and the vertical pairs are always on the same column.

To close panes, set the Window object's Split property to False:

```
Sub TestClosePanes( )
    ActiveWindow.Split = False
End Sub
```