

Sharing Data with Other Applications

Windows applications are designed to work together. The applications in Microsoft Office are an excellent example. These programs have a common look and feel, and sharing data among these applications is quite easy. This chapter explores some ways that you can make use of other applications while working with Excel, as well as some ways that you can use Excel while working with other applications.

Sharing Data with Other Windows Applications

Besides importing and exporting files, the following are the essential three ways in which you can transfer data to and from other Windows applications:

- Copy and paste, using either the Windows Clipboard or the Office Clipboard. Copying and pasting information creates a static copy of the data.
- Create a link so that changes in the source data are reflected in the destination document.
- Embed an entire object from one application into another application's document.

The following sections discuss these techniques and present an example for each one.

29

CHAPTER

In This Chapter

Sharing Data with Other Windows Applications

Using the Windows or Office Clipboards

Linking Data

Copying Excel Data to Word

Embedding Objects

Using Office Binders

Using the Windows or Office Clipboards

As you probably know, whenever Windows is running, you have access to the Windows Clipboard—an area of your computer's memory that acts as a shared holding area for information that you have cut or copied from an application. The Windows Clipboard works behind the scenes, and you usually aren't aware of it. Whenever you select data and then choose either Edit • Copy or Edit • Cut, the application places the selected data on the Windows Clipboard. Like most other Windows applications, Excel can then access the Clipboard data if you choose the Edit • Paste command (or the Edit • Paste Special command).



If you copy or cut information while working in an Office application, the application places the copied information on both the Windows Clipboard and the Office Clipboard.



Once you copy information to the Windows Clipboard, it remains on the Windows Clipboard even after you paste it, so you can use it multiple times. However, because the Windows Clipboard can hold only one item at a time, when you copy or cut something else, the information previously stored on the Windows Clipboard is replaced. The Office Clipboard, unlike the Windows Clipboard, can hold up to 12 separate selections. The Office Clipboard operates in all Office applications; for example, you can copy two selections from Word and three from Excel and paste any or all of them in PowerPoint.

Copying information from one Windows application to another is quite easy. The application that contains the information that you're copying is called the *source application*, and the application to which you're copying the information is called the *destination application*.

The general steps that are required to copy from one application to another are as follows. These steps apply to copying from Excel to another application and to copying from another application to Excel.

1. Activate the source document window that contains the information that you want to copy.
2. Select the information by using the mouse or the keyboard. If Excel is the source application, this information can be a cell, range, chart, or drawn object.
3. Select Edit • Copy. Excel places a copy of the information onto the Windows Clipboard and the Office Clipboard.
4. Activate the destination application. If the program isn't running, you can start it without affecting the contents of the Clipboard.
5. Move to the appropriate position in the destination application (where you want to paste the copied material).
6. Select Edit • Paste from the menu in the destination application. If the Clipboard contents are not appropriate for pasting, the Paste command is grayed (not available).

In Step 3 in the preceding steps, you also can select **Edit • Cut** from the source application menu. This step erases your selection from the source application after placing the selection on the Clipboard.



If you repeat Step 3 in any Office application, the Office Clipboard toolbar appears automatically. It continues to appear if the destination application that you activate in Step 4 is another Office application.



In Step 6 in the preceding steps, you can sometimes select the **Edit • Paste Special** command, which displays a dialog box that presents different pasting options.

If you're copying a graphics image, you may have to resize or crop it. If you're copying text, you may have to reformat it by using tools that are available in the destination application. The information that you copy from the source application remains intact, and a copy remains on the Clipboard until you copy or cut something else. Figure 29-1 shows an embedded Excel chart. You can easily insert a copy of this chart into a Microsoft Word report. First, select the chart in Excel by clicking it once. Then, copy it to the Clipboard by choosing **Edit • Copy**. Next, activate the Word document into which you want to paste the copy of the chart, and move the insertion point to the place where you want the chart to appear. When you select **Edit • Paste** from the Word menu bar, the chart is pasted from the Clipboard and appears in your document (see Figure 29-2).

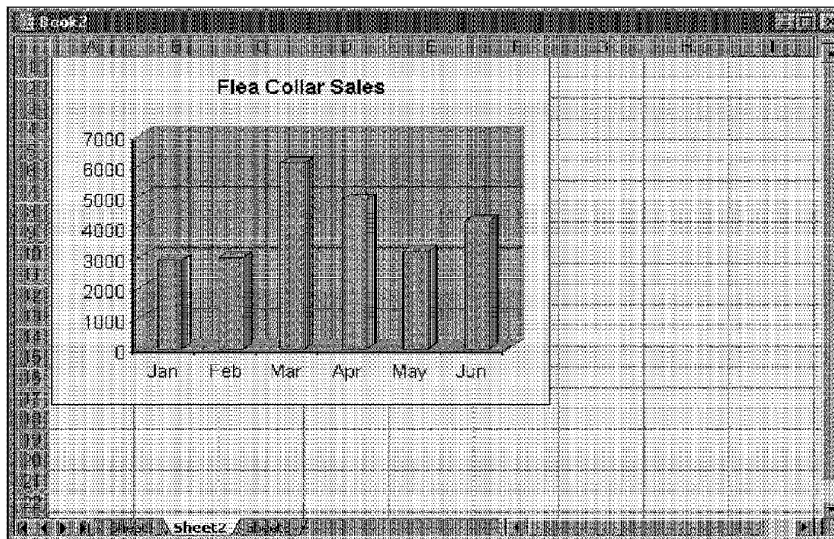


Figure 29-1: An Excel chart, ready to be copied into a Word document.

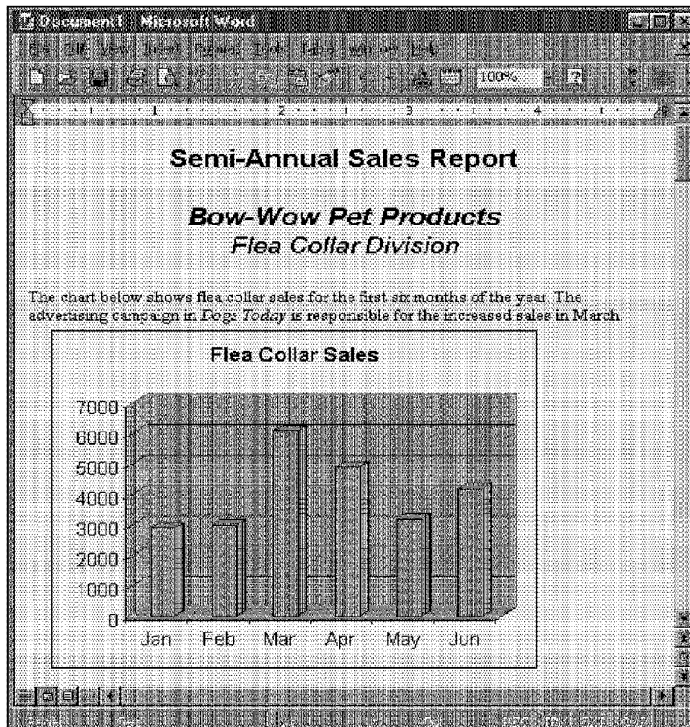


Figure 29-2: The Excel chart copied to a Word document.



Note

You need to understand that Windows applications vary in the way that they respond to data that you paste from the Clipboard. If the Edit • Paste command is not available (is grayed on the menu) in the destination application, the application can't accept the information from the Clipboard. If you copy a range of data from Excel to the Clipboard and paste it into Word, Word creates a table when you paste the data. Other applications may respond differently to Excel data. If you plan to do a lot of copying and pasting, I suggest that you experiment until you understand how the two applications handle each other's data.

You should understand that this copy-and-paste technique is static. In other words, no link exists between the information that you copy from the source application and the information that you paste into the destination application. If you're copying from Excel to a word processing document, for example, the word processing document *will not* reflect any subsequent changes that you make in your Excel worksheet or charts. Consequently, you have to repeat the copy-and-paste procedure to update the destination document with the source document changes. The next topic presents a way to get around this limitation.

Linking Data

If you want to share data that may change, the static copy-and-paste procedure described in the preceding section isn't your best choice. Instead, create a dynamic link between the data that you copy from one Windows application to another. In this way, if you change the data in the source document, you don't *also* need to make the changes in the destination document, because the link automatically updates the destination document.

When would you want to use this technique? If you generate proposals by using a word processor, for example, you may need to refer to pricing information that you store in an Excel worksheet. If you set up a link between your word processing document and the Excel worksheet, you can be sure that your proposals always quote the latest prices. Not all Windows applications support dynamic linking, so you must make sure that the application to which you are copying is capable of handling such a link.

Creating Links

Setting up a link from one Windows application to another isn't difficult, although the process varies slightly from application to application. The following are the general steps to take:

1. Activate the window in the source application that contains the information that you want to copy.
2. Select the information by using the mouse or the keyboard. If Excel is the source application, you can select a cell, range, or entire chart.
3. Select Edit • Copy from the source application's menu. The source application copies the information to the Windows Clipboard.
4. Activate the destination application. If it isn't open, you can start it without affecting the contents of the Clipboard.
5. Move to the appropriate position in the destination application.
6. Select the appropriate command in the destination application to paste a link. The command varies, depending on the application. In Microsoft Office applications, the command is Edit • Paste Special.
7. A dialog box will probably appear, letting you specify the type of link that you want to create. The following section provides more details.

More About Links

Keep in mind the following information when you're using links between two applications:

- Not all Windows applications support linking. Furthermore, you can link *from* but not *to* some programs. When in doubt, consult the documentation for the application with which you're dealing.
- When you save an Excel file that has a link, you save the most recent values with the document. When you reopen this document, Excel asks whether you want to update the links.
- Links can be broken rather easily. If you move the source document to another directory or save it under a different name, for example, the destination document's application won't be able to update the link. You can usually reestablish the link manually, if you understand how the application manages the links. In Excel, you use the Edit • Links command, which displays the Links dialog box, shown in Figure 29-3.
- You also can use the Edit • Links command to break a link. After breaking a link, the data remains in the destination document, but is no longer linked to the source document.
- In Excel, external links are stored in array formulas. If you know what you're doing, you can modify a link by editing the array formula.
- When Excel is running, it responds to link requests from other applications, unless you have disabled remote requests. If you don't want Excel to respond to link-update requests from other applications, choose Tools • Options, select the General tab, and then place a check in the Ignore other applications check box.

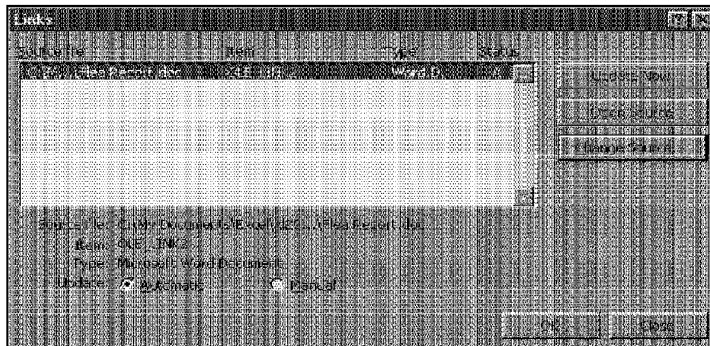


Figure 29-3: The Links dialog box lets you work with links to other applications.

Copying Excel Data to Word

One of the most frequently used software combinations is a spreadsheet and a word processor. This section discusses the types of links that you can create by using Microsoft Word.



Most information in this section also applies to other word processors, such as Corel's WordPerfect for Windows and Lotus Word Pro. The exact techniques vary, however. I use Word in the examples because readers who acquired Excel as part of the Microsoft Office have Word installed on their systems. If you don't have a word processor installed on your system, you can use the WordPad application that comes with Windows. The manner in which WordPad handles links is very similar to that for Word.

Figure 29-4 shows the Paste Special dialog box from Microsoft Word after a range of data has been copied from Excel to the Clipboard. The result that you get depends on whether you select the Paste or the Paste link option, and on your choice of the type of item to paste. If you select the Paste link option, you can choose to have the information pasted as an icon. If you do so, you can double-click this icon to activate the source worksheet.

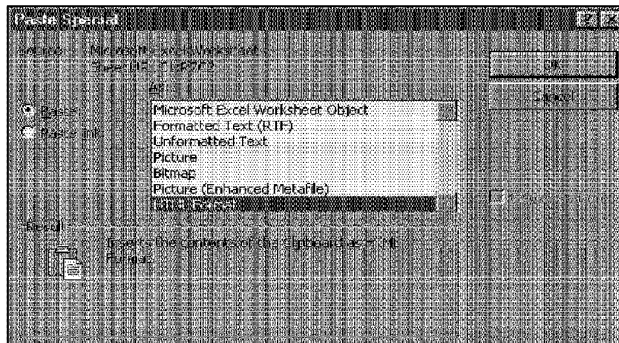


Figure 29-4: The Paste Special dialog box is where you specify the type of link to create.

Pasting Without a Link

Often, you don't need a link when you copy data. For example, if you're preparing a report in your word processor and you simply want to include a range of data from an Excel worksheet, you probably don't need to create a link.

Table 29-1 describes the effect of choosing the various paste choices when you select the Paste option—the option that *doesn't* create a link to the source data.

Table 29-1
Result of Using the Paste Special Command in Word
(Paste Option)

Paste Type	Result
Microsoft Excel Worksheet Object	An object that includes the Excel formatting. This creates an <i>embedded object</i> , described in the next section.
Formatted Text (RTF)	A Word table that is formatted as the original Excel range. No link to the source exists. This produces the same result as using Edit • Paste.
Unformatted Text	Text (not a table) that corresponds to Word's Normal style. Formatting from Excel is not transferred, and no link to the source exists.
Picture	A picture object that retains the formatting from Excel. No link to the source exists. This usually produces better results than the Bitmap option. Double-clicking the object after you paste it enables you to edit the picture.
Bitmap	A bitmap object that retains the formatting from Excel. No link exists to the source. Double-clicking the object after you paste it enables you to edit the bitmap.
HTML Format	A table that is formatted as the original Excel range. No link to the source exists. Use this format when you expect to publish the document as a Web page.

Figure 29-5 shows how a copied range from Excel appears in Word, using each of the paste special formats.

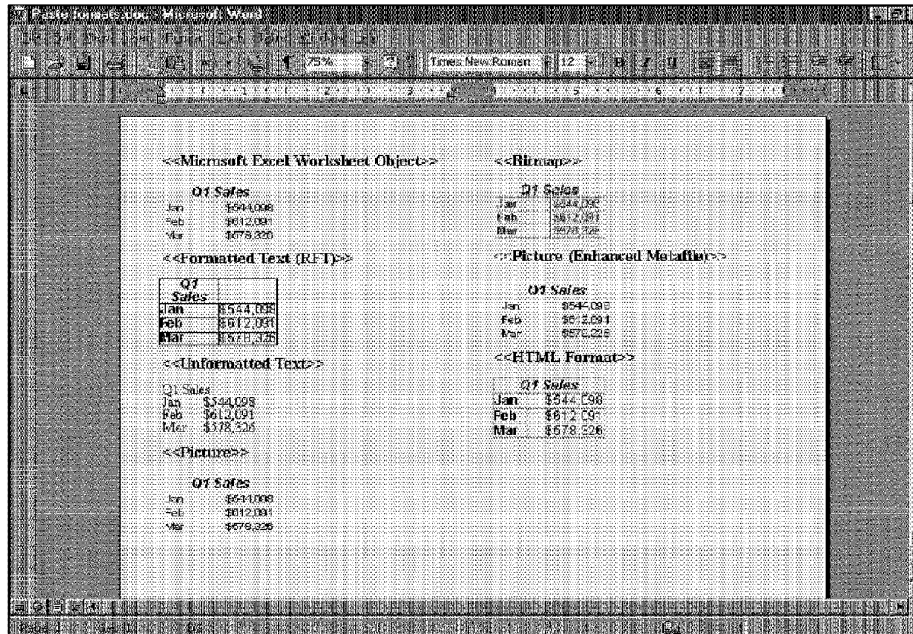


Figure 29-5: Data that is copied from Excel and pasted using various formats.

The pasted data *looks* the same regardless of whether the Paste or Paste link option is selected.

Some Excel formatting does not transfer when pasted to Word as formatted text. For example, Word doesn't support vertical alignment for table cells (but you can use Word's paragraph formatting commands to apply vertical alignment).

Pasting with a Link

If you think the data that you're copying will change, you may want to paste a link. If you paste the data by using the Paste link option in the Paste Special dialog box, you can make changes to the source document, and the changes appear in the destination application (a few seconds of delay may occur). You can test these changes by displaying both applications onscreen, making changes to the source document, and watching for them to appear in the destination document.

Table 29-2 describes the effect of choosing the various paste choices in Word's Paste Special dialog box when the Paste link option is selected.

Table 29-2
Result of Using the Paste Special Command in Word
(Paste Link Option)

<i>Paste Type</i>	<i>Result</i>
Microsoft Excel Worksheet Object	A linked object that includes the Excel formatting. Double-click the object after pasting it to edit the source data in Excel.
Formatted Text (RTF)	A Word table that is formatted as the original Excel range. Changes in the source are reflected automatically.
Unformatted Text	Text (not a table) that corresponds to Word's Normal style. Formatting from Excel is not transferred. Changes in the source are reflected automatically.
Picture	A picture object that retains the formatting from Excel. Changes in the source are reflected automatically. This usually produces better results than the Bitmap option. Double-click the object after pasting it to edit the source data in Excel.
Bitmap	A bitmap object that retains the formatting from Excel. Changes in the source are reflected automatically. Double-click the object after pasting it to edit the source data in Excel.
HTML Format	A table that is formatted as the original Excel range. Use this format when you expect to publish the document as a Web page.

Embedding Objects

Using *Object Linking and Embedding (OLE)*, you can also embed an object to share information between Windows applications. This technique enables you to insert an object from another program and use that program's editing tools to manipulate it. The OLE objects can be items such as those in the following list:

- Text documents from other products, such as word processors
- Drawings or pictures from other products
- Information from special OLE server applications, such as Microsoft Equation
- Sound files
- Video or animation files

Most of the major Windows applications support OLE. You can embed an object into your document in either of two ways:

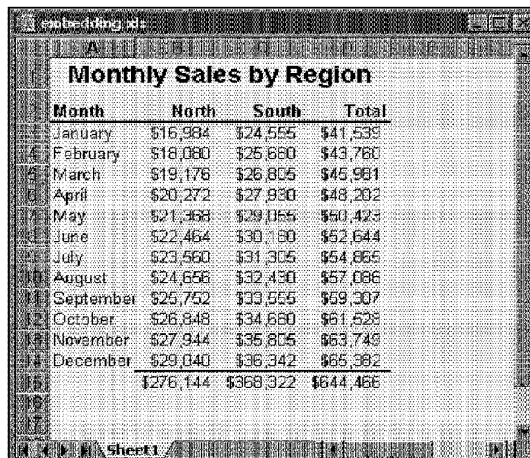
- Choose **Edit • Paste Special**, and select the “object” choice (if it’s available). If you do this, select the **Paste** option rather than the **Paste link** option.
- Select **Insert • Object**.

Some applications — such as those in Microsoft Office — can also embed an object by dragging it from one application to another.

The following sections discuss these two methods and provide a few examples using Excel and Word.

Embedding an Excel Range in a Word Document

This example embeds the Excel range shown in Figure 29-6 in a Word document.



Month	North	South	Total
January	\$16,984	\$24,555	\$41,539
February	\$18,080	\$25,880	\$43,780
March	\$19,176	\$26,805	\$45,981
April	\$20,272	\$27,930	\$48,202
May	\$21,368	\$29,055	\$50,423
June	\$22,464	\$30,180	\$52,644
July	\$23,560	\$31,305	\$54,865
August	\$24,656	\$32,430	\$57,086
September	\$25,752	\$33,555	\$59,307
October	\$26,848	\$34,680	\$61,528
November	\$27,944	\$35,805	\$63,749
December	\$29,040	\$36,930	\$65,970
	\$276,144	\$368,322	\$644,466

Figure 29-6: This range will be embedded in a Word document.

To start, select A1:D15 and copy the range to the Clipboard. Then, activate (or start) Word, open the document in which you want to embed the range, and then move the insertion point to the location in the document where you want the table to appear. Choose Word’s **Edit • Paste Special** command. Select the **Paste** option (not **Paste link**), and choose the **Microsoft Excel Worksheet Object** format (see Figure 29-7). Click **OK**, and the range appears in the Word document.

The pasted object is not a standard Word table. For example, you can’t select or format individual cells in the table. Furthermore, it’s not linked to the Excel source range. If you change a value in the Excel worksheet, the change does not appear in the embedded object in the Word document.

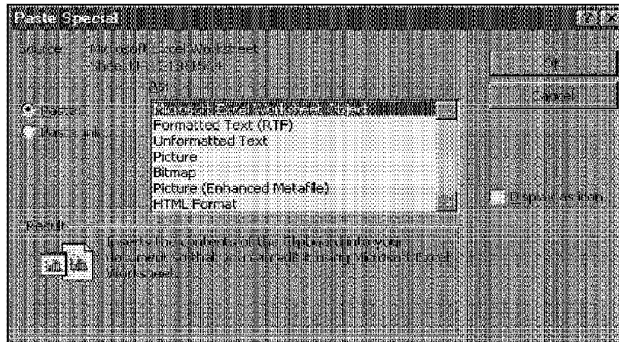


Figure 29-7: This operation embeds an Excel object in a Word document.

If you double-click the object, however, you notice something unusual: Word's menus and toolbars change to those used by Excel. In addition, the embedded object appears with Excel's familiar row and column borders. In other words, you can edit this object *in place* by using Excel's commands. Figure 29-8 shows how this looks. To return to Word, just click anywhere in the Word document.

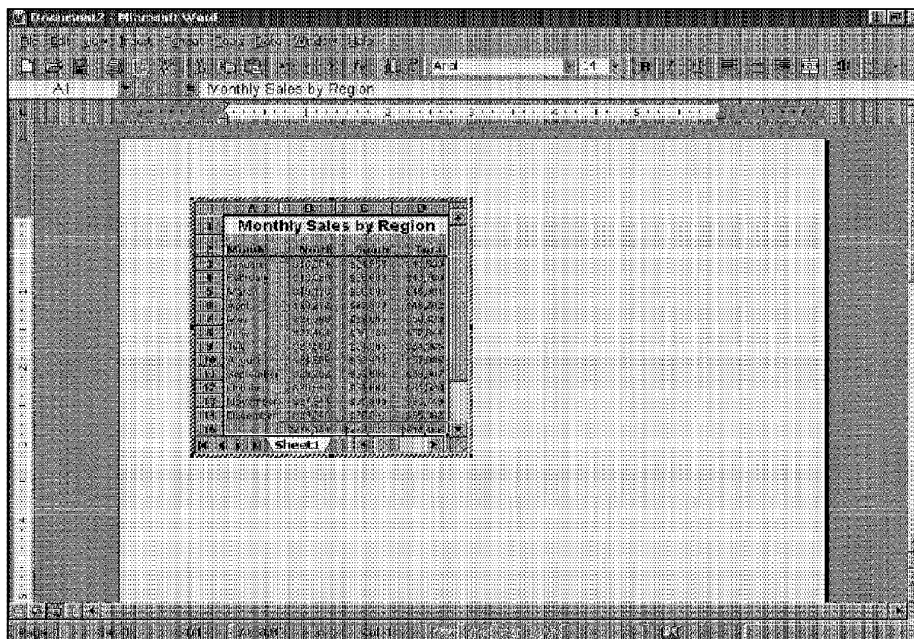


Figure 29-8: Double-clicking the embedded Excel object enables you to edit it in place. Note that Word now displays Excel's menus and toolbars.

Remember that no link is involved here. If you make changes to the embedded object in Word, these changes do not appear in the original Excel worksheet. The embedded object is completely independent from the original source.

Using this technique, you have access to all of Excel's features while you are still in Word. Microsoft's ultimate goal is to enable users to focus on their documents— not on the application that produces the document.



Tip

You can accomplish the embedding previously described by selecting the range in Excel and then dragging it to your Word document. In fact, you can use the Windows desktop as an intermediary storage location. For example, you can drag a range from Excel to the desktop and create a *scrap*. Then, you can drag this scrap into your Word document. The result is an embedded Excel object.

Creating a New Excel Object in Word

The preceding example embeds a range from an existing Excel worksheet into a Word document. This section demonstrates how to create a new (empty) Excel object in Word. This may be useful if you're creating a report and need to insert a table of values that doesn't exist in a worksheet. You *could* insert a normal Word table, but you can take advantage of Excel's formulas and functions to make this task much easier.

To create a new Excel object in a Word document, choose Insert • Object in Word. Word responds with the Object dialog box, shown in Figure 29-9. The Create New tab lists the types of objects that you can create (the contents of the list depends on the applications that you have installed on your system). Choose the Microsoft Excel Worksheet option and click OK.

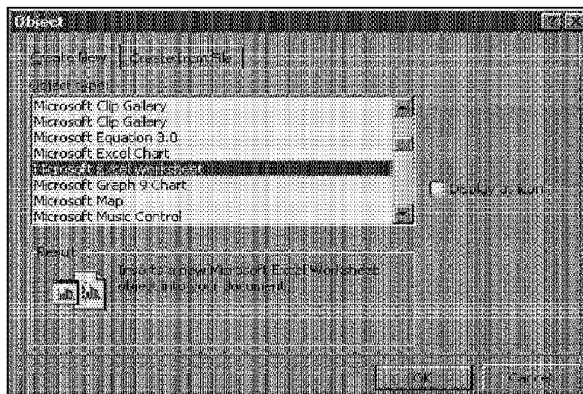


Figure 29-9: Word's Object dialog box enables you to create a new object.

Word inserts an empty Excel worksheet object into the document and activates it for you, as shown in Figure 29-10. You have full access to Excel commands, so you can enter whatever you want into the worksheet object. After you finish, click anywhere in the Word document. You can, of course, double-click this object at any time to make changes or additions.

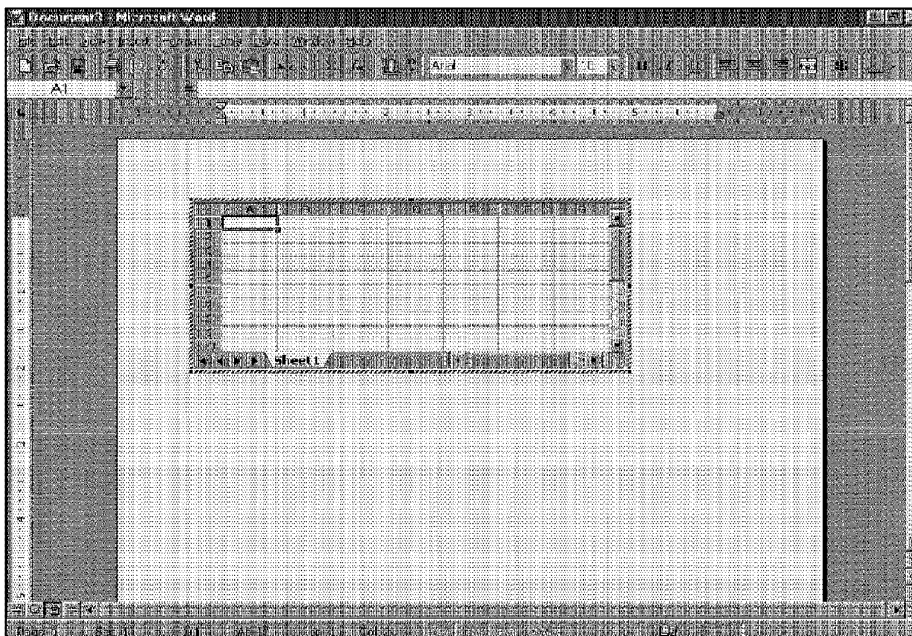


Figure 29-10: Word created an empty Excel worksheet object.

You can change the size of the object while it's activated by dragging any of the sizing handles that appear on the borders of the object. You also can crop the object, so that when it isn't activated, the object displays only cells that contain information. To crop an object in Word, select the object so that you can see sizing handles. Then, display Word's Picture toolbar (right-click any toolbar button and choose Picture). Click the Cropping tool (it looks like a pair of plus signs) and then drag any sizing handle on the object.



Even if you crop an Excel worksheet object in Word, when you double-click the object, you have access to all rows and columns in Excel. Cropping changes only the *displayed* area of the object.

Embedding an Existing Workbook in Word

Yet another option is to embed an existing workbook into a Word document. Use Word's Insert • Object command. In the Object dialog box, click the tab labeled Create from File (see Figure 29-11). Click the Browse button and locate the Excel workbook that you want to embed.

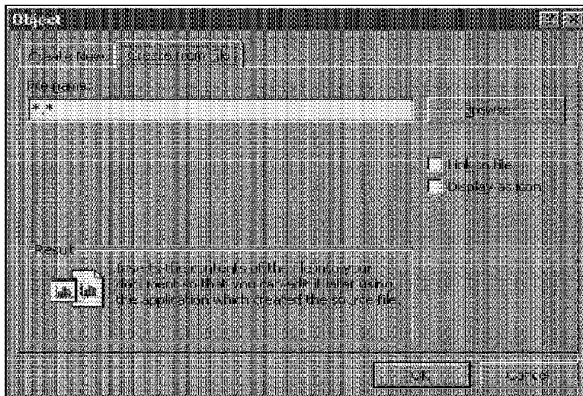


Figure 29-11: This dialog box enables you to locate a file to embed in the active document.

When you use this technique, you embed a *copy* of the selected workbook in the Word document. You can either use it as is or double-click it to make changes. Note that any changes that you make to this copy of the document are not reflected in the original workbook.

Embedding Objects in an Excel Worksheet

The preceding examples involve embedding Excel objects in a Word document. The same procedures can be used to embed other objects into an Excel worksheet.

For example, if you have an Excel workbook that requires a great amount of explanatory text, you have several choices:

- You can enter the text into cells. This is tedious and doesn't allow much formatting.
- You can use a text box. This is a good alternative, but it doesn't offer many formatting features.
- You can embed a Word document in your worksheet. This gives you full access to all of Word's formatting features.

To embed an empty Word document into an Excel worksheet, choose Excel's **Insert • Object** command. In the **Object** dialog box, click the **Create New** tab and select **Microsoft Word Document** from the **Object type** list.

The result is a blank Word document, activated and ready for you to enter text. Notice that Word's menus and toolbars replace Excel's menus and toolbars. You can resize the document as you like, and the words wrap accordingly. Figure 29-12 shows an example of a Word document embedded in an Excel worksheet.

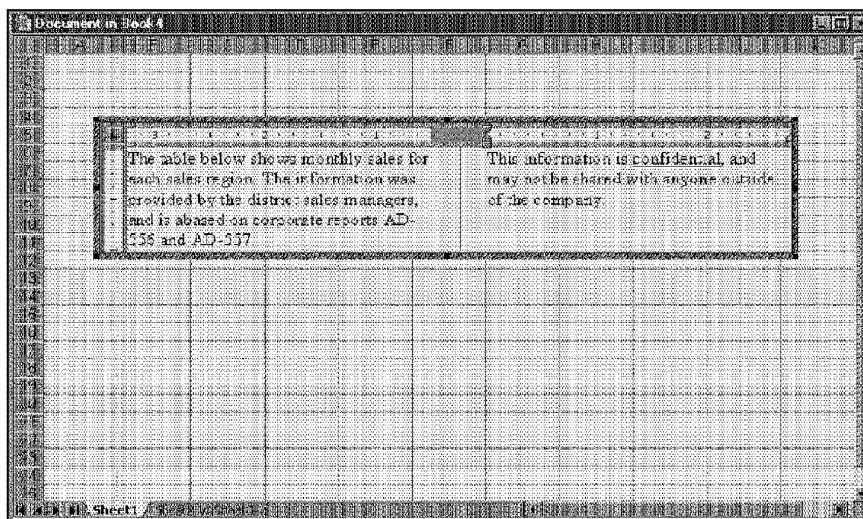


Figure 29-12: A Word document that is embedded in an Excel worksheet.

You can embed many other types of objects, including audio clips, video clips, MIDI sequences, and even an entire Microsoft PowerPoint presentation.

When you embed a video clip, Excel doesn't store the actual video clip file in the Excel document. Rather, Excel stores a pointer to the original file. If, for some reason, you want to embed the complete video clip file, you can use the **Object Packager** application. Be aware, however, that video clip files are typically quite large, and opening and saving the workbook will take a lot of time.

Microsoft Office includes a few additional applications that you may find useful. These all can be embedded in Excel documents:

- **Microsoft Equation:** Create equations, such as the one shown in Figure 29-13.
- **Microsoft WordArt:** Modify text in some interesting ways, as in Figure 29-14.
- **MS Organization Chart:** Create attractive organizational charts, as shown in Figure 29-15.

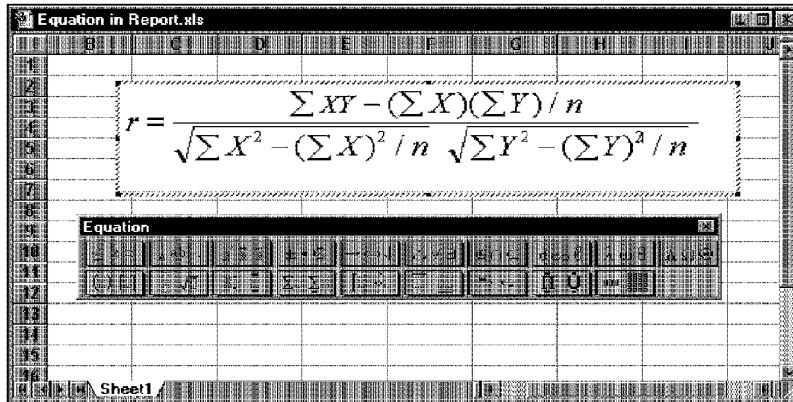


Figure 29-13: This object was created with Microsoft Equation.

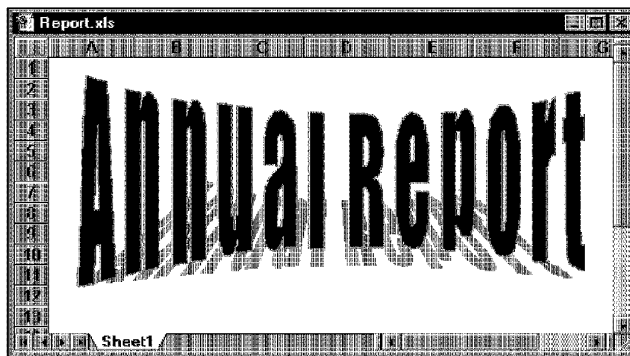


Figure 29-14: An example of Microsoft WordArt.

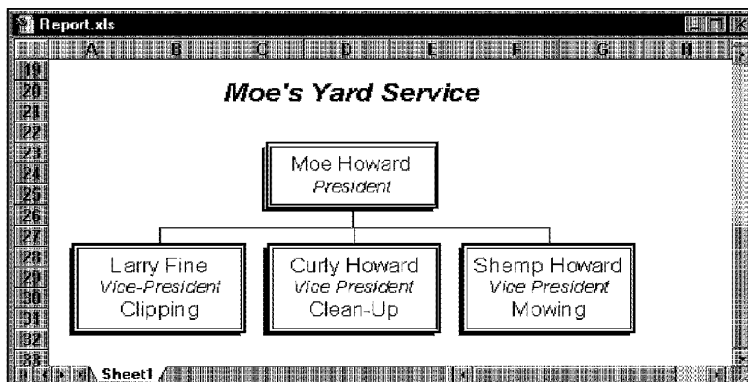


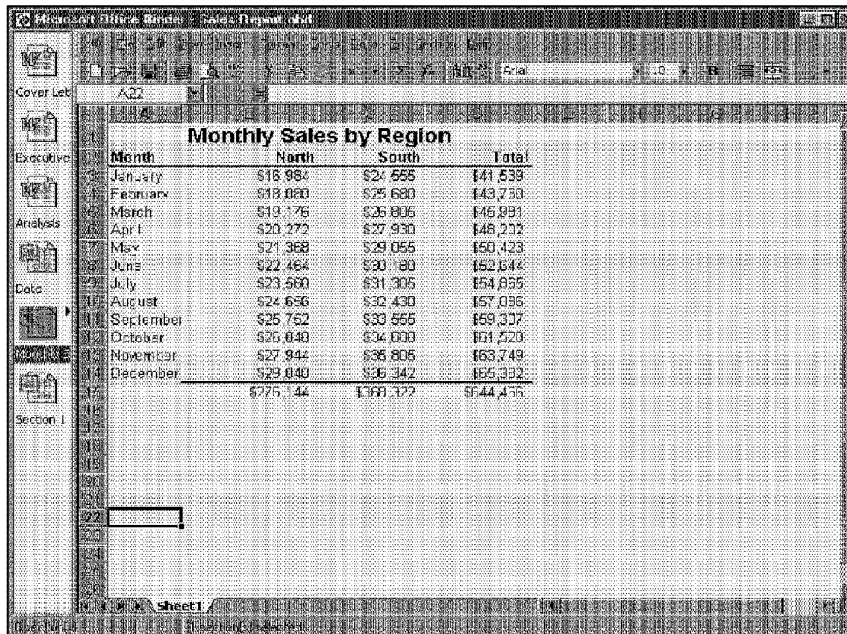
Figure 29-15: An example of an embedded organizational chart.

Using Office Binders

If you have Microsoft Office installed, you may take advantage of its binder feature. A *binder* is a container that can hold documents from different applications: Excel, Word, and PowerPoint.

You may find that a binder is useful when you are working on a project that involves documents from different applications. For example, you may be preparing a sales presentation that uses charts and tables from Excel, reports and memos from Word, and slides prepared with PowerPoint. You can store all the information in a single file. And, when you print the entire binder, pages are numbered sequentially.

To use a binder, start the Binder application, and an empty binder appears. You then can add existing documents to the binder or create new documents in the binder. Figure 29-16 shows a binder that contains Word, Excel, and PowerPoint documents. Consult the online Help for complete details on using this application.



The screenshot shows a Microsoft Office Binder window with a table titled "Monthly Sales by Region". The table has four columns: Month, North, South, and Total. The rows list the months from January to December, with a total row at the bottom. The data is as follows:

Month	North	South	Total
January	\$18,984	\$24,555	\$43,539
February	\$18,000	\$25,680	\$43,730
March	\$19,176	\$26,806	\$45,991
April	\$20,272	\$27,930	\$48,232
May	\$21,368	\$29,055	\$50,423
June	\$22,464	\$30,180	\$52,644
July	\$23,560	\$31,305	\$54,865
August	\$24,656	\$32,430	\$57,086
September	\$25,752	\$33,555	\$59,327
October	\$26,848	\$34,680	\$61,528
November	\$27,944	\$35,805	\$63,749
December	\$29,040	\$36,930	\$65,970
	\$275,144	\$369,322	\$644,466

Figure 29-16: An Office binder can hold documents that are produced by different applications.



You may need to rerun Office setup if Binder isn't installed on your computer. You'll find it under a category called Office Tools.

Summary

This chapter describes techniques that enable you to use data from other applications. These techniques include standard copy-and-paste options using the Windows and Office Clipboards, dynamic linking between applications, and embedding objects. This chapter concludes with a note on Microsoft Office's binder application, which enables you to work with documents that are produced by different applications.

• •

Excel and the Internet

Chances are, you're already involved in the Internet in some way. This technology seems to have taken the world by storm. The World Wide Web (WWW) is probably the most exciting thing happening these days in the world of computing. In fact, the Web reaches well beyond the computer community and is a pervasive force in our lives. It's now quite common to see Web site addresses listed in TV commercials, in magazine ads, and even on billboards.

The applications in Microsoft Office 2000—including Excel—have all been revamped to put them on a better footing with the Internet. This chapter provides an introduction to the Internet (for those who have yet to discover this resource) and discusses the Internet features that are available in Excel 2000.

What Is the Internet?

The *Internet*, in a nutshell, is a collection of computers that are located all around the world. These computers are all connected to each other, and they can pass information back and forth. Strange as it may seem, the Internet is essentially a non-commercial system, and no single entity “runs” the Internet.

Most people don't think of the Internet as a collection of computers. Rather, the Internet is a *resource* that contains information—and you use a computer to access that information. The computers that are connected to the Internet simply do the grunt work of passing the information from point A (which could be a computer in Hamburg, Germany) to point B (which could be the computer in your cubicle).

30

CHAPTER

.....

In This Chapter

What's Available on the Internet

Excel Newsgroups

Excel Mailing Lists

Internet Tools

.....

Internet Terminology for Newcomers

If you're just starting to explore the Internet, you'll encounter many new terms (many of which are acronyms). The following is a list of a few common Internet terms and their definitions:

- **Browser:** Software that is designed to download HTML documents, interpret them, and display their contents. You can also use a browser to download files from an FTP site. The two leading Web browsers are Microsoft Internet Explorer and Netscape Navigator.
- **Download:** To transfer a file from another computer to your computer.
- **E-mail:** A method of sending messages to others electronically. You may be able to send and receive e-mail only within your company, or you may be able to send and receive e-mail all over the world by using the Internet.
- **FTP:** An acronym for *File Transfer Protocol*. This is one method by which a file is transferred from one computer to another.
- **FTP site:** An area of a computer that contains files that can be downloaded. For example, Microsoft maintains several FTP sites that have files that you can download.
- **HTML document:** A computer file that contains information that is viewable in a browser. The file includes embedded "tags" that describe how the information is displayed and formatted. Browser software is designed to interpret these tags and display the information. Sometimes known as a *Web document* or a *Web page*.
- **HTTP:** An acronym for *Hypertext Transfer Protocol*. This is the method by which documents are transferred over the WWW.
- **Hyperlink:** A clickable object (or text) that opens another document. Most Web pages include hyperlinks, to allow the user to jump to another topic or Web site.
- **Internet:** A network of computers throughout the world that can communicate with each other and pass information back and forth.
- **Intranet:** A company-wide network of computers that uses Internet protocols to allow access to information. An intranet can be accessed only by users who have permission.
- **URL:** An acronym for *Uniform Resource Locator*. A URL uniquely describes an Internet resource, such as a WWW document or a file. For example, the URL for the opening page of Microsoft's Web site is `http://www.microsoft.com`.
- **Web site:** A collection of HTML documents and other files located on a particular computer. The files on a Web site are available to anyone in the world. For example, Microsoft maintains a Web site that contains information about its products, technical support, and other resources.
- **WWW:** The World Wide Web, which is a part of the Internet that supports the transfer of information between computers throughout the world.

What's Available on the Internet?

The amount and variety of information that's available on the Internet is simply mind-boggling. You can think of virtually any topic in the world, and an excellent chance exists that at least some information on that topic can be found on the Internet. Not unexpectedly, computer-related information is especially abundant.

So, where do you get this information? The following are the four primary sources for information on the Internet:

- **Web sites:** The Web has rapidly become the most popular part of the Internet. Hundreds of thousands of Web sites are available that you can access with your Web browser software. For example, my own Web site (The Spreadsheet Page) has the following URL: <http://www.j-walk.com/ss/>
- **FTP sites:** These are computers that have files available for download. You can download these files by using Web browser software or other software that is designed specifically to download files from FTP sites. The following is the URL for Microsoft's FTP site: <ftp://ftp.microsoft.com>
- **Newsgroups:** These are essentially electronic bulletin boards. People post messages or questions, and others respond to the messages or answer their questions. Thousands of newsgroups are available for just about any topic that you can think of. You need special "news reader" software to read or post messages to a newsgroup (although most Web browsers also include this feature). For more information, see the sidebar "Excel Newsgroups."
- **Mailing lists:** If you have access to Internet e-mail, you can subscribe to any of several thousand mailing lists that address a broad array of topics. Subscribers send e-mail to the mailing list, and then every other subscriber to the list receives that e-mail. There are two popular mailing lists that deal with Excel (refer to the "Excel Mailing Lists" sidebar for details).

How Do You Get on the Internet?

You can access the Internet in a number of ways. Here are some of the most common ways:

- **Through your company:** Your company may already be connected to the Internet. If so, just fire up your Web browser and you're there!
- **Through an Internet Service Provider (ISP):** Most communities have several companies that can set up an Internet account for you. For a small monthly fee (usually around \$20) you can have unlimited (or almost unlimited) access to the Internet. All that's required on your part is a computer, a modem, and a phone line.
- **Through an online service:** If you subscribe to any of the following online services, you can access the Internet through that service: America Online, CompuServe, Microsoft Network, or Prodigy.

Excel Newsgroups

Newsgroups are perhaps the best source for help with Excel. Typically, questions posed on a newsgroup are answered within 24 hours—assuming, of course, that the question is asked in a manner that makes others want to reply. The following is a list of newsgroups that deal with Excel:

- `comp.apps.spreadsheets` Covers all spreadsheets, but about 90 percent of the posts deal with Excel.
- `microsoft.public.excel.programming` Covers Excel programming issues, including VBA and XLM macros.
- `microsoft.public.excel.123quattro` Covers issues concerning conversion of 1-2-3 or Quattro Pro files to Excel.
- `microsoft.public.excel.worksheet.functions` Covers worksheet functions.
- `microsoft.public.excel.charting` Covers topics related to charts.
- `microsoft.public.excel.printing` Covers topics that deal with printing.
- `microsoft.public.excel.queryDAO` Discussion area about using the Microsoft Query and Data Access Objects (DAO) in Excel.
- `microsoft.public.excel.datamap` Covers the Data Map feature in Excel.
- `microsoft.public.excel.crashesGPFs` Covers General Protection Faults and other system failures.
- `microsoft.public.excel.misc` A catch-all group for topics that do not fit one of the other categories.
- `microsoft.public.excel.links` Covers topics related to using links in Excel.
- `microsoft.public.excel.interopoleddde` Discussion area for Object Linking and Embedding (OLE), Dynamic Data Exchange (DDE), and other cross-application issues.
- `microsoft.public.excel.setup` Covers problems dealing with setup and installation of Excel.
- `microsoft.public.excel.templates` Discussion area for the Spreadsheet Solutions templates and other XLT files.

**Note**

If your ISP doesn't carry the `microsoft.public.excel.*` groups, you can access them directly from Microsoft's news server. You need to configure your newsreader software or Web browser to access Microsoft's news server, which is `msnews.microsoft.com`.

Excel Mailing Lists

If you like the idea of communicating with other Excel users, you may want to join one of the Excel mailing lists. You can read messages, questions, and answers posted by others and eventually contribute your own messages to the list. If you find that the amount of mail is overwhelming, it's easy to "unsubscribe."

The EXCEL-G mailing list. For Excel users of all levels. To subscribe to the list, send e-mail to the following: `LISTSERV@PEACH.EASE.LSOFT.COM`.

In the body of the message, enter the following:

```
SUB EXCEL-G YourFirstName YourLastName
```

You'll receive complete instructions via e-mail.

The EXCEL-L mailing list. Primarily for Excel developers who discuss more advanced topics. To subscribe to the list, send e-mail to the following: `LISTSERV@PEACH.EASE.LSOFT.COM`.

In the body of the message, enter the following:

```
SUB EXCEL-L YourFirstName YourLastName
```

You'll receive complete instructions via e-mail.

Where to Find Out More About the Internet

The best place to find out more about the Internet is—you guessed it—the Internet. A good starting place is the IDG Books Web site. To access it, open the following URL in your Web browser: <http://www.idgbooks.com>.

IDG Books Worldwide publishes numerous Internet books for users of all levels, and you can find these listed and described on the IDG Web site.

Excel's Internet Tools

The remainder of this chapter describes the Internet-related features available in Excel 2000. These features include:

- Using HTML as a native file format (instead of the XLS file format).
- Saving a worksheet as an interactive Web page.
- Using Excel's Web toolbar.
- Inserting hyperlinks into a worksheet.

- Creating and using Web queries.
- Scheduling and conducting online meetings.
- Creating discussion groups.

Using HTML As a Native File Format



Excel's standard file format is, of course, an XLS file. Excel 2000, however, has the ability to use HTML as a native file format. This means that you can create a workbook and save it in HTML format. Then, you can reopen the file without losing any information. In other words, your Excel-specific information (such as formulas, charts, pivot tables, and macros) survive the translation to HTML.

If you've used the "save as HTML" feature in Excel 97, you probably know that the HTML file that's created works fine in Web browsers — but if you reopen the file in Excel, all of your formulas (as well as other Excel-specific features) will be gone. With Excel 2000, this problem no longer exists, because the HTML file contains lots of proprietary tags that are ignored by browsers but that enable Excel to re-create the workbook.

To save a workbook in HTML format, select File • Save As. You'll see the familiar Save As dialog box — but with some new options (see Figure 30-1). In the field labeled Save as type, make sure Web Page (*.htm, *.html) is selected. Provide a filename, and click Save. To reopen the file, use the normal File • Open command.

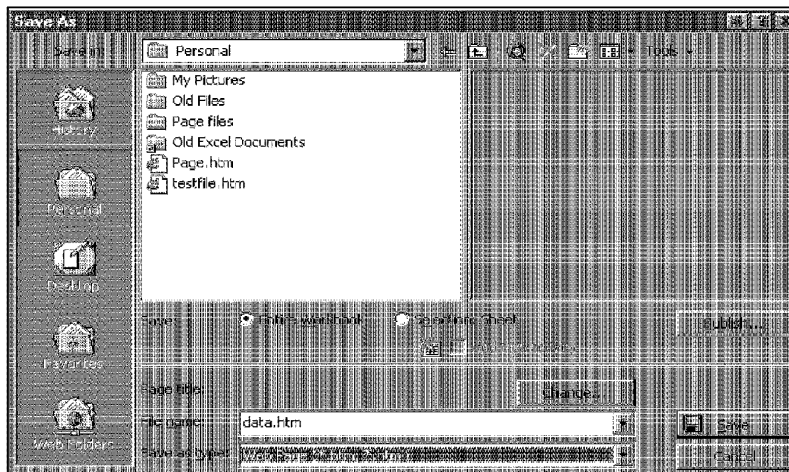


Figure 30-1: Use the Save As dialog box to save a workbook in HTML format.

**Caution**

Unless your workbook is very simple, saving it in HTML format generates additional “supporting” files, because the HTML file format can’t handle Excel-specific items, such as macros, charts, and pivot tables. The supporting files are stored in a separate subdirectory within the directory where you save the file. The directory name consists of the file’s name, followed by a space and the word “files.” Therefore, if you need to transfer the file to another computer, make sure that you also transfer the supporting files in the subdirectory.

If you save your work in HTML format, you should be aware of some additional options. Select **Tools • Options**, click the **General** tab, and then click the **Web Options** button. You’ll see the dialog box shown in Figure 30-2. Most of the time, the default settings work just fine. However, familiarizing yourself with the options available is worthwhile (these are described in the online Help). You can also access the Web Options dialog box from the **Tools** menu in the **Save As** dialog box.

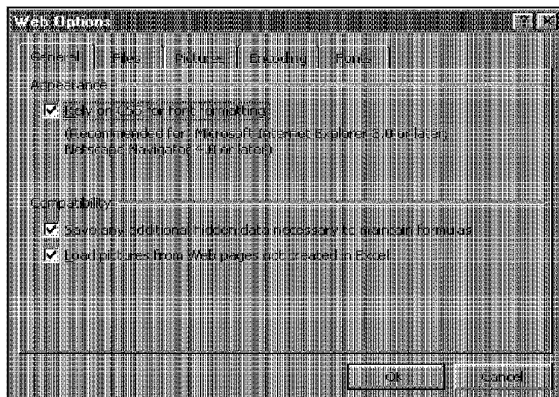


Figure 30-2: Use the Web Options dialog box to set various options for working with HTML files.

When you save a workbook in HTML format, by default, it will not be interactive when it’s opened in a browser. The browser displays a good rendition of the worksheet, but it’s essentially a “dead” workbook, because the user can’t change any cells. The next section describes how to save your Excel workbook in a way that provides interactivity within a Web browser.

Providing Interactivity in Your Web Documents

When you save an Excel workbook in HTML format, you can select an option that makes the file interactive within the browser. This means that the user can perform standard Excel operations directly in the browser. For example, the user can change cells or manipulate data in a pivot table. Saving an Excel file with interactivity is limited to a single sheet.



To take advantage of this interactivity, the user must have Office 2000 installed, or have a licensed copy of the Office Client Pak. The Office Client Pak consists of the ActiveX controls necessary to work with interactive Office documents in a Web browser. Currently, the only browser that supports this technology is Microsoft Internet Explorer.

Figure 30-3 shows an example of an Excel workbook displayed in Internet Explorer. The user can change the values, and the formulas display the calculated results.

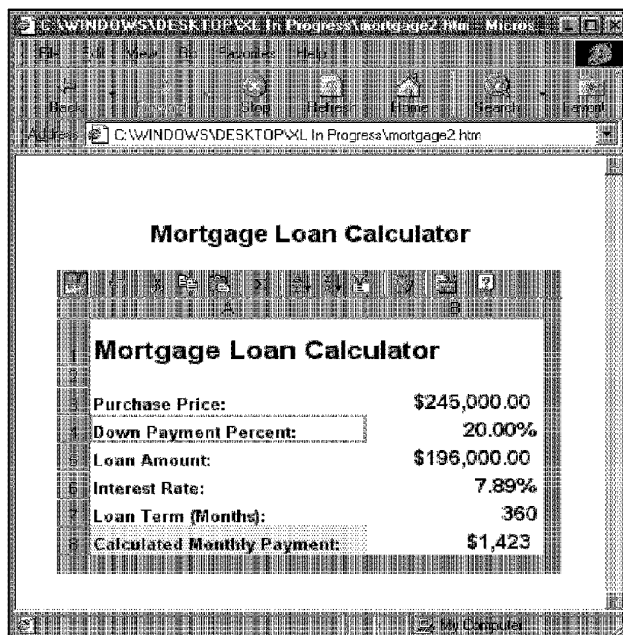


Figure 30-3: An interactive Excel workbook opened in Internet Explorer.

You need to understand that the interactivity is limited. For example, you can't execute macros when an interactive Excel file is displayed in a browser.

Using the Web Toolbar

Use the Web toolbar (shown in Figure 30-4) to move among files (Excel files and HTML documents); this is similar to using a Web browser. You can jump forward or backward among the workbooks and other files that you've visited, and add the ones that you may use frequently to a "favorites" list.



Figure 30-4: The Web toolbar.

Working with Hyperlinks

Hyperlinks are shortcuts that provide a quick way to jump to other workbooks and files. You can jump to files on your own computer, your network, and the Internet and Web.

Inserting a hyperlink

You can create hyperlinks from cell text or graphic objects, such as shapes and pictures. To create a text hyperlink, choose the **Insert • Hyperlink** command (or press **Ctrl+K**). Excel responds with the dialog box shown in Figure 30-5.

Select an icon in the **Link to** column that represents the type of hyperlink you want to create. Then, specify the location for the file that you want to link to. The dialog box will change, depending on the icon selected. Click **OK**, and Excel creates the hyperlink in the active cell.

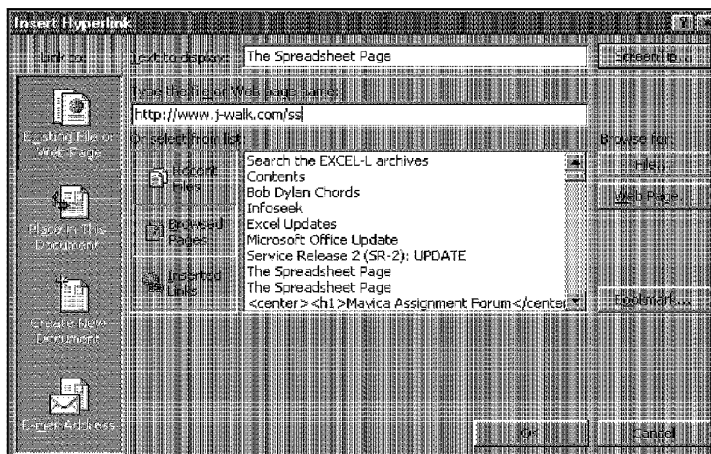


Figure 30-5: The Insert Hyperlink dialog box.

Adding a hyperlink to a graphic object works the same way. Add an object to your worksheet by using the Drawing toolbar. Select the object and then choose the **Insert • Hyperlink** command. Specify the required information as outlined in the previous paragraph.

Using hyperlinks

When you work with hyperlinks, remember that Excel attempts to mimic a Web browser. For example, when you click a hyperlink, the hyperlinked document replaces the current document — it takes on the same window size and position. The document that contains the hyperlink is hidden. You can use the Back and Forward buttons on the Web toolbar to activate the documents.

Web Queries

Excel enables you to pull in data contained in an HTML file by performing a Web query. The data is transferred to a worksheet, where you can manipulate it any way you like. You need to understand that performing a Web query does not actually open the HTML file in Excel.



The Web query feature is very similar to performing a normal database query (see Chapter 24). The only difference is that the data is coming from a Web page rather than a database file. Figure 30-6 shows a Web page that's a good candidate for a Web query.

The screenshot shows a browser window with the address bar containing `http://jwalk/ss/salesdata.htm`. The page title is "Corporate Sales Data (Confidential)". Below the title is a table with the following data:

Month	SalesRep	Type	UnitCost	Quantity	TotalSales
March	Wilson	New	175	5	875
March	Wilson	New	140	3	420
February	Franks	Existing	225	1	225
March	Wilson	New	125	5	625
January	Peterson	Existing	225	2	450
March	Sheldon	New	140	2	280
February	Peterson	Existing	225	6	1350
March	Jenkins	Existing	140	2	280
February	Sheldon	New	225	4	900
January	Wilson	New	140	4	560
January	Wilson	New	125	3	375
January	Sheldon	New	225	6	1350
February	Sheldon	New	175	5	875
January	Robinson	New	140	3	420

Figure 30-6: The table in this Web page will be brought into a worksheet as a Web query.

The best part about a Web query is that Excel remembers where the data came from. Therefore, after you create a Web query, you can “refresh” the query to pull in the most recent data.

To create a Web query, select **Data • Get External Data • New Web Query**. Excel displays the **New Web Query** dialog box, shown in Figure 30-7. In part 1, specify the HTML file, using the **Browse** button if you like. The HTML file can be on the Internet, a corporate intranet, or on a local or network drive. In part 2, select how much of the file you want to use. Most of the time, you’ll just want to bring in a particular table. In part 3, specify the type of formatting that you’d like to see. Click the **Advanced** button for some additional options — these options might be necessary if the data in the HTML file is not in the form of a table. Click **OK** and you get another dialog box asking where you want to place the data.

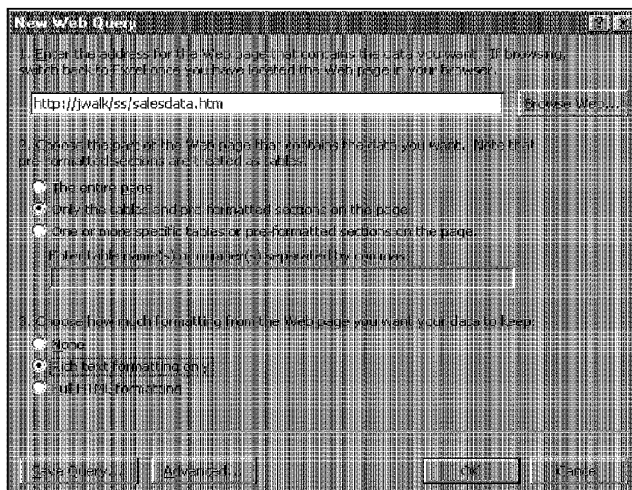


Figure 30-7: Use the **New Web Query** dialog box to specify the source of the data.

Figure 30-8 shows an Excel workbook, after performing a Web query.

Month	SalesRep	Type	UnitCost	Quantity	TotalSales
March	Wilson	New	175	5	875
March	Wilson	New	140	3	420
February	Frankie	Existing	225	1	225
March	Wilson	New	125	5	625
January	Peterson	Existing	225	2	450
March	Sheldon	New	140	2	280
February	Peterson	Existing	225	6	1350
March	Jenkins	Existing	140	2	280
February	Sheldon	New	225	4	900
January	Wilson	New	140	4	560
January	Wilson	New	125	3	375
January	Sheldon	New	225	6	1350
February	Sheldon	New	175	5	875
January	Robinson	New	140	3	420
February	Sheldon	New	125	2	250
March	Sheldon	New	140	6	840
March	Jenkins	Existing	225	3	675
January	Robinson	New	225	2	450
March	Sheldon	New	225	6	1350
February	Wilson	New	140	3	420
February	Robinson	New	140	3	420
February	Sheldon	New	140	5	700
March	Robinson	New	175	1	175

Figure 30-8: The data in this workbook resulted from a Web query.

After you create your Web query, you have some options. Activate any cell in the data range and select **Data • Get External Data • Data Range Properties**. Or, you can right-click and select the command from the shortcut menu. Either method displays the dialog box shown in Figure 30-9. Adjust the settings to your liking.

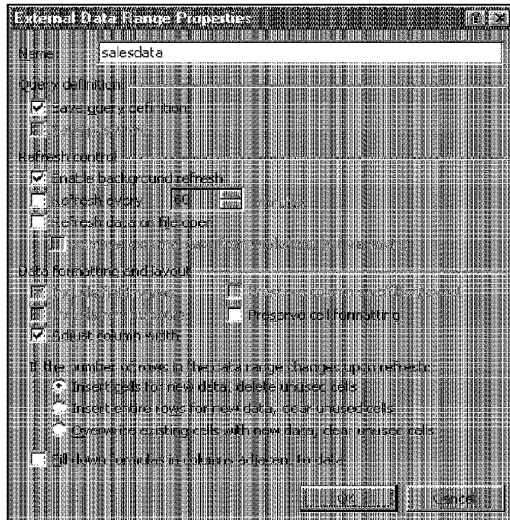


Figure 30-9: The External Data Range Properties dialog box provides you with some options regarding your Web query.

Summary

This chapter provides a brief introduction to the Internet and describes several Internet tools that are available in Excel. It explains how to use HTML as a native file format, use the Web toolbar, work with hyperlinks, and use Web queries.

• •

Making Your Worksheets Error-Free

The ultimate goal in developing a spreadsheet solution is to generate accurate results. For simple worksheets, this isn't difficult, and you can usually tell whether the results are correct. But when your worksheets are large or complex, ensuring accuracy becomes more difficult. This chapter provides you with tools and techniques to help you identify and correct errors.

Types of Worksheet Problems

Making a change in a worksheet — even a relatively minor change — may produce a ripple effect that introduces errors in other cells. For example, accidentally entering a value into a cell that formerly held a formula is all too easy to do. This can have a major impact on other formulas, and you may not discover the problem until long after you make the change. Or, you may *never* discover the problem.

An Excel worksheet can have many types of problems. Some problems — such as a formula that returns an error value — are immediately apparent. Other problems are more subtle. For example, if a formula was constructed using faulty logic, it may never return an error value — it simply returns the wrong values. If you're lucky, you can discover the problem and correct it.

Common problems that occur in worksheets are the following:

- Incorrect approach to a problem
- Faulty logic in a formula
- Formulas that return error values

31

CHAPTER

In This Chapter

Types of Worksheet Problems

Formula AutoCorrect

Tracing Cell Relationships

Other Auditing Tools

Spelling and Word-Related Options

Learning About an Unfamiliar Spreadsheet

- Circular references
- Spelling mistakes
- A worksheet is new to you, and you can't figure out how it works

Excel provides tools to help you identify and correct some of these problems. In the remaining sections, I discuss these tools along with others that I've developed.

Formula AutoCorrect

When you enter a formula that has a syntax error, Excel attempts to determine the problem and offers a suggested correction.

For example, if you enter the following formula (which has a syntax error), Excel displays the dialog box that is shown in Figure 31-1:

```
=SUM(A1:A12)/3B
```

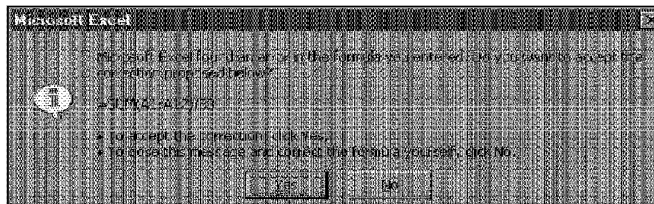


Figure 31-1: Excel can often offer a suggestion to correct a formula.



Be careful about accepting corrections for your formulas from Excel, because it doesn't always guess correctly. For example, I entered the following formula (which has mismatched parentheses):

```
=AVERAGE(SUM(A1:A12,SUM(B1:B12)))
```

Excel proposed the following correction to the formula:

```
=AVERAGE(SUM(A1:A12,SUM(B1:B12)))
```

You may be tempted to accept the suggestion without even thinking. In this case, the proposed formula is syntactically correct—but not what I intended.

Tracing Cell Relationships

Excel has several useful tools that can help you track down errors and logical flaws in your worksheets. This section discusses the following items:

- Go To Special dialog box
- Excel's built-in auditing tools

These tools are useful for debugging formulas. As you probably realize by now, the formulas in a worksheet can become complicated and refer (directly or indirectly) to hundreds or thousands of other cells. Trying to isolate a problem in a tangled web of formulas can be frustrating.

Before discussing these features, you need to be familiar with the following two concepts:

- **Cell precedents:** Applicable only to cells that contain a formula. A formula cell's precedents are all the cells that contribute to the formula's result. A *direct precedent* is a cell that you use directly in the formula. An *indirect precedent* is a cell that isn't used directly in the formula, but is used by a cell to which you refer in the formula.
- **Cell dependents:** Formula cells that depend on a particular cell. Again, the formula cell can be a direct dependent or an indirect dependent.

Often, identifying cell precedents for a formula cell sheds light on why the formula isn't working correctly. On the other hand, knowing which formula cells depend on a particular cell is often helpful. For example, if you're about to delete a formula, you may want to check whether it has any dependents.

The Go To Special Dialog Box

The Go To Special dialog box can be useful, because it enables you to specify the type of cells that you want Excel to select. To display this dialog box, choose Edit • Go To (or press F5). The Go To dialog box appears. Click the Special button, which displays the Go To Special dialog box, as shown in Figure 31-2.

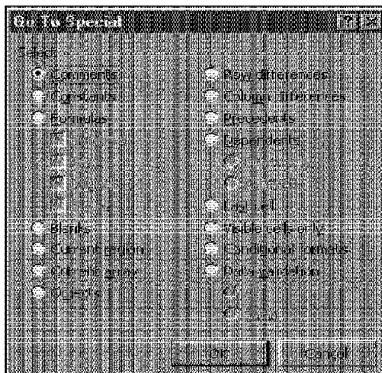


Figure 31-2: The Go To Special dialog box.

If you select a range before choosing **Edit • Go To**, the command looks only at the selected cells. If only a single cell is selected, the command operates on the entire worksheet.

You can use this dialog box to select cells of a certain type—which can often be helpful in identifying errors. For example, if you choose the **Formulas** option, Excel selects all the cells that contain a formula. If you zoom the worksheet out to a small size, you can get a good idea of the worksheet's organization (see Figure 31-3). It may also help you spot a common error: a formula that you overwrote with a value. If you find a cell that's not selected amid a group of selected formula cells, chances are good that the cell formerly contained a formula that has been replaced by a value.

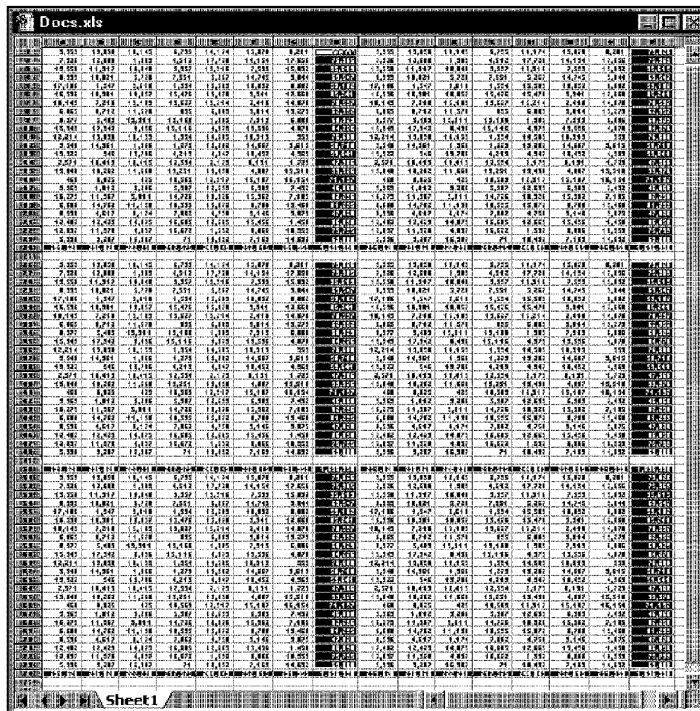


Figure 31-3: Zooming out and selecting all formula cells can give you a good overview of how the worksheet is designed.

You can also use the **Go To Special** dialog box to identify cell precedents and dependents. In this case, Excel selects all cells that qualify. In either case, you can choose whether to display direct or all levels.

Excel has shortcut keys that you can use to select precedents and dependents. These are listed in Table 31-1.

Table 31-1
Shortcut Keys to Select Precedents and Dependents

<i>Key Combination</i>	<i>What It Selects</i>
Ctrl+[Direct precedents
Ctrl+Shift+[All precedents
Ctrl+]	Direct dependents
Ctrl+Shift+]	All dependents

You also can select a formula cell's direct dependents by double-clicking the cell. This technique, however, works only when you turn off the Edit directly in the cell option on the Edit tab of the Options dialog box.

Excel's Auditing Tools

Excel provides a set of interactive auditing tools that you may find helpful. Access these tools either by selecting Tools • Auditing (which results in a submenu with additional choices) or by using the Auditing toolbar, shown in Figure 31-4.



Figure 31-4: The Auditing toolbar.

Pay Attention to the Colors

When you edit a cell that contains a formula, Excel color-codes the cell and range references in the formula. Excel also outlines the cells and ranges used in the formula by using corresponding colors. Therefore, you can see at a glance the cells that are used in the formula.

You can also manipulate the colored outline to change the cell or range reference. To change the references that are used, drag the outline's border or drag the outline's fill handle (at the lower-right corner of the outline).

The tools on the Auditing toolbar, from left to right, are as follows:

- **Trace Precedents:** Draws arrows to indicate a formula cell's precedents. Click this multiple times to see additional levels of precedents.
- **Remove Precedent Arrows:** Removes the most recently placed set of precedent arrows.

- **Trace Dependents:** Draws arrows to indicate a cell's dependents. Click this multiple times to see additional levels of dependents.
- **Remove Dependent Arrows:** Removes the most recently placed set of dependent arrows.
- **Remove All Arrows:** Removes all precedent and dependent arrows from the worksheet.
- **Trace Error:** Draws arrows from a cell that contains an error to the cells that may have caused the error.
- **New Comment:** Inserts a comment for the active cell. This really doesn't have much to do with auditing. It lets you attach a comment to a cell.
- **Circle Invalid Data:** Draws a circle around all the cells that contain invalid data. This applies only to cells that have validation criteria specified with the Data • Validation command.
- **Clear Validation Circles:** Removes the circles that are drawn around cells that contain invalid data.

These tools can identify precedents and dependents by drawing arrows (known as *cell tracers*) on the worksheet, as shown in Figure 31-5. In this case, cell G11 was selected and then the Trace Precedents toolbar button was clicked. Excel drew lines to identify the cells used by the formula in G11 (direct precedents).

The screenshot shows an Excel spreadsheet with the following data:

Sales Rep	Last Month	This Month	Change	Pct. Change	Met Goal?	Com-mission
Murray	101,233	96,744	(2,489)	-2.5%	FALSE	5,431
Kruckles	120,933	134,544	13,611	11.3%	FALSE	7,400
Lefty	112,344	134,887	22,543	20.1%	TRUE	8,768
Lucky	130,933	151,745	20,812	15.9%	TRUE	9,863
Scarface	150,932	140,778	(10,154)	-6.7%	FALSE	7,743
Totals	616,375	660,698	44,323	7.2%		39,205

Below the table, the following information is displayed:

Commission Rate: 5.50% Normal Commission Rate
 Sales Goal: 15% Improvement From Prior Month
 Bonus Rate: 6.50% Paid if Sales Goal is Attained

Average Commission Rate: 5.93%

Figure 31-5: Excel draws lines to indicate a cell's precedents.

Figure 31-6 shows what happens when the Trace Precedents button is clicked again. This time, Excel adds more lines to show the indirect precedents. The result is a graphical representation of the cells that are used (directly or indirectly) by the formula in cell G11.

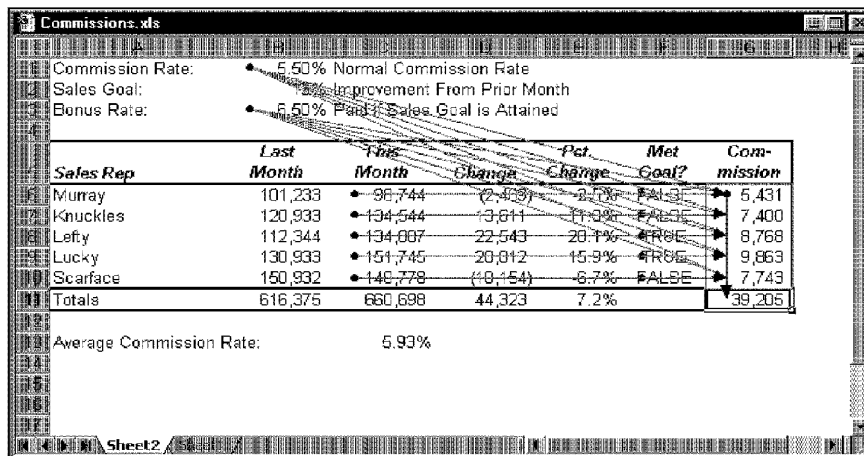


Figure 31-6: Excel draws more lines to indicate the indirect precedents.



This type of interactive tracing is often more revealing when the worksheet is zoomed out to display a larger area.

The best way to learn about these tools is to use them. Start with a worksheet that has formulas and experiment with the various buttons on the Auditing toolbar.

Tracing Error Values

The Trace Error button on the Auditing toolbar helps you to identify the cell that is causing an error value to appear. Often, an error in one cell is the result of an error in a precedent cell. Activate a cell that contains an error, and click the Trace Error button. Excel draws arrows to indicate the error source.

Table 31-2 lists the types of error values that may appear in a cell that has a formula. The Trace Error button works with all of these errors.

Table 31-2
Excel Error Values

<i>Error Value</i>	<i>Explanation</i>
#DIV/0!	The formula is trying to divide by zero (an operation that's not allowed on this planet). This also occurs when the formula attempts to divide by a cell that is empty.
#NAME?	The formula uses a name that Excel doesn't recognize. This can happen if you delete a name that's used in the formula or if you have unmatched quotation marks when using text.
#N/A	The formula refers to an empty cell range.
#NULL!	The formula uses an intersection of two ranges that do not intersect (this concept is described later in the chapter).
#NUM!	A problem with a value exists – for example, you specified a negative number where a positive number is expected.
#REF!	The formula refers to a cell that is not valid. This can happen if the cell has been deleted from the worksheet.
#VALUE!	The formula includes an argument or operand of the wrong type.

Circular References

A *circular reference* occurs when a formula refers to its own cell— either directly or indirectly. Usually, this is the result of an error (although some circular references are intentional). When a worksheet has a circular reference, Excel displays the cell reference in the status bar.



Refer to the discussion of circular references in Chapter 9.

Other Auditing Tools

The registered version of the Power Utility Pak includes a utility named Auditing Tools. The dialog box for this utility is shown in Figure 31-7.



Figure 31-7: The Worksheet Auditing dialog box from the Power Utility Pak.

This utility works with the active worksheet and can generate any or all of the following items:

- **Worksheet map:** A color-coded graphical map of the worksheet that shows the type of contents for each cell—value, text, formula, logical value, or error. See Figure 31-8.
- **Formula list:** A list of all formulas in the worksheet, including their current values.
- **Summary report:** An informative report that includes details about the worksheet, the workbook that it's in, and a list of all defined names.

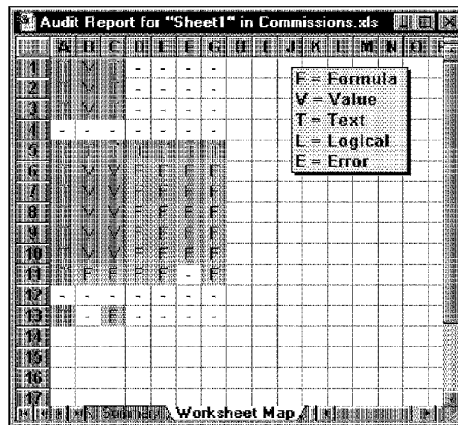


Figure 31-8: This worksheet map was produced by the Auditing Tools utility from the Power Utility Pak.



You can find the shareware version of the Power Utility Pak on this book's CD-ROM. Owners of this book can purchase the Power Utility Pak at a significant discount. Use the coupon in the back of the book to order your copy.

Spelling and Word-Related Options

Excel includes several handy tools to help you with the non-numeric problems—those related to spelling and words.

Spell Checking

If you use a word processing program, you probably run its spelling checker before printing an important document. Spelling mistakes can be just as embarrassing when they appear in a spreadsheet. Fortunately, Microsoft includes a spelling checker with Excel. You can access the spelling checker by using any of these methods:

- Select Tools • Spelling
- Click the Spelling button on the Standard toolbar
- Press F7

The result of using any one of these methods is the Spelling dialog box that is shown in Figure 31-9.

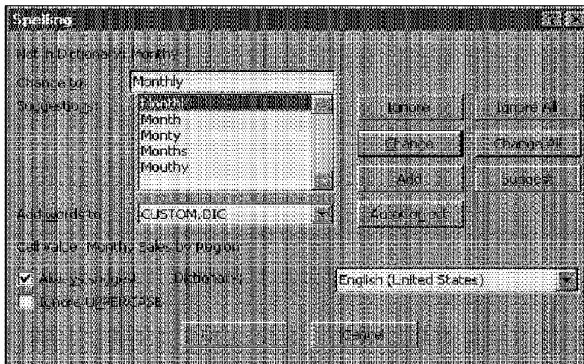


Figure 31-9: The Spelling dialog box.

The extent of the spell checking depends on what you selected before you opened the Spelling dialog box. If you selected a single cell, Excel checks the entire worksheet, including cell contents, notes, text in graphic objects and charts, and page headers and footers. Even the contents of hidden rows and columns are checked. If you select a range of cells, Excel checks only that range. If you select a group of characters in the formula bar, Excel checks only those characters.

The Spelling dialog box works similarly to other spelling checkers with which you may be familiar. If Excel encounters a word that isn't in the current dictionary or is misspelled, it offers a list of suggestions. You can respond by clicking one of the following buttons:

- **Ignore:** Ignores the word and continues the spell check.
- **Ignore All:** Ignores the word and all subsequent occurrences of it.
- **Change:** Changes the word to the selected word in the Change to edit box.

- **Change All:** Changes the word to the selected word in the Change to edit box and changes all subsequent occurrences of it without asking.
- **Add:** Adds the word to the dictionary.
- **Suggest:** Displays a list of replacement words. This button is grayed if the Always suggest check box is checked.
- **AutoCorrect:** Adds the misspelled word and its correct spelling to the list:

```
=SUM(A1:A12)/3B
```

Using AutoCorrect

AutoCorrect is a handy feature that automatically corrects common typing mistakes. You also can add words to the list that Excel corrects automatically. The AutoCorrect dialog box appears in Figure 31-10. You access this feature by choosing Tools • AutoCorrect.

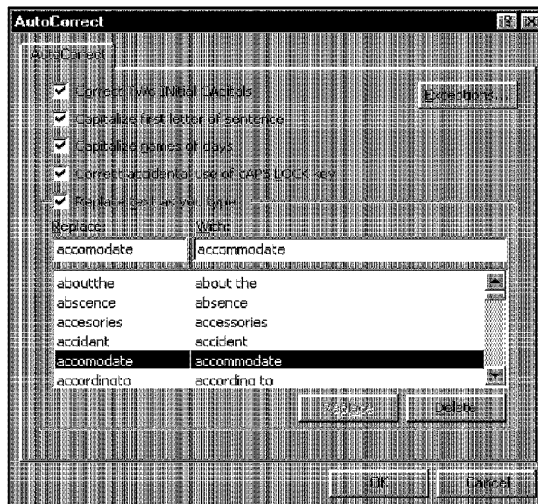


Figure 31-10: The AutoCorrect dialog box.

This dialog box has several options:

- **Correct Two Initial Capitals:** Automatically corrects words with two initial uppercase letters. For example, *Budget* is converted to *Budget*. This is a common mistake among fast typists. You can click on the Exceptions button to specify a list of exceptions to this rule. For example, my company name is *JWalk and Associates*, so I created an exception for *JWalk*.
- **Capitalize first letter of sentence:** Capitalizes the first letter in a sentence.

- **Capitalize names of days:** Capitalizes the days of the week. If you enter *monday*, Excel converts it to *Monday*.
- **Correct accidental use of cAPS LOCK key:** Corrects errors caused if you accidentally hit the CapsLock key while typing.
- **Replace text as you type:** AutoCorrect automatically changes incorrect words as you type them.

Excel includes a long list of AutoCorrect entries for commonly misspelled words. In addition, it has AutoCorrect entries for some symbols. For example, (*c*) is replaced with © and (*r*) is replaced with ® You can also add your own AutoCorrect entries. For example, if you find that you frequently misspell the word *January* as *Janruary*, you can create an AutoCorrect entry so that it's changed automatically. To create a new AutoCorrect entry, enter the misspelled word in the Replace box and the correctly spelled word in the With box. As I noted previously, you also can do this in the Spelling dialog box.

You also can use the AutoCorrect feature to create shortcuts for commonly used words or phrases. For example, if you work for a company named Consolidated Data Processing Corporation, you can create an AutoCorrect entry for an abbreviation, such as *cdp*. Then, whenever you type *cdp*, Excel automatically changes it to *Consolidated Data Processing Corporation*.

Using AutoComplete

AutoComplete automatically finishes a word as soon as Excel recognizes it. For Excel to recognize the word, it must appear elsewhere in the same column. This feature is most useful when you're entering a list that contains repeated text in a column. For example, assume that you're entering customer data in a list, and one of the fields is City. Whenever you start typing, Excel searches the other entries in the column. If it finds a match, it completes the entry for you. Press Enter to accept it. If Excel guesses incorrectly, keep typing to ignore the suggestion.

If AutoComplete isn't working, select Tools • Options, click on the Edit tab, and check the box labeled Enable AutoComplete for cell values.

You also can display a list of all items in a column by right-clicking and choosing Pick from list from the shortcut menu. Excel then displays a list box of all entries that are in the column (see Figure 31-11). Click on the one that you want, and Excel enters it into the cell for you.

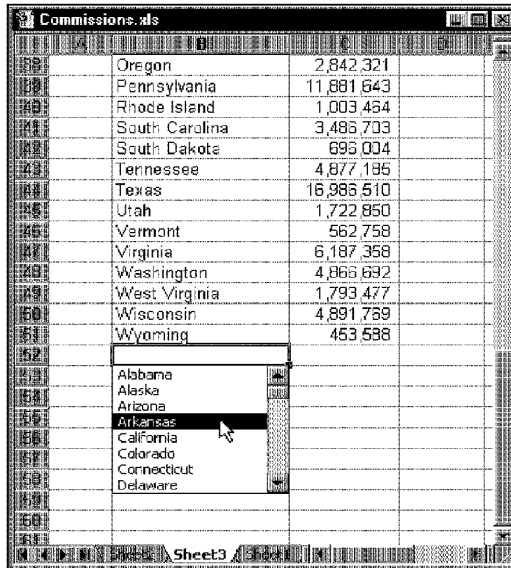


Figure 31-11: Choosing the Pick from list option from the shortcut menu gives you a list of entries from which to choose.

Learning About an Unfamiliar Spreadsheet

When you develop a workbook yourself, you have a thorough understanding of how it's put together. But if you receive an unfamiliar workbook from someone, it may be difficult to understand how it all fits together — especially if it's large.

First, identify the bottom-line cell or cells. Often, a worksheet is designed to produce results in a single cell or in a range of cells. After you identify this cell or range, you should be able to use the cell-tracing techniques described earlier in this chapter to determine the cell relationships.

Although every worksheet is different, a few techniques can help you become familiar with an unfamiliar workbook. I discuss these techniques in the following sections.

Zooming Out for the Big Picture

I find that it's often helpful to use Excel's zoom feature to zoom out to get an overview of the worksheet's layout. You can select View • Full Screen to see even more of the worksheet. When a workbook is zoomed out, you can use all of the normal

commands. For example, you can use the **Edit • Go To** command to select a name range. Or, you can use the options that are available in the **Go To Special** dialog box (explained previously in this chapter) to select formula cells, constants, or other special cell types.

Viewing Formulas

You can become familiar with an unfamiliar workbook by displaying the formulas rather than the results of the formulas. Select **Tools • Options**, and check the box labeled **Formulas** on the **View** tab. You may want to create a new window for the workbook before issuing this command. That way, you can see the formulas in one window and the results in the other.

Figure 31-12 shows an example. The window on the top shows the normal view (formula results). The window on the bottom displays the formulas.

Commissions.xls:1

Commission Rate:	5.50%	Normal Commission Rate
Sales Goal:	15%	Improvement From Prior Month
Bonus Rate:	6.50%	Paid if Sales Goal is Attained

Sales Rep	Last Month	This Month	Change	Change	Met Goal?	Commission
Murray	101,233	98,744	(2,489)	-2.5%	FALSE	5,431
Knuckles	120,933	134,544	13,611	11.3%	FALSE	7,400
Lefty	112,344	134,997	22,543	20.1%	TRUE	8,768
Lucky	130,933	151,745	20,812	15.9%	TRUE	9,863
Scarface	150,932	140,778	(10,154)	-6.7%	FALSE	7,743
Totals	616,375	660,608	44,233	7.2%		33,205

Average Commission Rate: 6.93%

Commissions.xls:2

Commission Rate	0.055	Normal Commission Rate	
Sales Goal	0.15	Improvement From Prior Month	
Bonus Rate	0.065	Paid if Sales Goal is Attained	
Sales Rep	Last Month	This Month	Change
Murray	=B6	=C6	=C6-B6
Knuckles	=B7	=C7	=C7-B7
Lefty	=B8	=C8	=C8-B8
Lucky	=B9	=C9	=C9-B9
Scarface	=B10	=C10	=C10-B10
Totals	=SUM(B6:B10)	=SUM(C6:C10)	=SUM(C6:D10)

Figure 31-12: The underlying formulas are shown in the bottom window.

Pasting a List of Names

If the worksheet uses named ranges, create a list of the names and their references. Move the cell pointer to an empty area of the worksheet and choose **Insert • Name • Paste**. Excel responds with its **Paste Name** dialog box. Click on the **Paste List** button to paste a list of the names and their references into the workbook. Figure 31-13 shows an example.

Murray	101,233	98,744	(2,489)	-2.5%
Knuckles	120,933	134,544	13,611	11.3%
Lefty	112,344	134,887	22,543	20.1%
Lucky	130,933	151,746	20,812	15.9%
Scarface	150,932	140,778	(10,154)	-6.7%
Totals	616,375	660,696	44,323	7.2%
Average Commission Rate:		5.93%		
BonusRate	=Sheet1!\$B\$3			
CommissionRate	=Sheet1!\$B\$1			
Last_Month	=Sheet1!\$B\$6:\$B\$11			
SalesGoal	=Sheet1!\$B\$2			
This_Month	=Sheet1!\$C\$6:\$C\$11			
Totals	=Sheet1!\$B\$11:\$G\$11			

Figure 31-13: Pasting a list of names (in A15:B20) can sometimes help you understand how a worksheet is constructed.

Summary

In this chapter, I discuss tools that can help you make your worksheets error-free. I identify the types of errors that you're likely to encounter. I also cover three tools that Excel provides, which can help you trace the relationships between cells: the Info window, the Go To Special dialog box, and Excel's interactive auditing tools. I go over text-related features, including spell checking, AutoCorrect, and AutoComplete. I conclude the chapter with general tips that can help you understand how an unfamiliar worksheet is put together.

• • • • •

Fun Stuff

Although Excel is used primarily for serious applications, many users discover that this product has a lighter side. This chapter is devoted to the less-serious applications of Excel, including games and interesting diversions.

Games

Excel certainly wasn't designed as a platform for games. Nevertheless, I've developed a few games using Excel and have downloaded several others from various Internet sites. I've found that the key ingredient in developing these games is creativity. In almost every case, I had to invent one or more workarounds to compensate for Excel's lack of game-making features. In this section, I show you a few of my own creations.



The examples in this chapter are either available on the companion CD-ROM or included with the registered version of my Power Utility Pak (see the coupon at the back of the book).

Tick-Tack-Toe

Although Tick-Tack-Toe is not the most mentally stimulating game, everyone knows how to play it. Figure 32-1 shows the Tick-Tack-Toe game that I developed using Excel. In this implementation, the user plays against the computer. I wrote some formulas and VBA macros to determine the computer's moves, and it plays a reasonably good game—about on par with a three-year-old child. I'm embarrassed to admit that the program has even beaten me a few times (OK, so I was distracted!).



This workbook is available on the companion CD-ROM.

You can choose who makes the first move (you or the computer) and which marker you want to use (X or O). The winning games and ties are tallied in cells at the bottom of the window.

32

CHAPTER

In This Chapter

Games

Power Utility
Pac Games

Animation Shapes

Symmetrical Pattern
Drawing

For Guitar Players

An April Fool's Prank

Creating Word
Search Puzzles

ASCII Art

Sound File Player

Fun with Charts

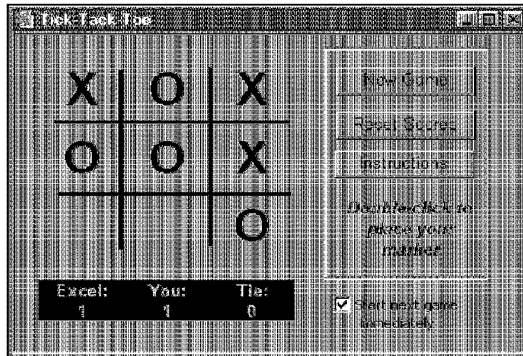


Figure 32-1: My Tick-Tack-Toe game.

Moving Tile Puzzle

At some time in your life, you've probably played one of those moving tile puzzles. They come in several variations, but the goal is always the same: rearrange the tiles so that they are in order.



This workbook is available on the companion CD-ROM.

Figure 32-2 shows a version of this game that I wrote using VBA. This version lets you choose the number of tiles (from a simple 3x3 matrix up to a challenging 6x6 matrix).

When you click the tile, it appears to move to the empty position. Actually, no movement is taking place. The program is simply changing the text on the buttons and making the button in the empty position invisible.

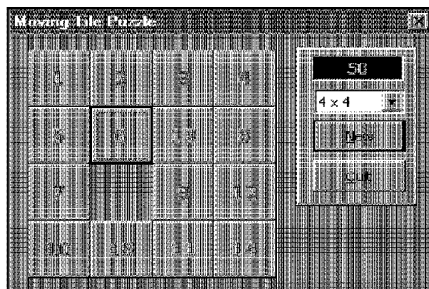


Figure 32-2: My Moving Tile puzzle.

Keno

If you've ever spent any time in a casino, you may be familiar with Keno (see Figure 32-3). If you're smart, you probably know to avoid this game like the plague, because it has the lowest return of any casino game. With my Keno for Excel, you don't have to worry about losing any money: all the action takes place on a worksheet, and no money changes hands. And, it's a lot faster than the casino version.



This workbook is available on the companion CD-ROM. In addition, I've included another workbook that calculates the various odds associated with Keno. Take a look at this workbook and you may never play casino Keno again!

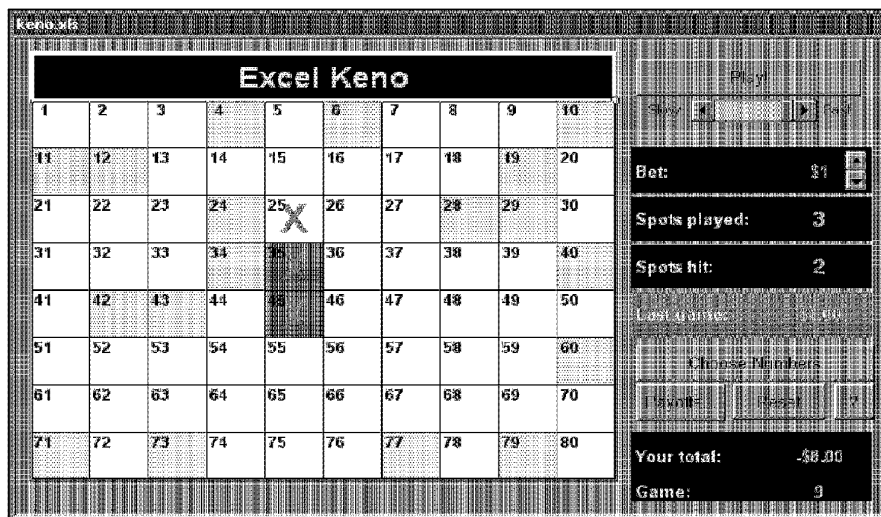


Figure 32-3: Keno for Excel.

Power Utility Pak Games

The four games listed in this section are included with my Power Utility Pak. Use the coupon in the back of the book to order your copy at a huge discount.

Video Poker

Developing my Video Poker game for Excel (see Figure 32-4) was quite a challenge. I was forced to spend many hours performing research at a local casino to perfect this game so that it captures the excitement of a real poker machine. The only problem is that I haven't figured out a way to dispense the winnings. Oh well, maybe in the next version.

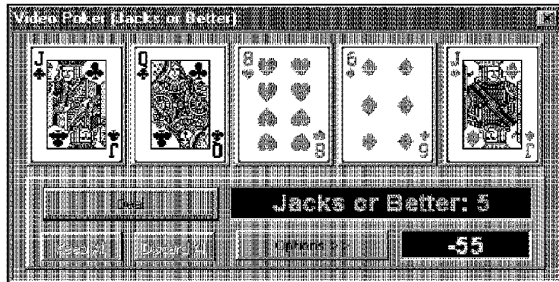


Figure 32-4: My Video Poker game.

This version has two games: Joker's Wild (a joker can be used for any card) and Jacks or Better (a pair of jacks or better is required to win). You select which cards to discard by clicking the card face. You can change the game (or the bet) at any time while playing. You can also request a graph that shows your cumulative winnings (or, more typically, your cumulative losses).

Identifying the various poker hands is done using VBA procedures. The game also has a Hide button that temporarily hides the game (pressing Esc has the same effect). You can then resume the game when your boss leaves the room.

This game is included with the registered version of the Power Utility Pak. See the coupon in the back of the book for details on how to get your copy.

Dice Game

The goal of the Dice Game (shown in Figure 32-5) is to obtain a high score by assigning dice rolls to various categories. You get to roll the dice three times on each turn, and you can keep or discard the dice before rolling again. Everything is done using VBA.

This game is included with the registered version of the Power Utility Pak. See the coupon in the back of the book for details on how to get your copy.

Bomb Hunt

Windows comes with a game called Minesweeper. I developed a version of this game for Excel and named it Bomb Hunt (see Figure 32-6). The goal is to discover the hidden bombs in the grid. Double-clicking a cell reveals a bomb (you lose) or a number that indicates the number of bombs in the surrounding cells. You use logic to determine where the bombs are located.

This game is included with the registered version of the Power Utility Pak. See the coupon in the back of the book for details on how to get your copy.

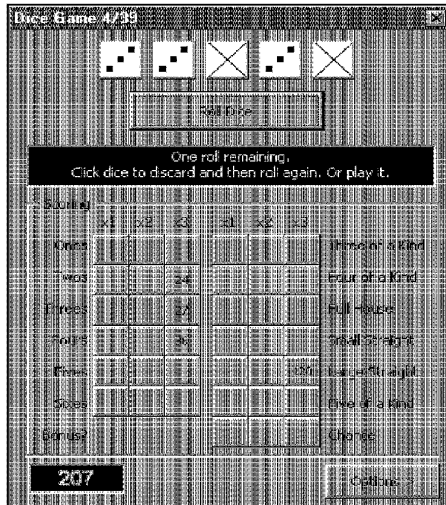


Figure 32-5: My Dice Game.

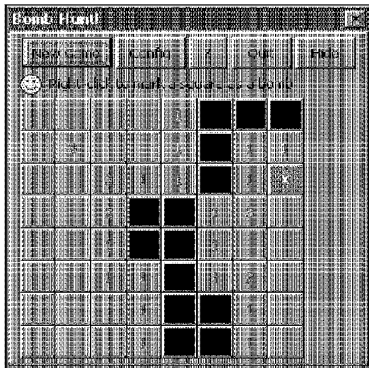


Figure 32-6: My Bomb Hunt game.

Hangman

Hangman is another game that almost everyone has played. Figure 32-7 shows a version that I developed for Excel. The objective is to identify a word by guessing letters. Correctly guessed letters appear in their proper position. Every incorrectly guessed letter adds a new body part to the person being hanged (to reduce gratuitous violence, I substituted a skeleton for the hanged gentleman). Ten incorrect guesses and the skeleton is completed—that is, the game is over.

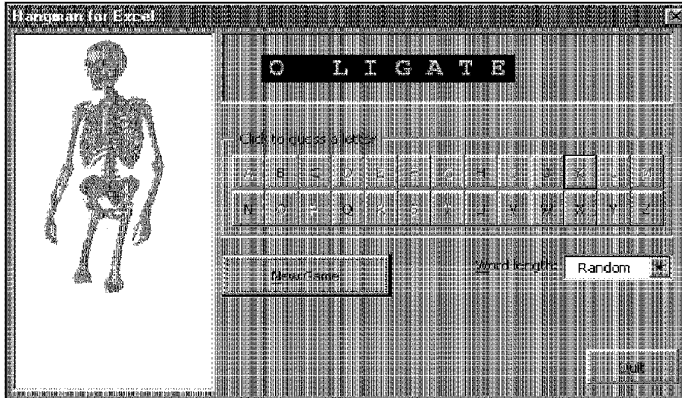


Figure 32-7: My Hangman game.

The workbook includes 1,400 words, ranging in length from 6 to 12 letters. You can either choose how many letters you want in the word or have the number of letters determined randomly. The entire game takes place in a dialog box.

Animated Shapes

With a bit of imagination (and lots of help from VBA), you can create some simple animations in a workbook. I've put together a few examples to demonstrate how it's done. Figure 32-8 shows an example (use your imagination — it really is animated).



Figure 32-8: Animated Shapes.



This workbook is available on the companion CD-ROM.

Symmetrical Pattern Drawing

I must admit, this program is rather addictive—especially for doodlers. It lets you create colorful symmetrical patterns by using the arrow keys on the keyboard. Figure 32-9 shows an example. As you draw, the drawing is reproduced as mirror images in the other three quadrants. When you move the cursor to the edge of the drawing area, it wraps around and appears on the other side. This workbook is great for passing the time on the telephone when you're put on hold.

The drawing is all done with VBA macros. I used the OnKey method to trap the following key presses: left, right, up, and down. Each of these keystrokes executes a macro that shades a cell. The cells in the drawing area are very tiny, so the shading appears as lines.



This workbook is available on the companion CD-ROM.

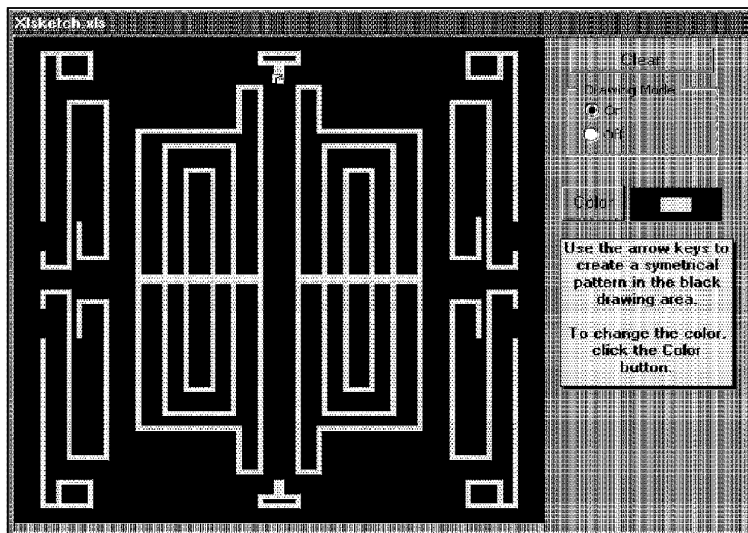


Figure 32-9: My Symmetrical Pattern Drawing worksheet.

For Guitar Players

If you play guitar, check out this workbook. As you see in Figure 32-10, this workbook has a graphic depiction of a guitar's fret board. It displays the notes (and fret positions) of the selected scale or mode in any key. You can even change the tuning of the guitar, and the formulas automatically recalculate.



This workbook is available on the companion CD-ROM.

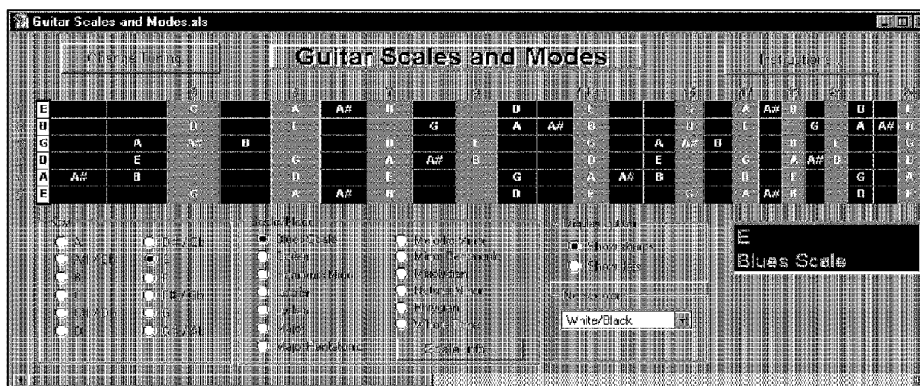


Figure 32-10: My guitar fret board application.

Other options include the choice to display half-notes as sharps or flats, to pop up information about the selected scale or mode, and to change the color of the guitar neck. This workbook uses formulas to do the calculation, and VBA plays only a minor role. This file was designated a “top pick” on America Online, and I’ve received positive feedback from fellow pickers all over the world.

An April Fool’s Prank

Here’s a good April Fool’s trick to play on an office mate (with luck, one with a sense of humor). When he or she is out of the office, load this workbook and click the button to reverse the menus. For example, the Insert • Name • Define command becomes the Tresni • Eman • Enifed command. Excel’s menus look like they’re in a strange language. Figure 32-11 shows how this looks.



This workbook is available on the companion CD-ROM.

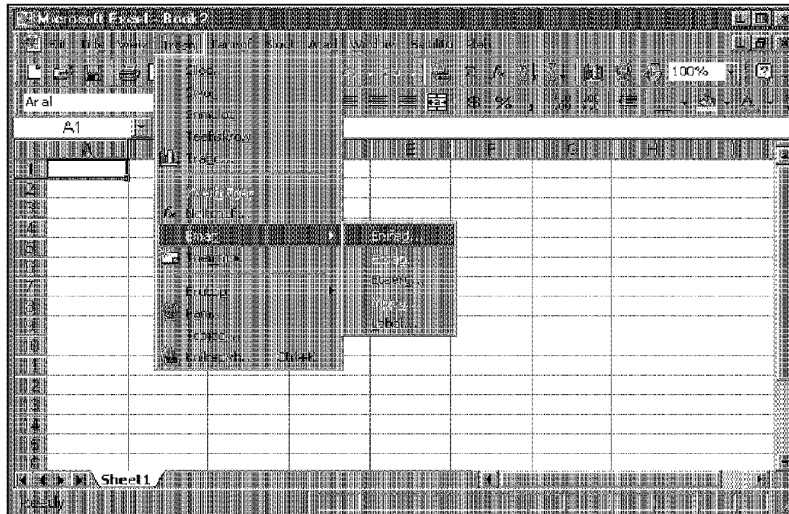


Figure 32-11: Excel with backward menus. The hot keys remain the same.

The routine performs its mischief by calling a custom function that reverses the text in the captions (except for the ellipses), converts the new text to proper case, and maintains the original hot keys. The net effect is a worksheet menu system that works exactly like the original (and is even keystroke-compatible) but looks very odd.

Clicking the Reset menu button returns the menus to normal.

Creating Word Search Puzzles

Most daily newspapers feature a word search puzzle. These puzzles contain words that are hidden in a grid. The words can be vertical, diagonal, horizontal, forwards, or backwards. If you've ever had the urge to create your own word search puzzle, this workbook can make your job a lot easier by doing it for you. You supply the words; the program places them in the grid and fills in the empty squares with random letters. Figure 32-12 shows the puzzle creation sheet plus a sample puzzle that was created with this application.

This is all done with VBA, and randomness plays a major role. Therefore, you can create multiple puzzles using the same words.



This workbook is available on the companion CD-ROM.

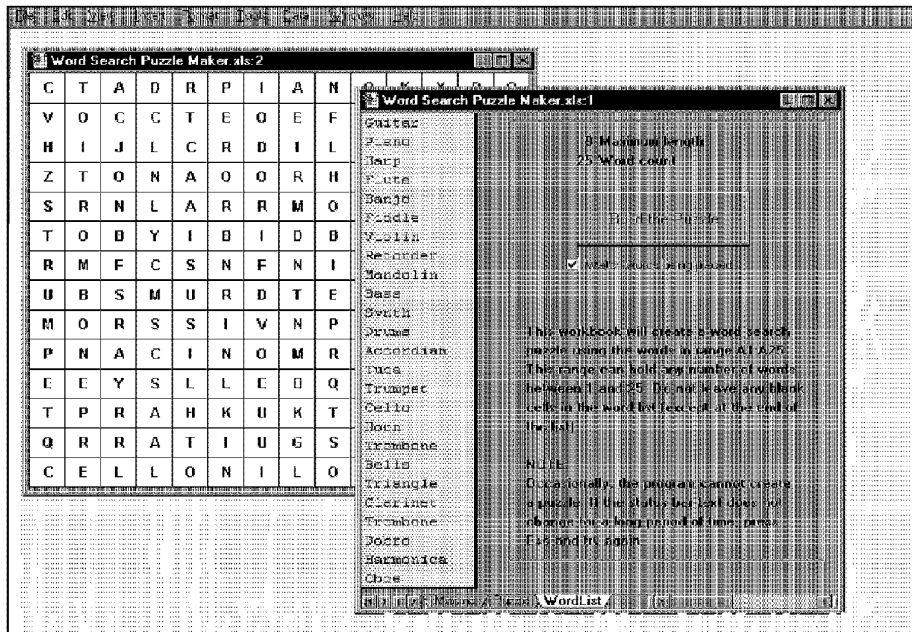


Figure 32-12: My Word Search Puzzle Maker.

ASCII Art

ASCII art consists of pictures made up of simple ASCII characters. The Internet is filled with thousands of examples of ASCII art. I created a workbook with a few examples that I picked up from the public domain. Figure 32-13 shows an example.



This workbook is available on the companion CD-ROM.

For the image to look correct, you must view ASCII art using a fixed-width font, such as Courier New.

Sound File Player

Excel doesn't have to be quiet. I created a simple macro that lets you play any WAV or MIDI file on your system.



This workbook is available on the companion CD-ROM.

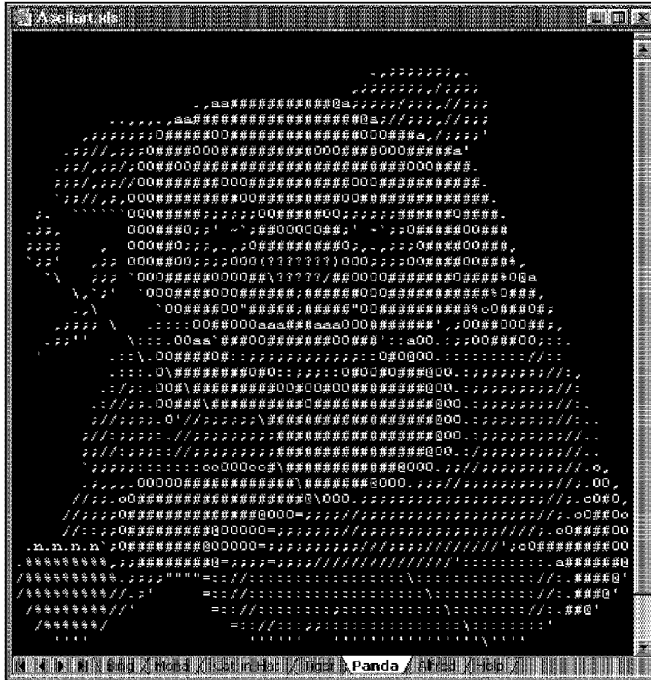


Figure 32-13: An example of ASCII art.

Fun with Charts

Excel's charting feature has the potential to be fun. In this section, I provide examples of some nonserious charting applications.

Plotting Trigonometric Functions

Although I don't know too much about trigonometry, I've always enjoyed plotting various trigonometric functions as XY charts. Sometimes you can come up with attractive images. Figure 32-14 shows an example of a trigonometric plot. Clicking the button changes a random number that makes a new chart.



This workbook is available on the companion CD-ROM.

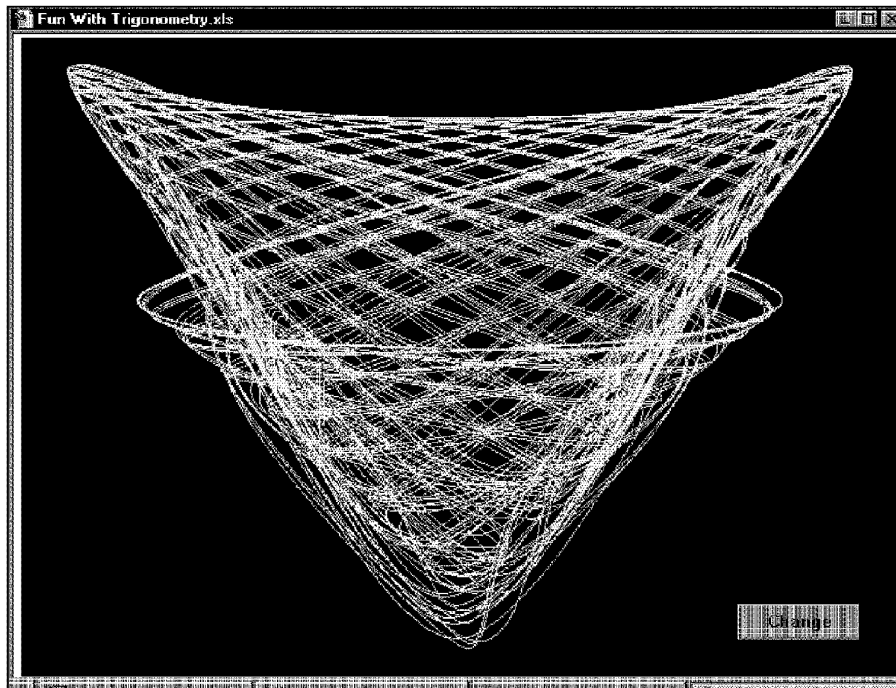


Figure 32-14: This chart plots trigonometric functions.

XY-Sketch

In this workbook, you use the controls to draw an XY chart (see Figure 32-15). Clicking a directional button adds a new X and Y value to the chart's data range, which is then plotted on the chart. You can change the step size, adjust the color, and choose between smooth and normal lines. I include a multilevel Undo button that successively removes data points that you added.



This workbook is available on the companion CD-ROM.

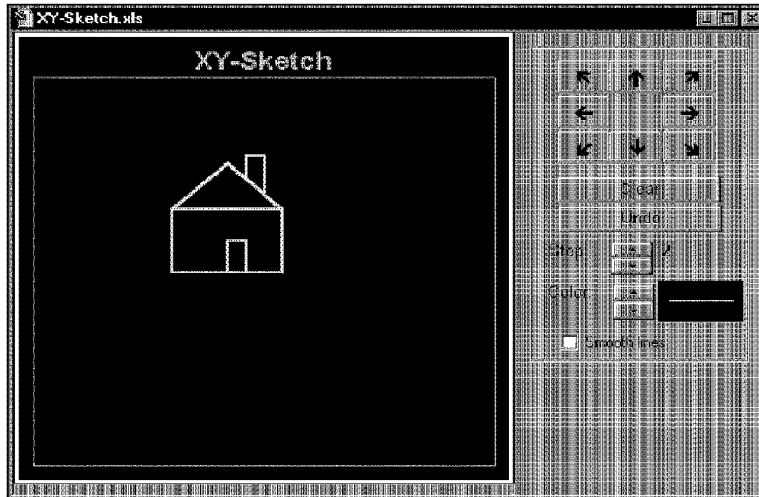


Figure 32-15: My XY-Sketch workbook.

Summary

In this chapter, I present several examples of nonserious applications for Excel. Some of these examples can most likely be adapted and used in more serious applications (well, maybe not).

•

Customizing Excel

P A R T

VI



In This Part

Chapter 33
Customizing Toolbars
and Menus

Chapter 34
Using and Creating
Templates

Chapter 35
Using Visual Basic for
Applications

Chapter 36
Creating Custom
Worksheet Functions

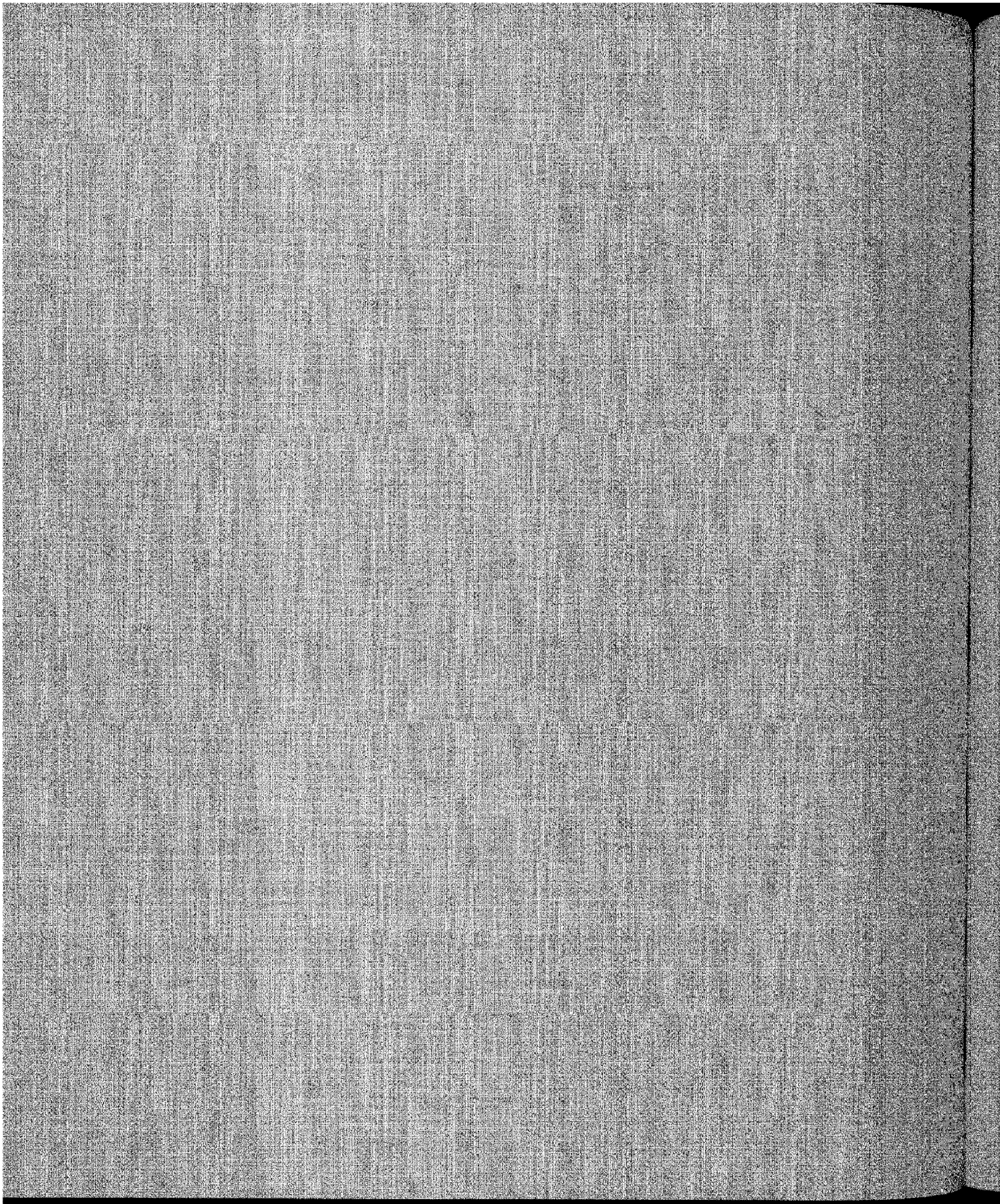
Chapter 37
Creating Custom
Dialog Boxes

Chapter 38
Using Dialog Box
Controls in Your
Worksheet

Chapter 39
VBA Programming
Examples

Chapter 40
Creating Custom
Excel Add-Ins





Customizing Toolbars and Menus

You're probably familiar with many of Excel's built-in toolbars, and you have most likely thoroughly explored the menu system. Excel lets you modify both toolbars and menus. This chapter explains how to customize the built-in toolbars, create new toolbars, and change the menus that Excel displays. Although many of these customizations are most useful when you create macros (discussed in subsequent chapters), even nonmacro users may find these techniques helpful.

Menu Bar = Toolbar

Beginning with Excel 97, virtually no distinction exists between a menu bar and a toolbar. In fact, the menu bar that you see at the top of Excel's window is actually a toolbar that is named Worksheet Menu Bar. As with any toolbar, you can move it to a new location by dragging it (see Figure 33-1).

Many of the menu items display icons in addition to text — a good sign that Excel's menus are not “real” menus. To further demonstrate that Excel's menu bars are different from those used in other programs, note that if you change the colors or fonts used for menus (using the Windows Control panel), these changes do not appear in Excel's menus.

33

CHAPTER

In This Chapter

Menu Bar = Toolbar

Customizing Toolbars

Moving Toolbars

Using the Customize
Dialog Box

Adding or Removing
Toolbar Buttons

Other Toolbar Button
Operations

Creating a Custom
Toolbar: An Example

Changing a Toolbar
Button's Image

Activating a Web
Page from a Toolbar
Button

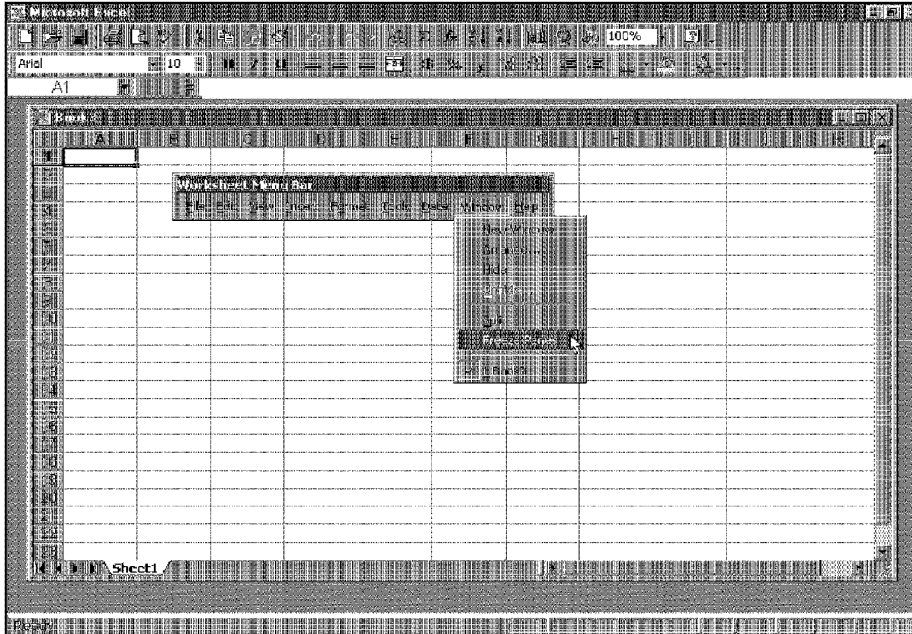


Figure 33-1: Excel's menu bar is actually a toolbar, and you can move it to any location that you want.

Customizing Toolbars

The official term for toolbars, menu bars, and shortcut menus is a *CommandBar*. All told, Excel comes with nearly 100 built-in *CommandBars*, made up of the following:

- Two menu bars (one for worksheets and one for chart sheets)
- 40 traditional style toolbars
- 51 shortcut menus (the menus that appear when you right-click a selection)

Each *CommandBar* consists of one or more “commands.” A command can take the form of an icon, text, or both. Some additional commands don’t appear on any of the prebuilt toolbars.

Many users like to create custom toolbars that contain the commands that they use most often.

How Excel Keeps Track of Toolbars

When you start Excel, it displays the same toolbar configuration that was in effect the last time that you used it. Did you ever wonder how Excel keeps track of this information? When you exit Excel, it updates a file in your Windows folder. This file stores your custom toolbars, as well as information about which toolbars are visible and the onscreen location of each. The file is stored in your Windows directory, and has an XLB extension (the actual filename will vary).

To restore the toolbars to their previous configuration, select File • Open to open this XLB file. This restores your toolbar configuration to the way that it was when you started Excel. You can also make a copy of the XLB file and give it a different name, which enables you to store multiple toolbar configurations that you can load at any time.

Types of Customizations

The following list is a summary of the types of customizations that you can make when working with toolbars (which also include menu bars):

- **Move toolbars.** Any toolbar can be moved to another location .
- **Remove buttons from built-in toolbars.** You may want to do this to eliminate buttons that you never use.
- **Add buttons to built-in toolbars.** You can add as many buttons as you want to any toolbar.
- **Create new toolbars.** You can create as many new toolbars as you like, with as many buttons as you like.
- **Change the functionality of a button.** You make such a change by attaching your own macro to a built-in toolbar button.
- **Change the image that appears on any toolbar button.** A rudimentary but functional toolbar-button editor is included with Excel.

Shortcut Menus

The casual user cannot modify Excel's shortcut menus (the menus that appear when you right-click an object). Doing so requires the use of VBA macros.

Moving Toolbars

A toolbar can be either floating or docked. A *docked* toolbar is fixed in place at the top, bottom, left, or right edge of Excel's workspace. *Floating* toolbars appear in an "always-on-top" window, and you can drag them wherever you like.

To move a toolbar, just click its border and drag it to its new position. If you drag it to one of the edges of Excel's window, it attaches itself to the edge and becomes docked. You can create several layers of docked toolbars. For example, the Standard and Formatting toolbars are (normally) both docked along the upper edge.

If a toolbar is floating, you can change its dimensions by dragging a border. For example, you can transform a horizontal toolbar to a vertical toolbar by dragging one of its corners.

Using the Customize Dialog Box

To make any changes to toolbars, you need to be in "customization mode." In customization mode, the Customize dialog box is displayed, and you can manipulate the toolbars in a number of ways. To get into customization mode, perform either of the following actions:

- Select View • Toolbars • Customize
- Select Customize from the shortcut menu that appears when you right-click a toolbar

Either of these methods displays the Customize dialog box that is shown in Figure 33-2. This dialog box lists all the available toolbars, including custom toolbars that you have created.

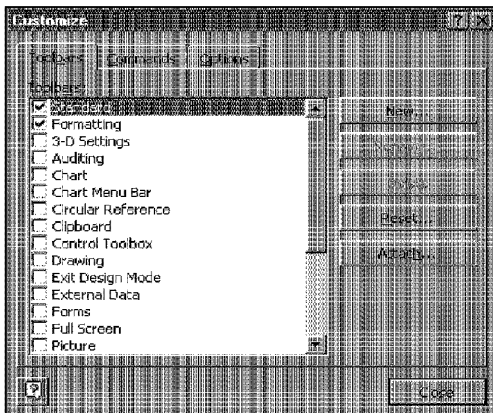


Figure 33-2: The Customize dialog box.

The Customize dialog box has three tabs, each of which is described in the following sections.

The Toolbars Tab

Figure 33-2 shows the Toolbars tab of the Customize dialog box. The following sections describe how to perform various procedures that involve toolbars.



Operations that you perform by using the Customize dialog box cannot be undone.

Hiding or displaying a toolbar

The Toolbars tab displays every toolbar (built-in toolbars and custom toolbars). Add a check mark to display a toolbar; remove the check mark to hide it. The changes take effect immediately.

Creating a new toolbar

Click the New button and then enter a name in the New Toolbar dialog box. Excel creates and displays an empty toolbar. You can then add buttons to the new toolbar. See “Adding or Removing Toolbar Buttons” later in this chapter.

Renaming a custom toolbar

Select a custom toolbar from the list and click the Rename button. Enter a new name in the Rename Toolbar dialog box. You cannot rename a built-in toolbar.

Deleting a custom toolbar

Select a custom toolbar from the list and click the Delete button. You cannot delete a built-in toolbar.

Resetting a built-in toolbar

Select a built-in toolbar from the list and click the Reset button. The toolbar is restored to its default state. If you’ve added any custom tools to the toolbar, they are removed. If you’ve removed any of the default tools, they are restored.

The Reset button is not available when a custom toolbar is selected.

Attaching a toolbar to a workbook

If you create a custom toolbar that you want to share with someone else, you can “attach” it to a workbook. To attach a custom toolbar to a workbook, click the Attach button, which presents the Attach Toolbars dialog box. Select the toolbars that you want to attach to a workbook (see Figure 33-3). You can attach any number of toolbars to a workbook.

A toolbar that’s attached to a workbook appears automatically when the workbook is opened, unless the workspace already has a toolbar by the same name.

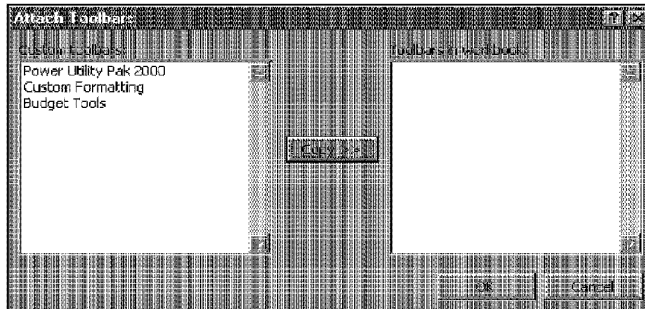


Figure 33-3: You can attach custom toolbars to a workbook in the Attach Toolbars menu.

The toolbar that's stored in the workbook is an exact copy of the toolbar at the time that you attach it. If you modify the toolbar after attaching it, the changed version is not stored in the workbook automatically. You must manually remove the old toolbar and then add the edited toolbar.

The Commands Tab

The Commands tab of the Customize dialog box contains a list of every tool that's available. Use this tab when you customize a toolbar. This feature is described later in the chapter (see “Adding or Removing Toolbar Buttons”).

The Options Tab

The Options tab of the Customize dialog box, shown in Figure 33-4, gives you several choices of ways to customize your menus, toolbars, icons, and the like. The following list explains these options.

- **Personalized Menus and Toolbars:** On the Options tab, the new options of Excel 2000 are Personalized Menus and Toolbars and, in the Other area, List font names in their font.
 - These options provide you with some control over how the menus and toolbars work. Set these options according to your personal preferences.
- **Large icons:** To change the size of the icons used in toolbars, select or deselect the Large icons check box. This option only affects the images that are in buttons. Buttons that contain only text (such as buttons in a menu) don't change.
- **List font names in their font:** This new feature displays the font names using the actual font. The advantage is that you can preview the font before you select it. The disadvantage is that it's a bit slower.

- **Show ScreenTips on toolbar:** ScreenTips are the pop-up messages that display the button names when you pause the mouse pointer over a button. If you find the ScreenTips distracting, remove the check mark from the Show ScreenTips on toolbars check box. The status bar still displays a description of the button when you move the mouse pointer over it.
- **Menu animations:** When you select a menu, Excel animates the display of the menu as it is dropping down. You can select the type of animation that you want:
 - **Slide:** The menu drops down with a sliding motion
 - **Unfold:** The menu unfolds as it drops down
 - **Random:** The menu either slides or unfolds randomly

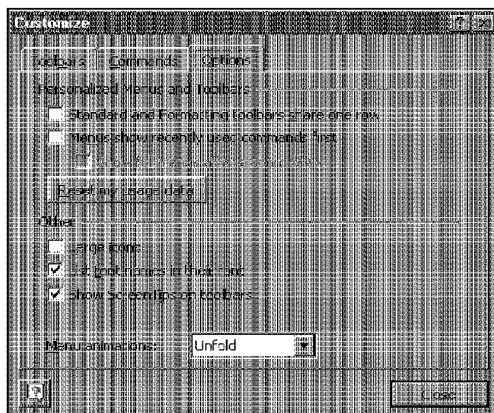


Figure 33-4: The Options tab of the Customize dialog box.

Toolbar Autosensing

Normally, Excel displays a particular toolbar automatically when you change contexts; this is called *autosensing*. For example, when you activate a chart, the Chart toolbar appears. When you activate a sheet that contains a pivot table, the PivotTable toolbar appears.

You can easily defeat autosensing by hiding the toolbar. After you do so, Excel no longer displays that toolbar when you switch to its former context. You can restore this automatic behavior, however, by displaying the appropriate toolbar when you're in the appropriate context. Thereafter, Excel reverts to its normal automatic toolbar display when you switch to that context.

Adding or Removing Toolbar Buttons

As noted earlier in this chapter, you can put Excel into customization mode by displaying the Customize dialog box. When Excel is in customization mode, you have access to all the commands and options in the Customize dialog box. In addition, you can perform the following actions:

- Reposition a button on a toolbar
- Move a button to a different toolbar
- Copy a button from one toolbar to another
- Add new buttons to a toolbar by using the Commands tab of the Customize dialog box

Moving and Copying Buttons

When the Customize dialog box is displayed, you can copy and move buttons freely among any visible toolbars. To move a button, drag it to its new location (the new location can be within the current toolbar or on a different toolbar).

To copy a button, press Ctrl while you drag the button to another toolbar. You can also copy a toolbar button within the same toolbar, but no reason really exists to have multiple copies of a button the same toolbar.

Inserting a New Button

To add a new button to a toolbar, you use the Commands tab of the Customize dialog box (see Figure 33-5).

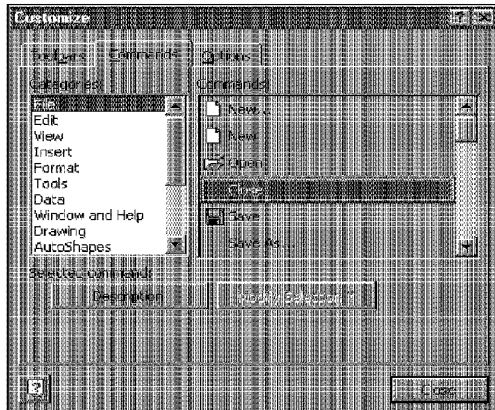


Figure 33-5: The Commands tab contains a list of every available button.

Other Toolbar Button Operations

When Excel is in customization mode (that is, the Customize dialog box is displayed), you can right-click a toolbar button to get a shortcut menu of additional actions for the tool. Figure 33-6 shows the shortcut menu that appears when you right-click a button in customization mode.

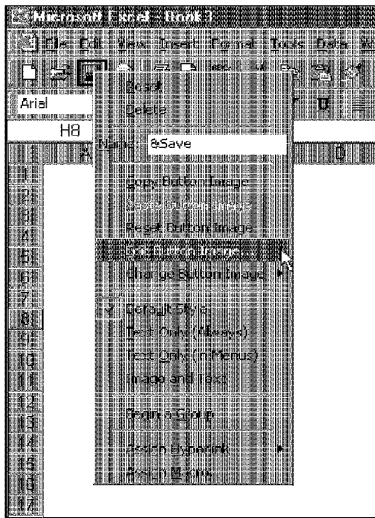


Figure 33-6: In customization mode, right-clicking a button displays this shortcut menu.

These commands are described in the following list (note that some of these commands are not available for certain toolbar tools):

- **Reset:** Resets the tool to its original state.
- **Delete:** Deletes the tool.
- **Name:** Lets you change the name of the tool.
- **Copy Button Image:** Makes a copy of the button's image and places it on the Clipboard.
- **Paste Button Image:** Pastes the image from the Clipboard to the button.
- **Reset Button Image:** Restores the button's original image.
- **Edit Button Image:** Lets you edit the button's image, using Excel's button editor.
- **Change Button Image:** Lets you change the image by selecting from a list of 42 button images.
- **Default Style:** Displays the tool with its default style (either text only or image and text).

- **Text Only (Always):** Always displays text (no image) for the tool.
- **Text Only (In Menus):** Displays text (no image) if the tool is in a menu bar.
- **Image and Text:** Displays the tool's image and text.
- **Begin a Group:** Inserts a divider in the toolbar. In a drop-down menu, a separator bar appears as a horizontal line between commands. In a toolbar, a separator bar appears as a vertical line.
- **Assign Hyperlink:** Lets you assign a hyperlink that will activate a Web page.
- **Assign Macro:** Lets you assign a macro that is executed when the button is clicked.



Assign Hyperlink is a new feature of Excel 2000.

Creating a Custom Toolbar: An Example

This section walks you through the steps that are used to create a custom toolbar. This toolbar is an enhanced Formatting toolbar that contains many additional formatting tools that aren't found on Excel's built-in Formatting toolbar. You may want to replace the built-in Formatting toolbar with this new custom toolbar.



If you don't want to create this toolbar yourself, this workbook is available on this book's CD-ROM.

Adding the First Button

The following steps are required to create this new toolbar and add one button (which has five subcommands):

1. Right-click any toolbar and select **Customize** from the shortcut menu.
Excel displays its **Customize** dialog box.
2. Click the **Toolbars** tab and then click **New**.
Excel displays its **New Toolbar** dialog box.
3. Enter a name for the toolbar: **Custom Formatting**. Click **OK**.
Excel creates a new (empty) toolbar.
4. In the **Customize** dialog box, click the **Commands** tab.
5. In the **Categories** list, scroll down and select **New Menu**.
The **New Menu** category has only one command (**New Menu**), which appears in the **Commands** list.
6. Drag the **New Menu** command from the **Commands** list to the new toolbar.
This creates a menu button in the new toolbar.

7. Right-click the New Menu button in the new toolbar and change the name to **Font**.
8. In the Customize dialog box, select **Format** from the Categories list.
9. Scroll down through the Commands list and drag the **Bold** command to the Font button in your new toolbar.
This step makes the Font button display a submenu (**Bold**) when the button is clicked.
10. Repeat Step 9, adding the following buttons from the Format category: **Italic**, **Underline**, **Font Size**, and **Font**.

At this point, you may want to click the Close button in the Customize dialog box to try out your new toolbar. The new toolbar contains only one button, but this button expands to show five font-related commands. Figure 33-7 shows the Custom Formatting toolbar at this stage.

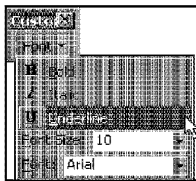


Figure 33-7: A new Custom Formatting toolbar after adding a menu button with five commands. In this example, Underline is selected.

Adding More Buttons

If you followed the steps in the previous section, you should understand how toolbar customization works, and you can now add additional buttons by following the procedures that you learned. To finish the toolbar, right-click a toolbar button and select **Customize**. Then, add additional tools.

Figure 33-8 shows the final version of the Custom Formatting toolbar, and Table 33-1 describes the tools on this toolbar. This customized toolbar includes all the tools that are on the built-in Formatting toolbar — plus quite a few more (38 tools in all). But, because the Custom Formatting toolbar uses five menus (which expand to show more commands), the toolbar takes up a relatively small amount of space.

You can, of course, customize the toolbar any way that you like. The tools that are listed in the table are my preferences. You may prefer to omit tools that you never use — or add other tools that you use frequently.

With two exceptions, all the tools are found in the Formatting category. The Clear Formatting tool is in the Edit category, and the Format Cells tool is in the Built-In Menus category.

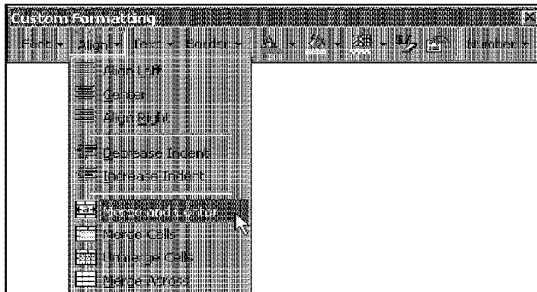


Figure 33-8: The final version of the Custom Formatting toolbar.

Table 33-1
Tools in the Custom Formatting Toolbar

<i>Tool</i>	<i>Subcommands</i>
New Menu (renamed Font)	Bold, Italic, Underline, Font Size, Font
New Menu (renamed Align)	Align Left, Center, Align Right, Decrease Indent, Increase Indent, Merge and Center, Merge Cells, Unmerge Cells, Merge Across
New Menu (renamed Text)	Vertical Text, Rotate Text Up, Rotate Text Down, Angle Text Downward, Angle Text Upward
New Menu (renamed Border)	Clear Border, Apply Outline Borders, Apply Inside Border, Left Border, Right Border, Top Border, Bottom Border, Inside Vertical Border, Inside Horizontal Border, Bottom Double Border
Font Color	(none)
Fill Color	(none)
Pattern	(none)
Clear Formatting	(none)
Format Cells	(none)
New Menu (renamed Number)	Currency Style, Percent Style, Comma Style, Decrease Decimal, Increase Decimal

Saving the Custom Toolbar

Excel doesn't have a command to save a toolbar. Rather, the new toolbar is saved when you exit Excel. Refer to the sidebar "How Excel Keeps Track of Toolbars," earlier in this chapter.



You need to remember that if Excel shuts down by non-normal means (that is, it crashes!), your custom toolbar will be lost. Therefore, if you invest a lot of time creating a new toolbar, you should close Excel to force the new toolbar to be saved.

Changing a Toolbar Button's Image

To change the image that is displayed on a toolbar button, you have several options:

- Choose 1 of the 42 images that are provided by Excel
- Modify or create the image by using Excel's Button Editor dialog box
- Copy an image from another toolbar button

Each of these methods is discussed in the following sections.

To make any changes to a button image, you must be in toolbar customization mode (the Customize dialog box must be visible). Right-click any toolbar button and select Customize from the shortcut menu.

Using a Built-in Image

To change the image on a toolbar button, right-click the button and select Change Button Image from the shortcut menu. As you can see in Figure 33-9, this menu expands to show 42 images from which you can choose. Just click the image that you want, and the selected button's image changes.

Editing a Button Image

If none of the 42 built-in images suits your tastes, you can edit an existing image or create a new image by using Excel's Button Editor.

To begin editing, right-click the button that you want to edit and then choose Edit Button Image from the shortcut menu. The image appears in the Button Editor dialog box (see Figure 33-10), in which you can change individual pixels and shift the entire image up, down, to the left, or to the right. If you've never worked with icons before, you may be surprised at how difficult it is to create attractive images in such a small area.

The Edit Button Image dialog box is straightforward. Just click a color and then click a pixel (or drag across pixels). When it looks good, click OK. Or, if you don't like what you've done, click Cancel, and the button keeps its original image.

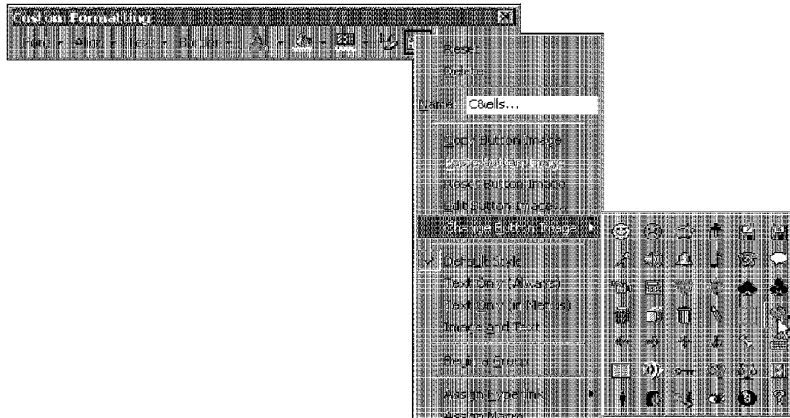


Figure 33-9: The Change Button Image option gives you 42 built-in button images to choose from.

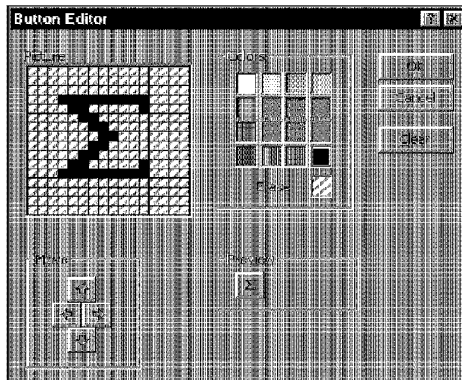


Figure 33-10: The Button Editor dialog box, in which you can design your own button image or edit an existing one.

Copying Another Button Image

Another way to get a button image on a custom toolbar is to copy it from another toolbar button. Right-click a toolbar button, and it displays a shortcut menu that enables you to copy a button image to the Clipboard or paste the Clipboard contents to the selected button.

Activating a Web Page from a Toolbar Button

You might want to create a button that activates your Web browser and loads a Web page.



This feature is available only in Excel 2000.

To add a new button and attach a hyperlink, make sure that you're in toolbar customization mode. Use the procedure previously described to add a new button and (optionally) specify a button image. Then, right-click the button and select **Assign Hyperlink • Open**. You'll see the **Assign Hyperlink: Open** dialog box, shown in Figure 33-11. Type a URL or select one from the list.

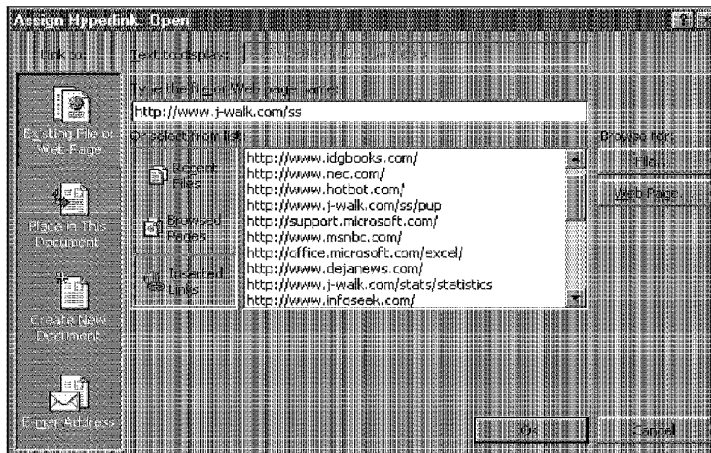


Figure 33-11: The **Assign Hyperlink: Open** dialog box enables you to assign a hyperlink to a toolbar button.

Summary

This chapter discusses how to modify two components of Excel's user interface: toolbars and menus. Users of all levels can benefit from creating custom toolbars. To create new commands that are executed by toolbar buttons, however, you need to write macros. This chapter also discusses how to change the image that appears on a toolbar button, and then introduces Excel's menu editor, which is most useful for macro writers.

•

Using and Creating Templates

This chapter covers one of the most potentially useful features in Excel — template files. Templates can be used for a variety of purposes, ranging from custom “fill-in-the-blanks” workbooks to a way to change Excel’s defaults for new workbooks or new worksheets.

An Overview of Templates

A *template* is essentially a model that serves as the basis for something else. An Excel template is a workbook that’s used to create other workbooks. If you understand this concept, you may save yourself a lot of work. For example, you may always use a particular header on your printouts. Consequently, every time that you print a worksheet, you need to select File • Page Setup to add your page header. The solution is to create a new workbook by modifying the template that Excel uses. In this case, you modify the template file by inserting your header into the template. Save the template file, and then every new workbook that you create has your customized page header.

Excel supports three types of templates:

- **The default workbook template:** Used as the basis for new workbooks.
- **The default worksheet template:** Used as the basis for new worksheets that are inserted into a workbook.
- **Custom workbook templates:** Usually, ready-to-run workbooks that include formulas. Typically, these templates are set up so that a user can simply plug in values and get immediate results. The Spreadsheet Solutions templates (included with Excel) are examples of this type of template.

34

CHAPTER

In This Chapter

• An Overview of Templates

• The Default Worksheet Template

• Custom Workbook Templates

• Creating Custom Templates

• Ideas for Creating Templates

Each template type is discussed in the following sections.

The Default Workbook Template

Every new workbook that you create starts out with some default settings. For example, the workbook's worksheets have gridlines, text appears in Arial 10-point font, values that are entered display in the General number format, and so on. If you're not happy with any of the default workbook settings, you can change them.

Changing the Workbook Defaults

Making changes to Excel's default workbook is fairly easy to do, and it can save you lots of time in the long run. Take the following steps to change Excel's workbook defaults:

1. Start with a new workbook.
2. Add or delete sheets to give the workbook the number of worksheets that you want.
3. Make any other changes that you want to make, which can include column widths, named styles, page setup options, and many of the settings that are available in the Options dialog box.



To change the default formatting for cells, choose **Format • Style**, and then modify the settings for the Normal style. For example, you can change the default font, size, or number format. Refer to "Using Named Styles" in Chapter 11 for details.

4. When your workbook is set up to your liking, select **File • Save As**.
5. In the Save As dialog box, select **Template (*.xlt)** from the Save as type box.
6. Enter **book.xlt** for the filename.
7. Save the file in your \XLStart folder. This folder is probably located within your c:\Program Files\Microsoft Office\Office folder.
You can also save your book.xlt template file in the folder that is specified as an alternate startup folder. You specify an alternate startup folder in the General tab of the Options dialog box.
8. Close the file.

After you perform the preceding steps, the new default workbook is based on the book.xlt workbook template. You can create a workbook based on your template by using any of the following methods:

- Click the **New** button on the Standard toolbar
- Press **Ctrl+N**
- Choose **File • New** and then select the **Workbook** icon in the General tab of the New dialog box (see Figure 34-1)

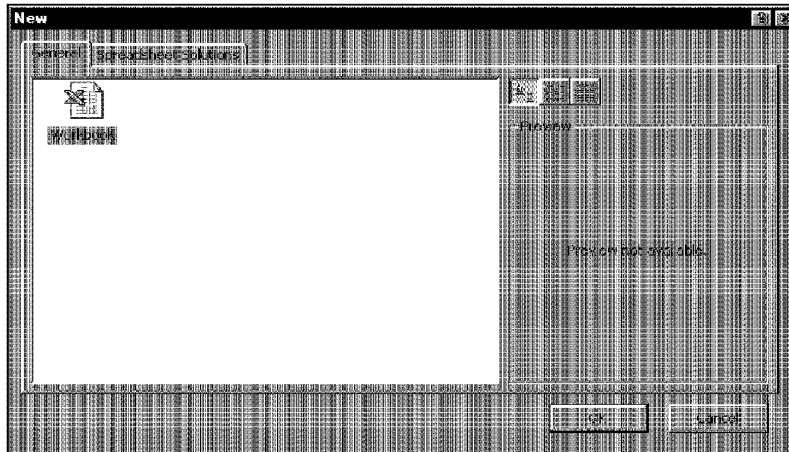


Figure 34-1: After you create a book.xlt template, clicking the Workbook icon creates a new workbook that is based on your template.

**Note**

Normally, the Xlstart folder does not contain a file named `book.xlt`. If a file with this name is not present, Excel creates new workbooks using built-in default settings.

Editing the book.xlt Template

After you create your `book.xlt` template, you may discover that you need to change it. You can open the `book.xlt` template file and edit it just like any other workbook. After you finish with your edits, save the workbook and close it.

Resetting the Default Workbook

If you create a `book.xlt` file and then decide that you would rather use the standard default workbook settings, simply delete the `book.xlt` template file from the Xlstart folder. Excel then resorts to its built-in default settings for new workbooks.

The Default Worksheet Template

When you insert a new worksheet into a workbook, Excel uses its built-in worksheet defaults for the worksheet. This includes items such as column width, row height, and so on.

**Note**

Versions of Excel prior to Excel 97 also use other sheet templates (`dialog.xlt` and `macro.xlt`). These templates are not used in Excel 97 or later versions.

If you don't like the default settings for a new worksheet, you can change them by using the following procedure:

1. Start with a new workbook, deleting all the sheets except one.
2. Make any changes that you want to make, which can include column widths, named styles, page setup options, and many of the settings that are available in the Options dialog box.
3. When your workbook is set up to your liking, select File • Save As.
4. In the Save As dialog box, select Template (*.xlt) from the Save as type box.
5. Enter **sheet.xlt** for the filename.
6. Save the file in your \XLStart folder. This folder is probably located within your c:\Program Files\Microsoft Office\Office folder.
You can also save your book.xlt template file in the folder that is specified as an alternate startup folder. You specify an alternate startup folder in the General tab of the Options dialog box.
7. Close the file.

After performing this procedure, all new sheets that you insert with the Insert • Worksheet command are formatted like your sheet.xlt template.

When you right-click a sheet tab and choose Insert from the shortcut menu, Excel displays its Insert dialog box (which looks just like the New dialog box). If you've created a template named sheet.xlt, you can select it by clicking the icon labeled Worksheet.

Editing the sheet.xlt Template

After you create your sheet.xlt template, you may discover that you need to change it. You can open the sheet.xlt template file and edit it just like any other workbook. After you make your changes, save the file and close it.

Resetting the Default New Worksheet

If you create a sheet.xlt template and then decide that you would rather use the standard default new worksheet settings, simply delete the sheet.xlt template file from the XLstart folder. Excel then resorts to its built-in default settings for new worksheets.

Custom Workbook Templates

The book.xlt and sheet.xlt templates discussed in the previous section are two special types of templates that determine default settings for new workbooks and

new worksheets. This section discusses other types of templates, referred to as *workbook templates*, which are simply workbooks that are set up to be used as the basis for new workbooks.

Why use a workbook template? The simple answer is that it saves you from repeating work. Assume that you create a monthly sales report that consists of your company's sales by region, plus several summary calculations and charts. You can create a template file that consists of everything except the input values. Then, when it's time to create your report, you can open a workbook based on the template, fill in the blanks, and you're finished.

You could, of course, just use the previous month's workbook and save it with a different name. This is prone to errors, however, because you easily can forget to use the Save As command and accidentally overwrite the previous month's file.

How Templates Work

When you create a workbook that is based on a template, Excel creates a copy of the template in memory so that the original template remains intact. The default workbook name is the template name with a number appended. For example, if you create a new workbook based on a template named Sales Report.xlt, the workbook's default name is Sales Report1.xls. The first time that you save a workbook that is created from a template, Excel displays its Save As dialog box, so that you can give the template a new name if you want to.

Templates That Are Included with Excel

Excel ships with three workbook templates (called Spreadsheet Solutions templates), which were developed by Village Software. When you select File • New, you can select one of these templates from the New dialog box. Click the tab labeled Spreadsheet Solutions to choose one of the following templates upon which to base your new workbook (see Figure 34-2).



These templates are included with Excel 2000.

- **Expense Statement:** Helps you to create expense report forms and a log to track them
- **Invoice:** Helps you to create invoices
- **Purchase Order:** Helps you to create purchase orders to send to vendors



A fourth template, named Village Software.xlt, describes additional templates that you can obtain from Village Software.



You can also download some additional templates from Microsoft's Web site: <http://www.microsoft.com/excel>.

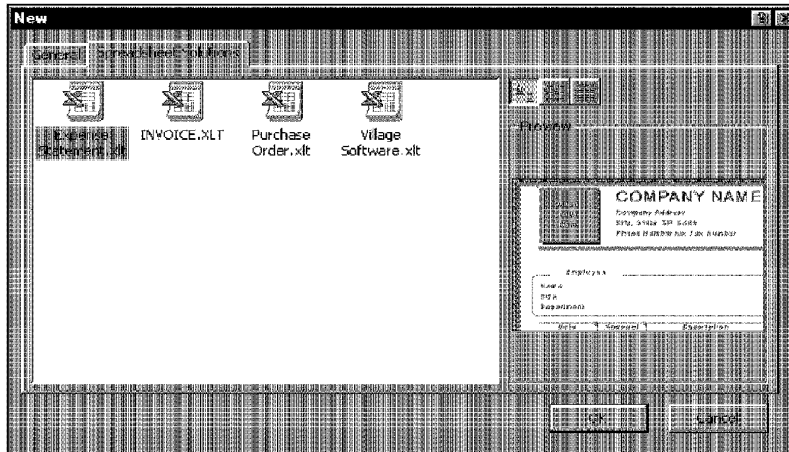


Figure 34-2: You can create a new workbook based on one of the Spreadsheet Solutions templates.

Creating Custom Templates

This section describes how to create workbook templates, which is really quite simple.

A *custom template* is essentially a normal workbook, and it can use any of Excel's features, such as charts, formulas, and macros. Usually, a template is set up so that the user can enter values and get immediate results. In other words, most templates include everything but the data — which is entered by the user.

If the template is going to be used by novices, you may consider locking all the cells except the input cells (use the Protection panel of the Format Cells dialog box for this). Then, protect the worksheet by choosing Tools • Protection • Protect Sheet.

To save the workbook as a template, choose File • Save As and select Template (*.xlt) from the drop-down list labeled Save as type. Save the template in your Microsoft Office\Templates folder (or a folder within that Templates folder).

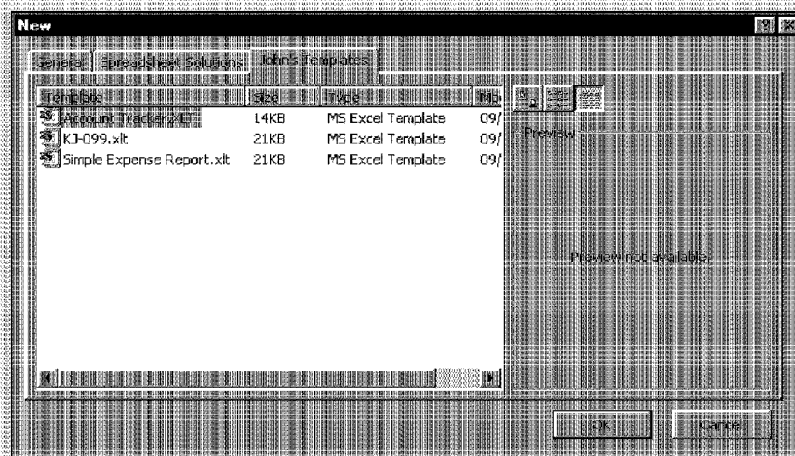
Before you save the template, you may want to specify that the file be saved with a preview image. Select File • Properties, and check the box that is labeled Save Preview Picture. That way, the New dialog box displays the preview when the template's icon is selected.

Where to Store Your Templates

Template files can be stored anywhere. When you open a template file (by selecting File • New), you don't actually open the template. Rather, Excel creates a new workbook that's based on the template that you specify. However, your templates are easier to access if you store them in one of the following locations:

- **Your \XLStart folder:** This is probably located within c:\Program Files\Microsoft Office\Office. If you create a default workbook template (book.xlt) or a default worksheet template (sheet.xlt), you store these templates in this folder.
- **Your \Templates folder:** This is probably located within your c:\Program Files\Microsoft Office\Office folder. Custom templates that are stored here appear in the New dialog box.
- **A folder located in your \Templates folder:** If you create a new folder within this folder, its name appears as a tab in the New dialog box. Clicking the tab displays the templates that are stored in that folder. The accompanying figure shows how the New dialog box looks when a new folder (named John's Templates) is in the Templates folder.

If you've specified an alternate startup folder (using the General panel of the Options dialog box), templates that are stored in that location also appear in the New dialog box.



If you later discover that you want to modify the template, choose File • Open to open and edit the template (don't use the File • New command, which creates a workbook that is based on the template).

Ideas for Creating Templates

This section provides a few ideas that may spark your imagination for creating templates. A partial list of the settings that you can adjust and use in your custom templates is as follows:

- **Multiple formatted worksheets:** You can, for example, create a workbook template that has two worksheets: one formatted to print in landscape mode and one formatted to print in portrait mode.
- **Workbook properties:** You can set one or more workbook properties. For example, Excel doesn't store a preview picture of your workbook. Select File • Properties and then change the Save Preview Picture option in the Summary panel.
- **Several settings in the View panel of the Options dialog box:** For example, you may not like to see sheet tabs, so you can turn off this setting.
- **Color palette:** Use the Color panel of the Options dialog box to create a custom color palette for a workbook.
- **Style:** The best approach is to choose Format • Style and modify the attributes of the Normal style. For example, you can change the font or size, the alignment, and so on.
- **Custom number formats:** If you create number formats that you use frequently, these can be stored in a template.
- **Column widths and row heights:** You may prefer that columns be wider or narrower, or you may want the rows to be taller.
- **Print settings:** Change these settings in the Page Setup dialog box. You can adjust the page orientation, paper size, margins, header and footer, and several other attributes.
- **Sheet settings:** These are options in the Options dialog box. They include gridlines, automatic page break display, and row and column headers.

Summary

This chapter introduces the concept of templates. Excel supports three template types: a default workbook template, a default worksheet template, and custom workbook templates. This chapter describes how to create such templates and where to store them. It also discusses the Template Wizard, a tool that helps you to create templates that can store data in a central database.

•

Using Visual Basic for Applications

This chapter is an introduction to the Visual Basic for Applications (VBA) macro language—perhaps the key component for users who want to customize Excel. A complete discussion of VBA would require an entire book. This chapter teaches you how to record macros and create simple macro subroutines. Subsequent chapters expand upon the topics in this chapter.

Introducing VBA Macros

In its broadest sense, a *macro* is a sequence of instructions that automates some aspect of Excel so that you can work more efficiently and with fewer errors. You may create a macro, for example, to format and print your month-end sales report. After the macro is developed and debugged, you can invoke the macro — with a single command — to perform many time-consuming procedures automatically.

Macros are usually considered to be one of the advanced features of Excel, because you must have a pretty thorough understanding of Excel to put them to good use. The truth is that the majority of Excel users have never created a macro and probably never will. If you want to explore one of the most powerful aspects of Excel, however, you should know about macros. This chapter is designed to acquaint you with VBA, which enables you to develop simple macros and execute macros that are developed by others.

35

CHAPTER

In This Chapter

Two Types of VBA Macros

Creating VBA Macros

Recording VBA Macros

More About Recording VBA Macros

Writing VBA Code

Learning More

Are Macros for You?

You need not be a power user to create and use simple VBA macros. Casual users can simply turn on Excel's macro recorder; Excel records and then converts your subsequent actions into a VBA macro — which is essentially a program. When you execute this program, Excel performs the actions again. More advanced users, though, can write code that tells Excel to perform tasks that can't be recorded. For example, you can write procedures that display custom dialog boxes, add new commands to Excel's menus, or process data in a series of workbooks.

VBA: One of Two Macro Languages in Excel

VBA was introduced in Excel 5. Prior to that version, Excel used an entirely different macro system, known as *XLM* (that is, the Excel 4 macro language). VBA is far superior in terms of both power and ease of use. For compatibility reasons, however, the XLM language is still supported in Excel 2000. This means that you can load an older Excel file and still execute the macros that are stored in it. However, Excel 2000 does not let you record XLM macros — and you really have no reason to do so.

What You Can Do with VBA

VBA is an extremely rich programming language with thousands of uses. The following list contains just a few things that you can do with VBA macros:

- **Insert a text string or formula:** If you need to enter your company name into worksheets frequently, you can create a macro to do the typing for you. The AutoCorrect feature can also do this.
- **Automate a procedure that you perform frequently:** For example, you may need to prepare a month-end summary. If the task is straightforward, you can develop a macro to do it for you.
- **Automate repetitive operations:** If you need to perform the same action in 12 different workbooks, you can record a macro while you perform the task once — and then let the macro repeat your action in the other workbooks.
- **Create a custom command:** For example, you can combine several of Excel's menu commands so that they are executed from a single keystroke or from a single mouse click.
- **Create a custom toolbar button:** You can customize Excel's toolbars with your own buttons to execute macros that you write.
- **Create a simplified "front end" for users who don't know much about Excel:** For example, you can set up a foolproof data entry template.

- **Develop a new worksheet function:** Although Excel includes a wide assortment of built-in functions, you can create custom functions that greatly simplify your formulas.
- **Create complete, turnkey, macro-driven applications:** Excel macros can display custom dialog boxes and add new commands to the menu bar.
- **Create custom add-ins for Excel:** Most of the add-ins that are shipped with Excel were created with Excel macros. I used VBA exclusively to create my Power Utility Pak.

Two Types of VBA Macros

Before getting into the details of creating macros, you need to understand a key distinction. A VBA macro (or procedure) can be one of two types: a *subroutine* or a *function*. The next two sections discuss the difference.

VBA Subroutines

You can think of a *subroutine macro* as a new command that can be executed by either the user or another macro. You can have any number of subroutines in an Excel workbook.

Figure 35-1 shows a simple VBA subroutine. When this subroutine is executed, VBA inserts the current date into the active cell, formats it, and then adjusts the column width.

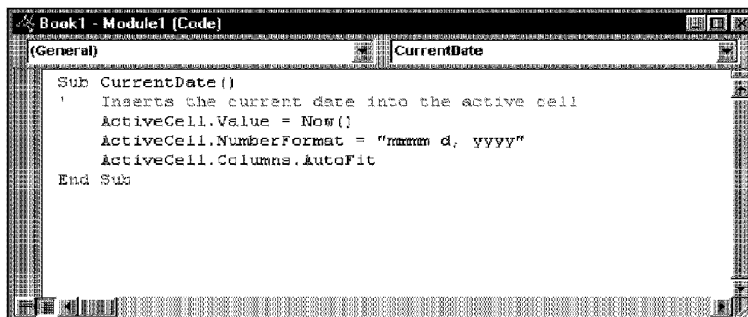


Figure 35-1: A simple VBA subroutine.

Subroutines always start with the keyword `Sub`, the macro's name (every macro must have a unique name), and then a pair of parentheses. (The parentheses are required; they are empty unless the procedure uses one or more arguments.) The `End Sub` statement signals the end of a subroutine. The lines in between comprise the procedure's code.

Using VBA in Excel 2000

If you've used VBA in an Excel version prior to Excel 97, you should be aware that Excel 2000 handles VBA very differently than do previous versions. The following is a list of a few of the key areas in which Excel 2000 differs from Excel 95 and Excel 5:

- Excel 2000 does not display module sheets in a workbook. To view or edit VBA code, you must activate the Visual Basic Editor (Alt+F11 toggles between Excel and the Visual Basic Editor).
- Excel 2000 makes use of UserForms rather than dialog sheets. You create and edit UserForms in the Visual Basic Editor. A UserForm also contains the VBA code that works with the objects in the form. This feature is discussed in Chapter 37.
- Excel 2000 includes many new objects and methods, providing the macro programmer with a great deal of new capability.

Excel 5 and Excel 95 macros and dialog boxes continue to work in Excel 2000. However, the compatibility is not perfect, and you may notice a few problems due to changes.

The subroutine shown in Figure 35-1 also includes a comment. *Comments* are simply notes to yourself, and they are ignored by VBA. A comment line begins with an apostrophe. You can also put a comment after a statement. In other words, when VBA encounters an apostrophe, it ignores the rest of the text in the line.

You execute a subroutine in any of the following ways:

- Choose Tools • Macro and then select the subroutine's name from the list.
- Press the subroutine's shortcut key combination (if it has one).
- If the Visual Basic Editor is active, move the cursor anywhere within the subroutine and press F5.
- Refer to the subroutine in another VBA procedure.

Subroutines are covered in detail later in this chapter.

VBA Functions

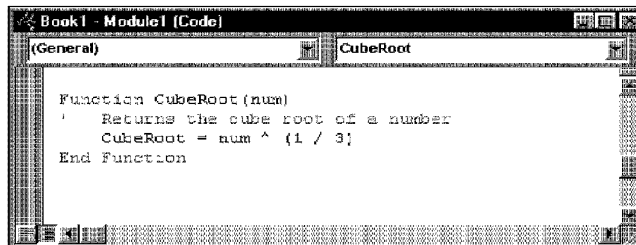
The second type of VBA procedure is a function. A *function* always returns a single value (just as a worksheet function always returns a single value). A VBA function can be executed by other VBA procedures or used in worksheet formulas, just as you would use Excel's built-in worksheet functions.

Some Definitions

VBA newcomers are often overwhelmed by the terminology that is used in VBA. I've put together some key definitions to help you keep the terms straight. These terms cover VBA and UserForms (custom dialog boxes) – two important elements that are used to customize Excel:

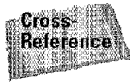
- **Code:** VBA instructions that are produced in a module sheet when you record a macro. You also can enter VBA code manually.
- **Controls:** Objects on a UserForm (or in a worksheet) that you manipulate. Examples include buttons, check boxes, and list boxes.
- **Function:** One of two types of VBA macros that you can create (the other is a subroutine). A function returns a single value. You can use VBA functions in other VBA macros or in your worksheets.
- **Macro:** A set of Excel instructions that are performed automatically. Excel macros can be XLM macros or VBA macros. This book focuses exclusively on VBA macros, which are also known as *procedures*.
- **Method:** An action that is taken on an object. For example, applying the `Clear` method to a range object erases the contents of the cells.
- **Module:** A container for VBA code.
- **Object:** An element that you manipulate with VBA. Examples include ranges, charts, drawing objects, and so on.
- **Procedure:** Another name for a macro. A VBA procedure can be a subroutine or a function.
- **Property:** A particular aspect of an object. For example, a range object has properties such as `Height`, `Style`, and `Name`.
- **Subroutine:** One of two types of Visual Basic macros that you can create. The other is a function.
- **UserForm:** A container that holds controls for a custom dialog box, and holds VBA code to manipulate the controls. (Custom dialog boxes are explained in depth in Chapter 36).
- **VBA:** Visual Basic for Applications. The macro language that is available in Excel as well as in the other applications in Microsoft Office 2000.
- **VBE:** Visual Basic Editor. The window (separate from Excel) that you use to create VBA macros and UserForms.

Figure 35-2 shows the listing of a custom worksheet function and shows the function in use in a worksheet. This function is named `CubeRoot` and requires a single argument. `CubeRoot` calculates the cube root of its argument. A function looks much like a subroutine. Notice, however, that function procedures begin with the keyword `Function`, and end with an `End Function` statement.



```
Book1 - Module1 (Code)
(CubeRoot)
Function CubeRoot(num)
    Returns the cube root of a number
    CubeRoot = num ^ (1 / 3)
End Function
```

Figure 35-2: This VBA function returns the cube root of its argument.



Creating VBA functions that you use in worksheet formulas can simplify your formulas and enable you to perform calculations that otherwise may be impossible. VBA functions are discussed in greater detail in Chapter 36.

Creating VBA Macros

Excel provides two ways to create macros:

- Turn on the macro recorder and record your actions
- Enter the code directly into a VBA module

The following sections describe both of these methods.

Recording VBA Macros

The basic steps that you take to record a VBA macro are described in this section. In most cases, you can record your actions as a macro and then simply replay the macro; you needn't look at the code that's generated. If this is as far as you go with VBA, you don't need to be concerned with the language itself (although a basic understanding of how things work doesn't do any harm).

Recording Your Actions to Create VBA Code: The Basics

Excel's macro recorder translates your actions into VBA code. To start the macro recorder, choose **Tools • Macro • Record New Macro**. Excel displays the Record Macro dialog box, shown in Figure 35-3.

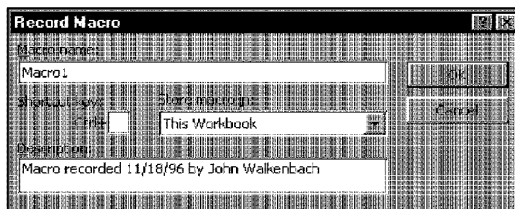


Figure 35-3: The Record Macro dialog box.

The Record Macro dialog box presents several options:

- **Macro name:** The name of the macro. By default, Excel proposes names such as Macro1, Macro2, and so on.
- **Shortcut key:** You can specify a key combination that executes the macro. You can also press Shift when you enter a letter. For example, pressing Shift while you enter the letter H makes the shortcut key combination Ctrl+Shift+H.
- **Store macro in:** The location for the macro. Your choices are the current workbook, your Personal Macro Workbook (described later in this chapter), or a new workbook.
- **Description:** A description of the macro. By default, Excel inserts the date and your name. You can add additional information if you like.

To begin recording your actions, click **OK**. Excel displays the Stop Recording toolbar, which contains two buttons: Stop Recording and Relative Reference. After you finish recording the macro, choose **Tools • Macro • Stop Recording** (or click the Stop Recording button on the toolbar).



Recording your actions always results in a new subroutine procedure. You can't create a function procedure by using the macro recorder. Function procedures must be created manually.

Recording a Macro: An Example

This example demonstrates how to record a macro that changes the formatting for the current range selection. The macro makes the selected range use Arial 16-point type, boldface, and the color red. To create the macro, follow these steps:

1. Enter a value or text into a cell—anything is okay. This gives you something to start with.
2. Select the cell that contains the value or text that you entered in the preceding step.
3. Select **Tools • Macro • Record New Macro**. Excel displays the Record Macro dialog box.
4. Enter a new name for the macro, to replace the default Macro1 name. A good name is **FormattingMacro**.
5. Assign this macro to the shortcut key Ctrl+Shift+F by entering **F** in the edit box labeled Shortcut key.
6. Click OK. This closes the Record Macro dialog box. Excel displays a toolbar called Stop Recording.
7. Select **Format • Cells** and then click the Font tab. Choose Arial font, Bold, and 16-point type, and make the color red. Click OK to close the Format Cells dialog box.
8. The macro is finished, so click the Stop Recording button on the Stop Recording toolbar (or select **Tools • Macro • Stop Recording**).

Examining the Macro

The macro was recorded in a new module named Module1. To view the code in this module, you must activate the Visual Basic Editor (VBE). You can activate the VBE in either of two ways:

- Press Alt+F11
- Choose **Tools • Macro • Visual Basic Editor**

Figure 35-4 shows the VBE window. Although the module is stored in the Excel workbook, you can view the module only in the VBE window.

The Project window displays a list of all open workbooks and add-ins. This list is displayed as a tree diagram, which can be expanded or collapsed. The code that you recorded previously is stored in Module1 in the current workbook. When you double-click Module1, the code in the module is displayed in the Code window.

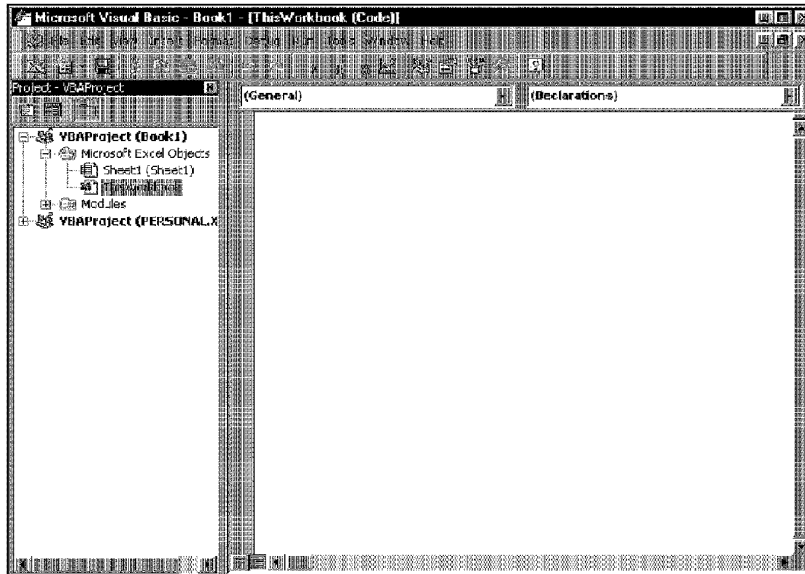


Figure 35-4: The VBE window.

Figure 35-5 shows the recorded macro, as displayed in the Code window.

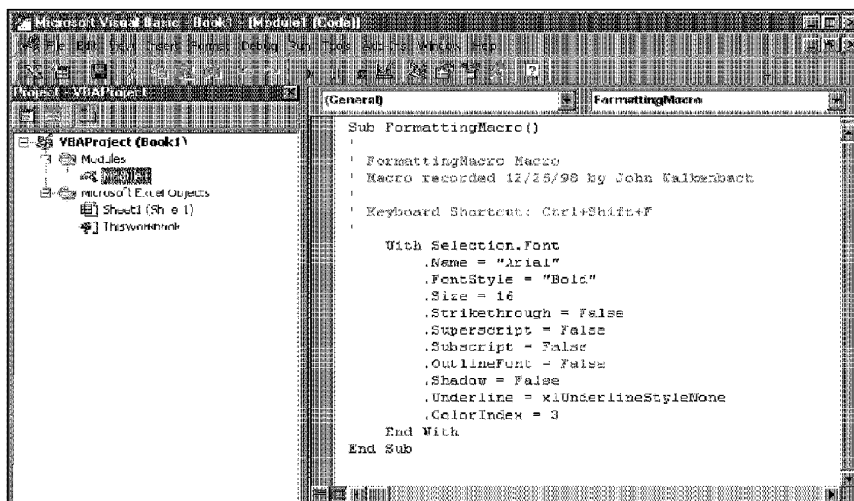


Figure 35-5: The FormattingMacro subroutine was generated by Excel's macro recorder.

Activate the module and examine the macro. It should consist of the following code:

```
Sub FormattingMacro Macro()  
.  
' FormatCells Macro  
' Macro recorded by John Walkenbach  
.  
' Keyboard Shortcut: Ctrl+Shift+F  
.  
    With Selection.Font  
        .Name = "Arial"  
        .FontStyle = "Bold"  
        .Size = 16  
        .Strikethrough = False  
        .Superscript = False  
        .Subscript = False  
        .OutlineFont = False  
        .Shadow = False  
        .Underline = xlUnderlineStyleNone  
        .ColorIndex = 3  
    End With  
End Sub
```

The macro recorded is a subroutine (it begins with a Sub statement) that is named `FormattingMacro`. The statements tell Excel what to do when the macro is executed.

Notice that Excel inserted some comments at the top of the subroutine. This is the information that appeared in the Record Macro dialog box. These comment lines (which begin with an apostrophe) aren't really necessary, and deleting them has no effect on how the macro runs.



Note You may notice that the macro recorded some actions that you didn't take. For example, it sets the `Strikethrough`, `Superscript`, and `Subscript` properties to `False`. This is just a byproduct of the method that Excel uses to translate actions into code. Excel sets the properties for every option in the Font tab of the Format Cells dialog box, even though you didn't change all of them.

Testing the Macro

Before you recorded this macro, you set an option that assigned the macro to the `Ctrl+Shift+F` shortcut key combination. To test the macro, return to Excel by using either of the following methods:

- Press `Alt+F11`
- Click the View Microsoft Excel button on the VBE toolbar

When Excel is active, activate a worksheet (it can be in the workbook that contains the VBA module or in any other workbook). Select a cell or range, and press `Ctrl+Shift+F`. The macro immediately changes the formatting of the selected cell(s).

Continue testing the macro with other selections. You'll find that the macro always applies exactly the same formatting.



In the preceding example, notice that you selected the cell to be formatted *before* you started recording your macro. This is important. If you select a cell while the macro recorder is turned on, the actual cell that you selected will be recorded into the macro. In such a case, the macro would always format that particular cell, and it would not be a "general-purpose" macro.

Editing the Macro

After you record a macro, you can change it (although you must know what you're doing). Assume that you discover that you really want to make the text 14 point rather than 16 point. You could rerecord the macro, but this is a simple modification, so editing the code is more efficient. Just activate Module1, locate the statement that sets the font size, and change 16 to 14. You can also remove the following lines:

```
.Strikethrough = False  
.Superscript = False  
.Subscript = False  
.OutlineFont = False  
.Shadow = False  
.Underline = xlNone
```

Removing these lines causes the macro to ignore the properties that are referred to in the statements. For example, if the cell has underlining, the underlining isn't affected by the macro.

The edited macro appears as follows:

```
Sub FormattingMacro()  
  With Selection.Font  
    .Name = "Arial"  
    .FontStyle = "Bold"  
    .Size = 14  
    .ColorIndex = 3  
  End With  
End Sub
```

Test this new macro, and you see that it performs as it should. Also, notice that it doesn't remove a cell's underlining, which occurred in the original version of the macro.

Another Example

This example shows you how to record a slightly more complicated VBA macro that converts formulas into values. Converting formulas into values is usually a two-step process in Excel:

1. Copy the range to the Clipboard.
2. Choose Edit • Paste Special (with the Values option selected) to paste the values over the formulas.

This macro combines these steps into a single command.

Furthermore, you want to be able to access this command by pressing a shortcut key combination (Ctrl+Shift+V). Take the following steps to create this macro:

1. Enter a formula into a cell. Any formula will do.
2. Select the cell that contains the formula.
3. Choose Tools • Macro • Record New Macro. Excel displays the Record Macro dialog box.
4. Complete the New Macro dialog box so that it looks like Figure 35-6. This assigns the macro the name **FormulaConvert**. It also gives it a Ctrl+Shift+V shortcut key.

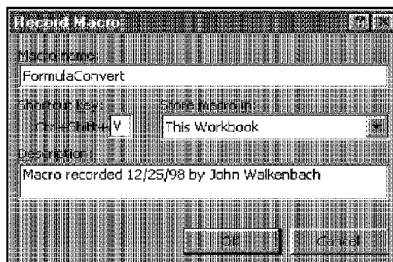


Figure 35-6: How the Record Macro dialog box should look when recording the sample macro.

5. Click OK to begin recording.
6. With the range still selected, choose Edit • Copy to copy the range to the Clipboard.
7. Select Edit • Paste Special, click the Values option, and then click OK to close the dialog box.
8. Press Esc to cancel Paste mode. (Excel removes the moving border around the selected range.)
9. Click the Stop Recording button (or choose Tools • Macro • Stop Recording).

To test the macro, activate a worksheet, enter some formulas, and then select the formulas. You can execute the macro in two ways:

- Press Ctrl+Shift+V

- Choose Tools • Macro • Macros command and double-click the macro name (FormulaConvert)

Excel converts the formulas in the selected range to their values—in a single step instead of two.



Caution

Be careful when you use this macro, because you can't undo the conversion of formulas to values. Actually, you can edit the macro so that its results can be undone, but the procedure is beyond the scope of this discussion.



Note

The shortcut key combination (Ctrl+Shift+V) is valid only when the workbook is open. When you close the workbook, pressing Ctrl+Shift+V has no effect.

The recorded macro appears as follows:

```

' FormulaConvert Macro
' Macro recorded by John Walkenbach
'
' Keyboard Shortcut: Ctrl+Shift+V
'
Sub ConvertFormulas()
    Selection.Copy
    Selection.PasteSpecial Paste:=xlValues, Operation:=xlNone, _
        SkipBlanks:=False, Transpose:=False
    Application.CutCopyMode = False
End Sub

```

Again, Excel added some comment lines that describe the macro. The actual macro begins with the Sub statement. The subroutine has three statements. The first simply copies the selected range. The second statement, which is displayed on two lines (the underscore character means that the statement continues on the next line), pastes the Clipboard contents to the current selection. The second statement has several arguments, representing the options in the Paste Special dialog box. The third statement cancels the moving border around the selected range. (I generated the statement by pressing Esc after the paste operation.)

If you prefer, you can delete the underscore character in the second statement and combine the two lines into one (a VBA statement can be any length). This action may make the macro easier to read.

More About Recording VBA Macros

If you followed along with the preceding examples, you should have a better feel for how to record macros. If you find the VBA code confusing, don't worry—you don't really have to be concerned with it as long as the macro that you record works correctly. If the macro doesn't work, rerecording the macro rather than editing the code often is easier.

A good way to learn about what gets recorded is to set up your screen so that you can see the code that is being generated in the Visual Basic Editor windows. Figure 35-7 shows an example of such a setup. While you're recording your actions, make sure that the VBE window is displaying the module in which the code is being recorded (you may have to double-click the module name in the Project window).

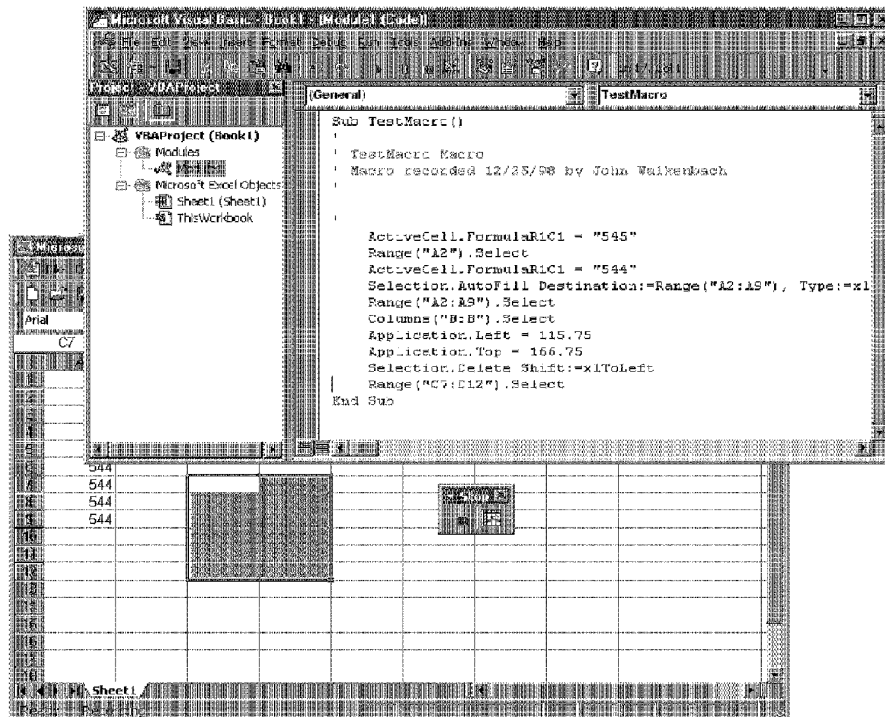


Figure 35-7: This window arrangement lets you see the VBA code as you record your actions.

Tip

If you want to view the code as it's being recorded, using a high-resolution video display really helps, such as 1024 x 768. Otherwise, you may find that fitting the windows of both Excel and VBE onscreen is very difficult.

Absolute Versus Relative Recording

If you're going to work with macros, you need to understand the concept of *relative* versus *absolute* recording. Normally, when you record a macro, Excel stores exact references to the cells that you select (that is, it performs *absolute* recording). If you

select the range B1:B10 while you're recording a macro, for example, Excel records this selection as

```
Range("B1:B10").Select
```

This means exactly what it says: "Select the cells in the range B1:B10." When you invoke this macro, the same cells are always selected, regardless of where the active cell is located.

You may have noticed that the Stop Recording toolbar has a tool named Relative Recording. When you click this tool while recording a macro, Excel changes its recording mode from absolute (the default) to relative. When recording in relative mode, selecting a range of cells is translated differently, depending on where the active cell is located. For example, if you're recording in relative mode and cell A1 is active, selecting the range B1:B10 generates the following statement:

```
ActiveCell.Offset(0, 1).Range("A1:A10").Select
```

This statement can be translated as "From the active cell, move 0 rows and 1 column, and then treat this new cell as if it were cell A1. Now select what would be A1:A10." In other words, a macro that is recorded in relative mode starts out by using the active cell as its base and then stores relative references to this cell. As a result, you get different results, depending on the location of the active cell. When you replay this macro, the cells that are selected depend on the active cell. It selects a range that is 10 rows by 1 column, offset from the active cell by 0 rows and 1 column.

When Excel is recording in relative mode, the Relative Reference toolbar button appears depressed. To return to absolute recording, click the Relative Reference button again (and it displays its normal, undepressed state).



Note

The recording mode – either absolute or relative – can make a *major* difference in how your macro performs. Therefore, understanding the distinction is important.



In previous version of Excel, recording commands such as Shift+Ctrl+right-arrow key or Shift+Ctrl+down-arrow key (commands that extend the selection to the end of a block of cells) were not recorded correctly. The macro recorder always recorded the exact cells that were selected. The problem is fixed in Excel 2000, so recording these types of selection commands produces macros that work properly.

Storing Macros in the Personal Macro Workbook

Most macros that are created by users are designed for use in a specific workbook, but you may want to use some macros in all of your work. You can store these general-purpose macros in the Personal Macro Workbook, so that they are always available to you. The Personal Macro Workbook is loaded whenever you start Excel. The file, `personal.xls`, is stored in the XlStart folder, which is in your Excel folder. This file doesn't exist until you record a macro, using Personal Macro Workbook as the destination.



The Personal Macro Workbook normally is in a hidden window (to keep it out of the way).

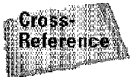
To record the macro in your Personal Macro Workbook, select the Personal Macro Workbook option in the Record Macro dialog box before you start recording.

If you store macros in the Personal Macro Workbook, you don't have to remember to open the Personal Macro Workbook when you load a workbook that uses macros. When you want to exit, Excel asks whether you want to save changes to the Personal Macro Workbook.

Assigning a Macro to a Toolbar Button

When you record a macro, you can assign it to a shortcut key combination. After you record the macro and test it, you may want to assign the macro to a toolbar button. You can follow these steps to do so:

1. If the macro is a general-purpose macro that you plan to use in more than one workbook, make sure that the macro is stored in your Personal Macro Workbook.
2. Select View • Toolbars • Customize. Excel displays its Customize dialog box.
3. Click the Toolbars tab in the Customize dialog box and make sure that the toolbar is visible that is to contain the new button.
4. Click the Commands tab in the Customize dialog box.
5. Click the Macros category.
6. In the Commands list, drag the Custom Button icon to the toolbar.
7. Right-click the toolbar button and select Assign Macro from the shortcut menu. Excel displays its Assign Macro dialog box.
8. Select the macro name from the list and click OK.
9. At this point, you can right-click the button again to change its name and button image.
10. Click Close to exit the Customize dialog box.



See Chapter 33 for details about customizing toolbars.

Writing VBA Code

As demonstrated in the preceding sections, the easiest way to create a simple macro is to record your actions. To develop more complex macros, however, you have to enter the VBA code manually—in other words, *write a program*. To save time, you can often combine recording with manual code entry.

Before you can begin writing VBA code, you must have a good understanding of topics such as objects, properties, and methods—and it doesn't hurt to be familiar with common programming constructs, such as looping and If-Then statements.

This section is an introduction to VBA programming, which is essential if you want to write (rather than record) VBA macros. This is not intended to be a complete instructional guide. My book titled *Excel 2000 Power Programming with VBA* (IDG Books Worldwide, Inc.) covers all aspects of VBA and advanced spreadsheet application development.

The Basics: Entering and Editing Code

Before you can enter code, you must insert a module into the workbook. If the workbook already has a module sheet, you can use the existing module sheet for your new code.

Use the following steps to insert a new module:

1. Press Alt+F11 to activate the Visual Basic Editor window. The Visual Basic Editor window is a separate application, although it works very closely with Excel.
2. The Project window displays a list of all open workbooks and add-ins. Locate the workbook that you are currently working in, and select it (see Figure 35-8).
3. Choose Insert • Module. VBA inserts a new (empty) module into the workbook and displays it in the Code window.

A VBA module, which is displayed in a separate window, works like a text editor. You can move through the sheet, select text, insert, copy, cut, paste, and so on.

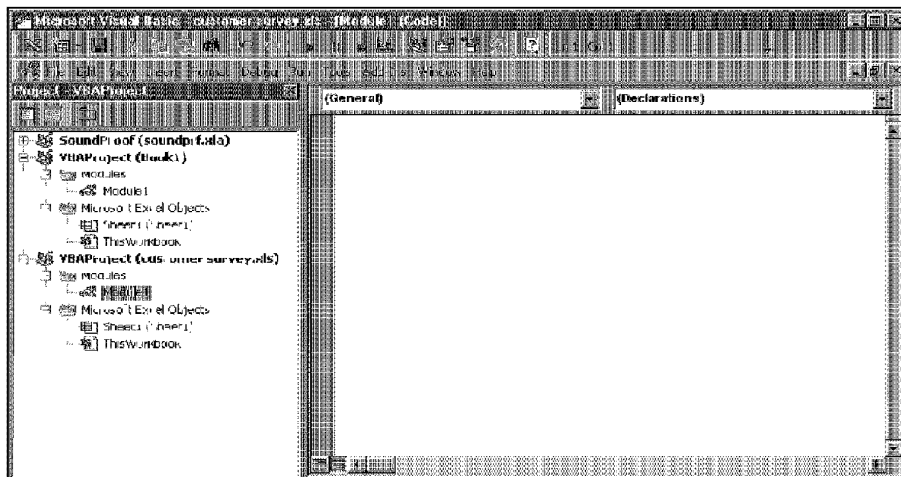


Figure 35-8: The Project window displays all open workbooks and add-ins.

VBA Coding Tips

When you enter code in a module sheet, you're free to use indenting and blank lines to make the code more readable (in fact, this is an excellent habit).

After you enter a line of code (by pressing Enter), it is evaluated for syntax errors. If none are found, the line of code is reformatted, and colors are added to keywords and identifiers. This automatic reformatting adds consistent spaces (before and after an equal sign, for example) and removes extra spaces that aren't needed. If a syntax error is found, you receive a pop-up message, and the line is displayed in a different color (red, by default). You need to correct your error before you can execute the macro.

A single statement can be as long as needed. However, you might want to break the statement into two or more lines. To do so, insert a space followed by an underscore (_). The following code, although written as two lines, is actually a single VBA statement:

```
Sheets("Sheet1").Range("B1").Value =  
    Sheets("Sheet1").Range("A1").Value
```

You also can put two or more statements in a single line. You do this by using a colon (:) to separate the statements. The following line consists of three statements:

```
x = 4: y = 6: z = 12
```

You can insert comments freely into your VBA code. The comment indicator is an apostrophe ('). Any text that follows a single quote is ignored. A comment can be a line by itself, or inserted after a statement. The following examples show two comments:

```
' Assign the values to the variables  
Rate = .085 'Rate as of November 16
```

How VBA Works

VBA is by far the most complex feature in Excel, and you can easily get overwhelmed. To set the stage for the details of VBA, here is a concise summary of how VBA works:

- You perform actions in VBA by writing (or recording) code in a VBA module sheet and then executing the macro in any one of various ways. VBA modules are stored in an Excel workbook, and a workbook can hold any number of VBA modules. To view or edit a VBA module, you must activate the Visual Basic Editor window (press Alt+F11 to toggle between Excel and the VBE window).
- A VBA module consists of subroutine procedures. A *subroutine procedure* is basically computer code that performs some action with objects. The following is an example of a simple subroutine called ShowSum (it adds 1 + 1 and displays the result):

```
Sub ShowSum()
    Sum = 1 + 1
    MsgBox "The answer is " & Sum
End Sub
```

- A VBA module also can store function procedures. A *function procedure* performs some calculations and returns a single value. A function can be called from another VBA procedure or can even be used in a worksheet formula. Here's an example of a function named `AddTwo` (it adds two values, which are supplied as arguments):

```
Function AddTwo(arg1, arg2)
    AddTwo = arg1 + arg2
End Function
```

- VBA manipulates objects. Excel provides well over 100 objects that you can manipulate. Examples of objects include a workbook, a worksheet, a range on a worksheet, a chart, and a drawn rectangle.
- Objects are arranged in a hierarchy, and can act as containers for other objects. For example, Excel itself is an object called `Application`, and it contains other objects such as `Workbook` objects. The `Workbook` object can contain other objects, such as `Worksheet` objects and `Chart` objects. A `Worksheet` object can contain objects such as `Range` objects, `PivotTable` objects, and so on. The arrangement of these objects is referred to as an *object model*. Excel's object model is depicted in the online Help system (see Figure 35-9).
- Objects that are alike form a *collection*. For example, the `Worksheets` collection consists of all worksheets in a particular workbook. The `CommandBars` collection consists of all `CommandBar` objects (that is, menu bars and toolbars). Collections are objects in themselves.
- You refer to an object in your VBA code by specifying its position in the object hierarchy, using a period as a separator.

For example, you can refer to a workbook named `Book1.xls` as

```
Application.Workbooks("Book1")
```

This refers to the `Book1.xls` workbook in the `Workbooks` collection. The `Workbooks` collection is contained in the `Application` object (that is, Excel). Extending this to another level, you can refer to `Sheet1` in `Book1` as follows:

```
Application.Workbooks("Book1").Worksheets("Sheet1")
```

You can take it to still another level and refer to a specific cell as follows:

```
Application.Workbooks("Book1").Worksheets("Sheet1").
Range("A1")
```

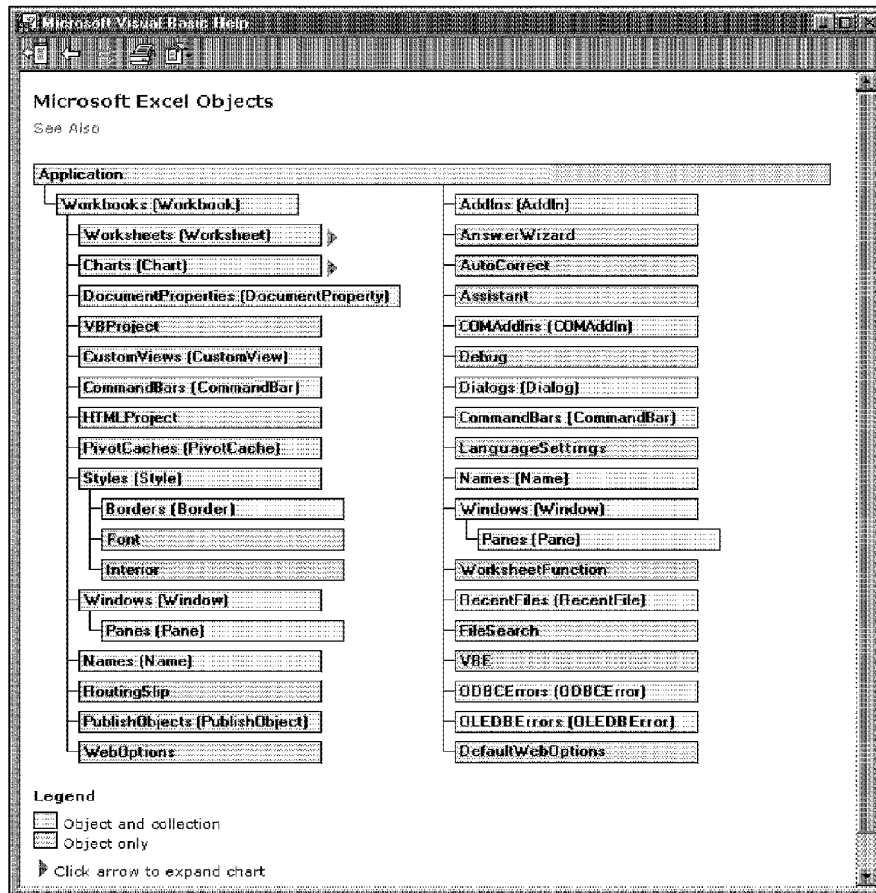


Figure 35-9: A depiction of part of Excel's object model.

- If you omit specific references, Excel uses the *active* objects. If Book1 is the active workbook, the preceding reference can be simplified as follows:
`Worksheets("Sheet1").Range("A1")`
 If you know that Sheet1 is the active sheet, you can simplify the reference even more:
`Range("A1")`
- Objects have properties. A *property* can be thought of as a setting for an object. For example, a range object has properties such as `Value` and `Name`. A chart object has properties such as `HasTitle` and `Type`. You can use VBA both to determine object properties and to change them.

- You refer to properties by combining the object with the property, separated by a period. For example, you can refer to the value in cell A1 on Sheet1 as follows:

```
Worksheets("Sheet1").Range("A1").Value
```

- You can assign values to variables. To assign the value in cell A1 on Sheet1 to a variable called Interest, use the following VBA statement:

```
Interest = Worksheets("Sheet1").Range("A1").Value
```

- Objects have methods. A *method* is an action that is performed with the object. For example, one of the methods for a range object is ClearContents. This method clears the contents of the range.
- You specify methods by combining the object with the method, separated by a period. For example, to clear the contents of cell A1, use the following statement:

```
Worksheets("Sheet1").Range("A1:C12").ClearContents
```

- VBA also includes all the constructs of modern programming languages, including arrays, looping, and so on.

Believe it or not, this describes VBA in a nutshell. Now you just have to learn the details, some of which are covered in the rest of this chapter.

Objects and Collections

VBA is an *object-oriented language*, which means that it manipulates *objects*, such as ranges, charts, drawing objects, and so on. These objects are arranged in a hierarchy. The Application object (which is Excel) contains other objects. For example, the Application object contains a number of objects, including the following:

- AddIns (a collection of AddIn objects)
- Windows (a collection of Window objects)
- WorksheetFunction
- Workbooks (a collection of Workbook objects)

Most of these objects can contain other objects. For example, a Workbook object can contain the following objects:

- Charts (a collection of Chart objects)
- Names (a collection of Name objects)
- Styles (a collection of Style objects)
- Windows (a collection of Window objects in the workbook)
- Worksheets (a collection of Worksheet objects)

Each of these objects, in turn, can contain other objects. A `Worksheet` object, for example, can contain the following objects:

- `ChartObjects` (a collection of all `ChartObject` objects)
- `PageSetup`
- `PivotTables` (a collection of all `PivotTable` objects)
- `Range`

A *collection* consists of all like objects. For example, the collection of all `Workbook` objects is known as the `Workbooks` collection. You can refer to an individual object in a collection by using an index number, or a reference. For example, if a workbook has three worksheets (named `Sheet1`, `Sheet2`, and `Sheet3`), you can refer to the first object in the `Worksheets` collection in either of these ways:

```
Worksheets(1)
Worksheets("Sheet1")
```

Properties

The objects that you work with have *properties*, which you can think of as attributes of the objects. For example, a range object has properties such as `Column`, `Row`, `Width`, and `Value`. A chart object has properties such as `Legend`, `ChartTitle`, and so on. `ChartTitle` is also an object, with properties such as `Font`, `Orientation`, and `Text`. Excel has many objects, and each has its own set of properties. You can write VBA code to do the following:

- Examine an object's current property setting and take some action based on it
- Change an object's property setting

You refer to a property in your VBA code by placing a period and the property name after the object's name. For example, the following VBA statement sets the `Value` property of a range named `frequency` to 15 (that is, it causes the number 15 to appear in the range's cells):

```
Range("frequency").Value = 15
```

Some properties are *read-only*, which means that you can examine the property, but you can't change the property. For a single-cell range object, the `Row` and `Column` properties are read-only properties: You can determine where a cell is located (in which row and column), but you can't change the cell's location by changing these properties.

A range object also has a `Formula` property, which is *not* read-only; that is, you can insert a formula into a cell by changing its `Formula` property. The following statement inserts a formula into a cell named `total` by changing the cell's `Formula` property:

```
Range("total").Formula = "=SUM(A1:A10)"
```


**Note**

Contrary to what you may think, Excel doesn't have a `Cell` object. When you want to manipulate a single cell, you use the `Range` object (with only one cell in it).

You need to be aware of the `Application` object, which is actually Excel, the program. The `Application` object has several useful properties:

- `Application.ActiveWorkbook`: Returns the active workbook (a `Workbook` object) in Excel.
- `Application.ActiveSheet`: Returns the active sheet (a `Sheet` object) of the active workbook.
- `Application.ActiveCell`: Returns the active cell (a `Range` object) object of the active window.
- `Application.Selection`: Returns the object that is currently selected in the active window of the `Application` object. This can be a range, a chart, a shape, or some other selectable object.

You also should understand that properties can return objects. In fact, that's exactly what the preceding examples do. The result of `Application.ActiveCell`, for example, is a `Range` object. Therefore, you can access properties by using a statement such as the following:

```
Application.ActiveCell.Font.Size = 15
```

In this case, `Application.ActiveCell.Font` is an object, and `Size` is a property of the object. The preceding statement sets the `Size` property to 15; that is, it causes the font in the currently selected cell to have a size of 15 points.

**Tip**

Because `Application` properties are so commonly used, you can omit the object qualifier (`Application`). For example, to get the row of the active cell, you can use a statement such as the following:

```
ActiveCell.Row
```

Many different ways to refer to the same object may exist. Assume that you have a workbook named `Sales.xls` and it's the only workbook open. Furthermore, assume that this workbook has one worksheet, named `Summary`. Your VBA code can refer to the `Summary` sheet in any of the following ways:

```
Workbooks("Sales.xls").Worksheets("Summary")  
Workbooks(1).Worksheets(1)  
Workbooks(1).Sheets(1)  
Application.ActiveWorkbook.ActiveSheet  
ActiveWorkbook.ActiveSheet  
ActiveSheet
```

The method that you use is determined by how much you know about the workspace. For example, if more than one workbook is open, the second or third method is not reliable. If you want to work with the active sheet (whatever it may be), either of the last three methods would work. To be absolutely sure that you're referring to a specific sheet on a specific workbook, the first method is your best choice.

Methods

Objects also have *methods*. You can think of a method as an action taken with an object. For example, range objects have a `Clear` method. The following VBA statement clears the range named `total`, an action that is equivalent to selecting the range and then choosing `Edit • Clear • All`:

```
Range("total").Clear
```

In VBA code, methods *look* like properties, because they are connected to the object with a "dot." However, methods and properties are different concepts.

Variables

Like all programming languages, VBA enables you to work with variables. In VBA (unlike in some languages), you don't need to declare variables explicitly before you use them in your code (although it's definitely a good practice).

In the following example, the value in cell `A1` on `Sheet1` is assigned to a variable named `rate`:

```
rate = Worksheets("Sheet1").Range("A1").Value
```

You then can work with the variable `rate` in other parts of your VBA code. Note that the variable `rate` is not a named range, which means that you can't use it as such in a worksheet formula.

Controlling Execution

VBA uses many constructs that are found in most other programming languages. These constructs are used to control the flow of execution. This section introduces a few of the more common programming constructs.

The If-Then construct

One of the most important control structures in VBA is the `If-Then` construct. This common command gives your applications decision-making capability. The basic syntax of the `If-Then` structure is as follows:

```
If condition Then statements [Else elsestatements]
```

The following is an example (which doesn't use the optional `Else` clause). This subroutine checks the active cell. If it contains a negative value, the cell's color is changed to red. Otherwise, nothing happens.

```
Sub CheckCell()  
    If ActiveCell.Value < 0 Then ActiveCell.Font.ColorIndex = 3  
End Sub
```

For-Next loops

For example, you can use a `For-Next` loop to process a series of items. Its syntax is as follows:

```
For counter = start To end [Step stepval]  
    [statements]  
    [Exit For]  
    [statements]  
Next [counter]
```

The following is an example of a `For-Next` loop:

```
Sub SumSquared()  
    Total = 0  
    For Num = 1 To 10  
        Total = Total + (Num ^ 2)  
    Next Num  
    MsgBox Total  
End Sub
```

This example has one statement between the `For` statement and the `Next` statement. This single statement is executed ten times. The variable `Num` takes on successive values of 1, 2, 3, and so on, up to 10. The variable `Total` stores the sum of `Num` squared, added to the previous value of `Total`. The result is a value that represents the sum of the first ten integers squared. This result is displayed in a message box.

The With-End With construct

Another construct that you encounter if you record macros is the `With-End With` construct. This is a shortcut way of dealing with several properties or methods of the same object. The following is an example:

```
Sub AlignCells()  
    With Selection  
        .HorizontalAlignment = xlCenter  
        .VerticalAlignment = xlCenter  
        .WrapText = False  
        .Orientation = xlHorizontal  
    End With  
End Sub
```

The following subroutine performs exactly the same operations, but doesn't use the With-End With construct:

```
Sub AlignCells()  
    Selection.HorizontalAlignment = xlCenter  
    Selection.VerticalAlignment = xlCenter  
    Selection.WrapText = False  
    Selection.Orientation = xlHorizontal  
End Sub
```

The Select Case construct

The Select Case construct is useful for choosing among two or more options. The syntax for the Select Case structure is as follows:

```
Select Case testexpression  
    [Case expressionlist-n  
        [statements-n]] . . .  
    [Case Else  
        [elasticsearchments]]  
End Select
```

The following example demonstrates the use of a Select Case construct. In this example, the active cell is checked. If its value is less than 0, it's colored red. If it's equal to 0, it's colored blue. If the value is greater than 0, it's colored black.

```
Sub CheckCell()  
    Select Case ActiveCell.Value  
        Case Is < 0  
            ActiveCell.Font.ColorIndex = 3 'Red  
        Case 0  
            ActiveCell.Font.ColorIndex = 5 'Blue  
        Case Is > 0  
            ActiveCell.Font.ColorIndex = 1 'Black  
    End Select  
End Sub
```

Any number of statements can go below each Case statement, and they all get executed if the case is true. If you use only one statement, as in the preceding example, you may want to put the statement on the same line as the Case statement.

A Macro That Can't Be Recorded

The following is a VBA macro that can't be recorded, because it uses an If-Then structure. This macro enables you to identify quickly cells that exceed a certain value. When you run this macro, it prompts the user for a value and then evaluates every cell in the selection. If the cell's value is greater than the value that is entered by the user, the macro makes the cell bold and red.

```
Sub SelectiveFormat()  
'This procedure selectively shades cells greater than  
'a specified target value  
'Get target value from user  
Message = "Change attributes of values greater than or  
equal_to..."  
Target = InputBox(Message)  
Target=Val(Target)  
  
'Evaluate each cell in the selection  
For Each Item In Selection  
If IsNumeric(Item) Then  
If Item.Value >= Target Then  
With Item  
.Font.Bold = True  
.Font.ColorIndex = 3 'Red  
End With  
End If  
End If  
Next Item  
End Sub
```

Although this macro may look complicated, it's fairly simple when you break it down.

First, the macro assigns text to a variable named `Message`. It then uses the `InputBox` function to solicit a value from the user. The `InputBox` function has a single argument (which is the `Message` variable), and returns a string—which is assigned to the `Target` variable. Next, the `Val` function is used to convert this string to a value.

The `For-Next` loop checks every cell in the selected range. The first statement within the loop uses the `IsNumeric` function to determine whether the cell can be evaluated as a number. This is important, because a cell without a value would generate an error when the `Value` property is accessed in the next statement. If the cell is numeric, it is checked against the target value. If it's greater than or equal to the target value, the `Bold` and `ColorIndex` properties are changed. Otherwise, nothing happens and the loop is incremented.

After entering this macro, named `SelectiveFormat`, into a module sheet, you can provide a shortcut key to access it. Choose `Tools • Macro • Macros` to display the `Macros` dialog box. Select the macro from the list, and click `Options`. Excel displays a new dialog box (see Figure 35-10) that enables you to specify a shortcut key combination to execute the macro.

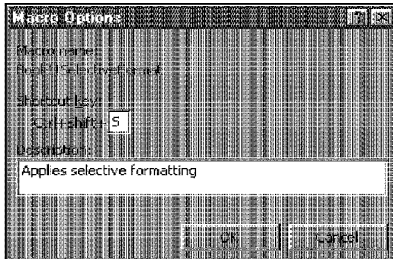


Figure 35-10: You can execute this macro by pressing Ctrl+S.

Figure 35-11 shows the macro in action. Note that you must select the range before you execute the macro.

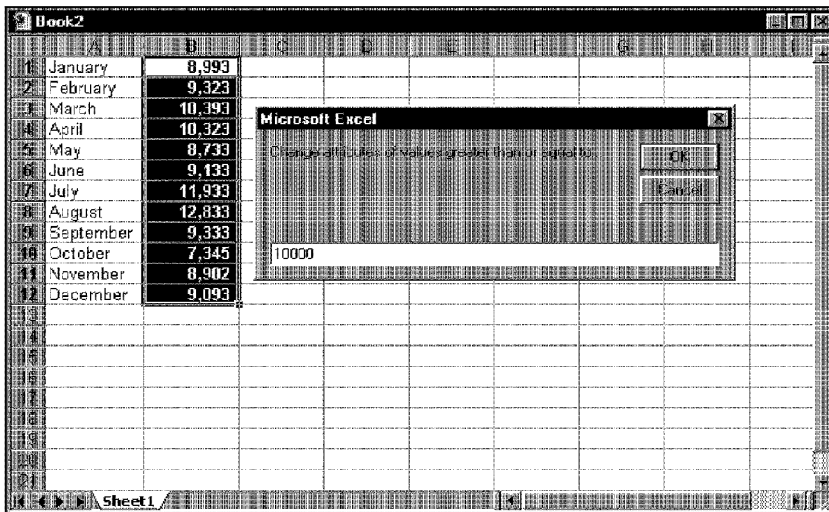


Figure 35-11: The macro uses the `InputBox` function to prompt the user for a value.

As macros go, this example is not very good. It's not very flexible and doesn't include any error handling. For example, if a nonrange object (such as a graphic object) is selected, the macro halts and displays an error message. To avoid this error message and abort the macro if anything except a range is selected, you can insert the following statement as the first statement in the procedure (directly below the `Sub` statement):

```
If TypeName(Selection) <> "Range" Then Exit Sub
```

This causes the macro to halt if the selection is not a Range object.

Notice also that the macro is executed even if you click Cancel in the input box. To avoid this problem, enter the following statement directly above the `Target=Val(Target)` statement:

```
If Target = "" then Exit Sub
```

This aborts the subroutine if `Target` is empty.

A much more versatile version of this utility is part of the Power Utility Pak (see Figure 35-12). The shareware version is available from this book's Web site.

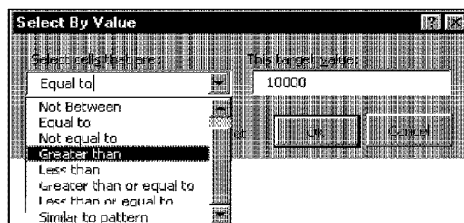


Figure 35-12: The Select By Value utility in the Power Utility Pak is a more versatile version of this macro.

Learning More

This chapter barely scratches the surface of what you can do with VBA. If this is your first exposure to VBA, you're probably a bit overwhelmed by objects, properties, and methods. I don't blame you. If you try to access a property that an object doesn't have, you get a run-time error, and your VBA code grinds to a screeching halt until you correct the problem. Fortunately, several good ways are available to learn about objects, properties, and methods.

Read the Rest of the Book

This book has three more chapters that are devoted to VBA. Chapter 36 covers VBA functions, Chapter 37 describes custom dialog boxes, and Chapter 38 consists of useful (and informative) VBA examples.

Record Your Actions

The best way — without question — to become familiar with VBA is to turn on the macro recorder and record actions that you make in Excel. This learning technique is even better if the VBA module in which the code is being recorded is visible while you're recording.

Use the Online Help System

The main source of detailed information about Excel's objects, methods, and procedures is the online Help system. Help is very thorough and easy to access. When you're in a VBA module, just move the cursor to a property or method and press F1. You get help that describes the word that is under the cursor.

Buy Another Book

Okay, I promise. This is the last plug for my other book, *Excel 2000 Power Programming With VBA*. I've received feedback from hundreds of previous-edition readers who claim that it's the best Excel/VBA book available. You be the judge.

Summary

This chapter introduces VBA, one of two macro languages included with Excel. If you want to learn macro programming, VBA is the language to use. In this chapter, you learn that a VBA module can contain subroutine procedures and function procedures, and that VBA is based on objects, properties, and methods. You also learn how to use the macro recorder to translate your actions into VBA code and write simple code directly in a VBA module. Three other chapters in this book provide additional information about VBA.

•

Creating Custom Worksheet Functions

As mentioned in the preceding chapter, you can create two types of VBA procedures: subroutines and functions. This chapter focuses on function procedures.

Overview of VBA Functions

Function procedures that you write in VBA are quite versatile. You can use these functions in two situations:

- As part of an expression in a different VBA procedure
- On formulas that you create in a worksheet

In fact, you can use a function procedure anywhere that you can use an Excel worksheet function or a VBA built-in function. Custom functions also appear in the Paste Function dialog box, so they appear to be part of Excel.

Excel contains hundreds of predefined worksheet functions. With so many from which to choose, you may be curious as to why anyone would need to develop additional functions. The main reason is that creating a custom function can greatly simplify your formulas by making them shorter—and shorter formulas are more readable and easier to work with. For example, you can often replace a complex formula with a single function. Another reason is that you can write functions to perform operations that would otherwise be impossible.



Note

This chapter assumes that you are familiar with entering and editing VBA code in the Visual Basic Editor (VBE). Refer to Chapter 35 for an overview of the VBE.

36 CHAPTER

In This Chapter

Overview of VBA Functions

An Introductory Example

About Function Procedures

Function Procedure Arguments

Debugging Custom Functions

Pasting Custom Functions

Learning More

An Introductory Example

The process of creating custom functions is relatively easy, once you understand VBA. Without further ado, here's an example of a VBA function procedure. This function is stored in a VBA module, which is accessible from the VBE.

A Custom Function

This example function, named `NumSign`, uses one argument. The function returns a text string of `Positive` if its argument is greater than zero, `Negative` if the argument is less than zero, and `Zero` if the argument is equal to zero. The function is shown in Figure 36-1.

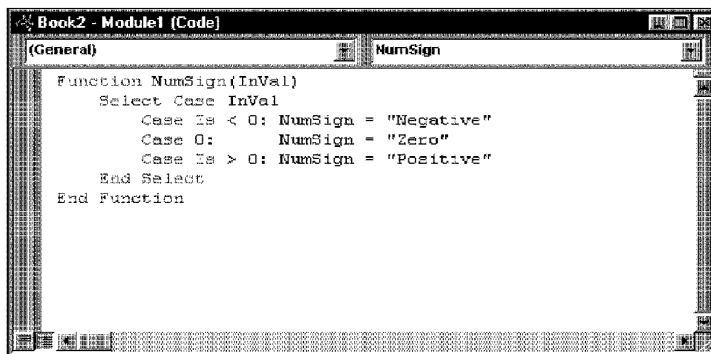


Figure 36-1: A custom function.

You could, of course, accomplish the same effect with the following worksheet formula, which uses a nested IF function:

```
=IF(A1=0,"Zero",IF(A1>0,"Positive","Negative"))
```

Many would agree that the custom function solution is easier to understand and to edit than the worksheet formula.

Using the Function in a Worksheet

When you enter a formula that uses the `NumSign` function, Excel executes the function to get the result (see Figure 36-2). This custom function works just like any built-in worksheet function. You can insert it in a formula by using the **Insert • Function** command, which displays the **Paste Function** dialog box (custom functions are located in the **User Defined** category). You also can nest custom functions and combine them with other elements in your formulas.

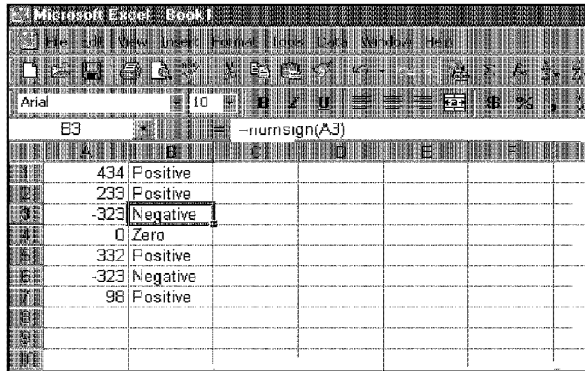


Figure 36-2: Using a custom function in a worksheet formula.

Using the Function in a VBA Subroutine

The following VBA subroutine procedure, which is defined in the same module as the custom `NumSign` function, uses the built-in `MsgBox` function to display the result of the `NumSign` function:

```
Sub ShowSign()
    CellValue = ActiveCell.Value
    MsgBox NumSign(CellValue)
End Sub
```

In this example, the variable `CellValue` contains the value in the active cell (this variable could contain any value, not necessarily obtained from a cell). `CellValue` is then passed to the function as its argument. Figure 36-3 shows the result of executing the `NumSign` subroutine.

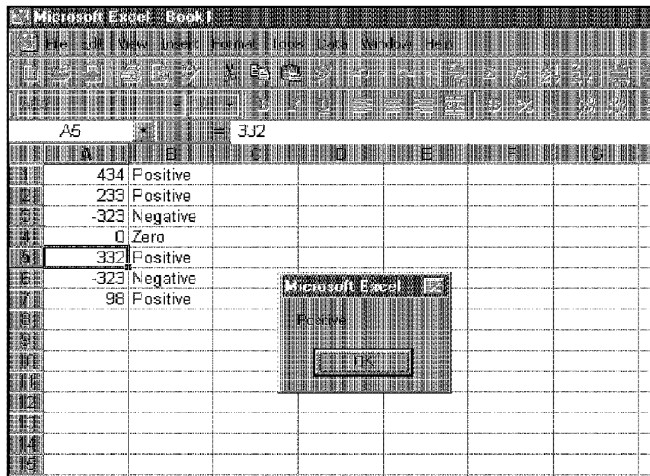


Figure 36-3: Using a custom function in a VBA subroutine.

Analyzing the Custom Function

This section describes the NumSign function. Here again is the code:

```
Function NumSign(InVal)
    Select Case InVal
        Case Is < 0: NumSign = "Negative"
        Case 0: NumSign = "Zero"
        Case Is > 0: NumSign = "Positive"
    End Select
End Function
```

Notice that the procedure starts with the keyword `Function` rather than `Sub`, followed by the name of the function (`NumSign`). This custom function uses one argument (`InVal`); the argument's name is enclosed in parentheses. `InVal` is the cell or variable that is to be processed. When the function is used in a worksheet, the argument can be a cell reference (such as `A1`) or a literal value (such as `-123`). When the function is used in another procedure, the argument can be a numeric variable, a literal number, or a value that is obtained from a cell.

The `NumSign` function uses the `Select Case` construct (described in Chapter 35) to take a different action, depending on the value of `InVal`. If `InVal` is less than zero, `NumSign` is assigned the text `Negative`. If `InVal` is equal to zero, `NumSign` is `Zero`. If `InVal` is greater than zero, `NumSign` is `Positive`. The value returned by a function is always assigned to the function's name.

The procedure ends with an `End Function` statement.

About Function Procedures

A custom function procedure has a lot in common with a subroutine procedure, covered in the preceding chapter. Function procedures have some important differences, however, which are discussed in this section.

Declaring a Function

The syntax for declaring a function is as follows:

```
[Public | Private][Static] Function name [(arglist)] [As type]  
  [statements]  
  [name = expression]  
  [Exit Function]  
  [statements]  
  [name = expression]  
End Function
```

These elements are defined as follows:

- **Public:** Indicates that the function is accessible to all other procedures in all other modules in the workbook. (Optional)
- **Private:** Indicates that the function is accessible only to other procedures in the same module. Private functions can't be used in worksheet formulas and do not appear in the Paste Function dialog box. (Optional)
- **Static:** Indicates that the values of variables declared in the function are preserved between calls, rather than being reset. (Optional)
- **Function:** A keyword that indicates the beginning of a function procedure. (Required)
- **name:** Any valid variable name. When the function finishes, the single-value result is assigned to the function's name. (Required)
- **arglist:** A list (one or more) of variables that represent arguments passed to the function. The arguments are enclosed in parentheses. Use a comma to separate arguments. (Optional)
- **type:** The data type that is returned by the function. (Optional)
- **statements:** Valid VBA statements. (Optional)
- **Exit Function:** A statement that causes an immediate exit from the function. (Optional)
- **End Function:** A keyword that indicates the end of the function. (Required)

Keep in mind that a value is assigned to the function's name when a function is finished executing.

To create a custom function, follow these steps:

1. Activate the Visual Basic Editor (or press Alt+F11).
2. Select the workbook in the Project window.
3. Choose Insert • Module to insert a VBA module (or you can use an existing module).
4. Enter the keyword `Function` followed by the function's name and a list of the arguments (if any) in parentheses.
5. Insert the VBA code that performs the work—and make sure that the variable corresponding to the function's name has the appropriate value (this is the value that the function returns).
6. End the function with an `End Function` statement.

Function names must adhere to the same rules as variable names, and you can't use a name that looks like a worksheet cell (for example, a function named J21 isn't accepted).

What a Function Can't Do

Almost everyone who starts creating custom worksheet functions using VBA makes a fatal mistake: They try to get the function to do more than is possible.

A worksheet function returns a value, and it must be completely "passive." In other words, the function cannot change anything on the worksheet. For example, it's impossible to develop a worksheet function that changes the formatting of a cell (every VBA programmer has tried this, and not one of them has been successful!). If your function attempts to perform an action that is not allowed, the function simply returns an error.

VBA functions that are not used in worksheet formulas can do anything that a regular subroutine can do—including changing cell formatting.

Executing Function Procedures

Although many ways exist to execute a *subroutine* procedure, you can execute a *function* procedure in just two ways:

- Call it from another procedure
- Use it in a worksheet formula

Calling custom functions from a procedure

You can call custom functions from a procedure just as you call built-in VBA functions. For example, after you define a function called `CalcTax`, you can enter a statement such as the following:

```
Tax = CalcTax(Amount, Rate)
```

This statement executes the `CalcTax` custom function with `Amount` and `Rate` as its arguments. The function's result is assigned to the `Tax` variable.

Using custom functions in a worksheet formula

Using a custom function in a worksheet formula is like using built-in functions. You must ensure that Excel can locate the function procedure, however. If the function procedure is in the same workbook, you don't have to do anything special. If the function is defined in a different workbook, you may have to tell Excel where to find the function. The following are the three ways in which you can do this:

- **Precede the function's name with a file reference.** For example, if you want to use a function called `CountNames` that's defined in a workbook named `MyFunctions`, you can use a reference such as the following:

```
=MyFunctions.xls!CountNames(A1:A1000)
```

If you insert the function with the Paste Function dialog box, the workbook reference is inserted automatically.

- **Set up a reference to the workbook.** If the custom function is defined in a reference workbook, you don't need to precede the function name with the workbook name. You establish a reference to another workbook with the Tools • References command (in the Visual Basic Editor). You are presented with a list of references that includes all open workbooks. Place a check mark in the item that refers to the workbook that contains the custom function (use the Browse button if the workbook isn't open).

Create an add-in. When you create an add-in from a workbook that has function procedures, you don't need to use the file reference when you use one of the functions in a formula; the add-in must be installed, however. Chapter 40 discusses add-ins.

**Note**

If you plan on developing custom worksheet functions, make sure that you heed the warning in the sidebar, "What a Function Can't Do."

Your function procedures don't appear in the Macros dialog box when you select Tools • Macro, because you can't execute a function directly. As a result, you need to do extra, up-front work to test your functions as you're developing them. One approach is to set up a simple subroutine that calls the function. If the function is designed to be used in worksheet formulas, you can enter a simple formula to test it as you're developing the function.

Function Procedure Arguments

Keep in mind the following about function procedure arguments:

- Arguments can be variables (including arrays), constants, literals, or expressions.
- Some functions do not have arguments.
- Some functions have a fixed number of required arguments (from 1 to 60).
- Some functions have a combination of required and optional arguments.

The following section presents a series of examples that demonstrate how to use arguments effectively with functions. Coverage of optional arguments is beyond the scope of this book.

Example: A Function with No Argument

Like subroutines, functions don't necessarily have to use arguments. Excel, for example, has a few built-in worksheet functions that don't use arguments. These include RAND, TODAY, and NOW.

The following is a simple example of a function that has no arguments. This function returns the `UserName` property of the Application object, which is the name that appears in the Options dialog box (General tab). This example is simple, but it can be useful, because no other way is available to get the user's name to appear in a worksheet formula.

```
Function User()  
    ' Returns the name of the current user  
    User = Application.UserName  
End Function
```

When you enter the following formula into a worksheet cell, the cell displays the name of the current user:

```
=User()
```

As with Excel's built-in functions, when you use a function with no arguments, you must include a set of empty parentheses.

The following example is a simple subroutine that uses the `User` custom function as an argument for the `MsgBox` function. The concatenation operator (&) joins the literal string with the result of the `User` function.

```
Sub ShowUser()  
    MsgBox ("The user is " & User())  
End Sub
```


Example: A Function with One Argument

This section contains a more complex function that is designed for a sales manager who needs to calculate the commissions that are earned by the sales force. The commission rate is based on the amount sold—those who sell more earn a higher commission rate. The function returns the commission amount, based on the sales made (which is the function's only argument—a required argument). The calculations in this example are based on the following table:

<i>Monthly Sales</i>	<i>Commission Rate</i>
0 – \$9,999	8.0%
\$10,000 – \$19,999	10.5%
\$20,000 – \$39,999	12.0%
\$40,000+	14.0%

Several ways exist to calculate commissions for various sales amounts that are entered into a worksheet. You could write a formula such as the following:

```
=IF(AND(A1>=0,A1<=9999.99),A1*0.08,IF(AND(A1>=10000,A1<=19999.99),A1*0.105,IF(AND(A1>=20000,A1<=39999.99),A1*0.12,IF(A1>=40000,A1*0.14,0))))
```

This is not the best approach, for a couple of reasons. First, the formula is overly complex and difficult to understand. Second, the values are hard coded into the formula, making the formula difficult to modify if the commission structure changes.

A better approach is to use a lookup table function to compute the commissions; for example:

```
=VLOOKUP(A1,Table,2)*A1
```

Using the VLOOKUP function requires that you have a table of commission rates set up in your worksheet.

An even better approach is to create a custom function, such as the following:

```
Function Commission(Sales)
' Calculates sales commissions
Tier1 = 0.08
Tier2 = 0.105
Tier3 = 0.12
Tier4 = 0.14
Select Case Sales
    Case 0 To 9999.99: Commission = Sales * Tier1
    Case 1000 To 19999.99: Commission = Sales * Tier2
    Case 20000 To 39999.99: Commission = Sales * Tier3
    Case Is >= 40000: Commission = Sales * Tier4
End Select
End Function
```

After you define the `Commission` function in a VBA module, you can use it in a worksheet formula or call it from other VBA procedures.

Entering the following formula into a cell produces a result of 3,000 (the amount, 25,000, qualifies for a commission rate of 12 percent):

```
=Commission(25000)
```

Even if you don't need custom functions in a worksheet, creating function procedures can make your VBA coding much simpler. If your VBA procedure calculates sales commissions, for example, you can use the `Commission` function and call it from a VBA subroutine. The following is a tiny subroutine that asks the user for a sales amount and then uses the `Commission` function to calculate the commission due and to display it:

```
Sub CalcComm()
    Sales = InputBox("Enter Sales:")
    MsgBox "The commission is " & Commission(Sales)
End Sub
```

The subroutine starts by displaying an input box that asks for the sales amount. Then, the procedure displays a message box with the calculated sales commission for that amount. The `Commission` function must be available in the active workbook; otherwise, Excel displays a message saying that the function is not defined.

Example: A Function with Two Arguments

This example builds on the previous one. Imagine that the sales manager implements a new policy: The total commission paid is increased by one percent for every year that the salesperson has been with the company. For this example, the custom `Commission` function (defined in the preceding section) has been modified so that it takes two arguments — both of which are required arguments. Call this new function `Commission2`:

```

Function Commission2(Sales, Years)
    ' Calculates sales commissions based on years in service
    Tier1 = 0.08
    Tier2 = 0.105
    Tier3 = 0.12
    Tier4 = 0.14
    Select Case Sales
        Case 0 To 9999.99: Commission2 = Sales * Tier1
        Case 1000 To 19999.99: Commission2 = Sales * Tier2
        Case 20000 To 39999.99: Commission2 = Sales * Tier3
        Case Is >= 40000: Commission2 = Sales * Tier4
    End Select
    Commission2 = Commission2 + (Commission2 * Years / 100)
End Function

```

The modification was quite simple. The second argument (Years) was added to the Function statement and an additional computation was included that adjusts the commission, before exiting the function.

The following is an example of how you write a formula by using this function (it assumes that the sales amount is in cell A1, and the number of years that the salesperson has worked is in cell B1):

```
=Commission2(A1,B1)
```

Example: A Function with a Range Argument

The example in this section demonstrates how to use a worksheet range as an argument. Actually, it's not at all tricky; Excel takes care of the details behind the scenes.

Assume that you want to calculate the average of the five largest values in a range named Data. Excel doesn't have a function that can do this, so you can write the following formula:

```
=(LARGE(Data,1)+LARGE(Data,2)+LARGE(Data,3)+LARGE(Data,4)+LARGE(Data,5))/5
```

This formula uses Excel's LARGE function, which returns the *n*th largest value in a range. The preceding formula adds the five largest values in the range named Data and then divides the result by 5. The formula works fine, but it's rather unwieldy. And, what if you need to compute the average of the top six values? You would need to rewrite the formula—and make sure that all copies of the formula also get updated.

Wouldn't it be easier if Excel had a function named TopAvg? For example, you could use the following (nonexistent) function to compute the average:

```
=TopAvg(Data,5)
```

This is an example of when a custom function can make things much easier for you. The following is a custom VBA function, named `TopAvg`, which returns the average of the top n values in a range:

```
Function TopAvg(InRange, Num)
    ' Returns the average of the highest Num values in InRange
    Sum = 0
    For i = 1 To Num
        Sum = Sum + WorksheetFunction.Large(InRange, i)
    Next i
    TopAvg = Sum / Num
End Function
```

This function takes two arguments: `InRange` (which is a worksheet range) and `Num` (the number of values to average). The code starts by initializing the `Sum` variable to 0. It then uses a `For-Next` loop to calculate the sum of the n th largest values in the range. Note that Excel's `LARGE` function is used within the loop. You can use an Excel worksheet function in VBA if you precede the function with `WorksheetFunction` and a period. Finally, `TopAvg` is assigned the value of `Sum` divided by `Num`.

You can use all of Excel's worksheet functions in your VBA procedures, *except* those that have equivalents in VBA. For example, VBA has a `Rnd` function that returns a random number. Therefore, you can't use Excel's `RAND` function in a VBA procedure.

Debugging Custom Functions

Debugging a function procedure can be a bit more challenging than debugging a subroutine procedure. If you develop a function to use in worksheet formulas, an error in the function procedure simply results in an error display in the formula cell (usually `#VALUE!`). In other words, you don't receive the normal run-time error message that helps you to locate the offending statement.

When you are debugging a worksheet formula, using only one instance of the function in your worksheet is the best technique. The following are three methods that you may want to use in your debugging:

- **Place `MsgBox` functions at strategic locations to monitor the value of specific variables.** Fortunately, message boxes in function procedures pop up when the procedure is executed. But, make sure that you have only one formula in the worksheet that uses your function; otherwise, the message boxes appear for each formula that's evaluated.
- **Test the procedure by calling it from a subroutine procedure.** Run-time errors display normally, and you can either fix the problem (if you know what it is) or jump right into the debugger.
- **Set a breakpoint in the function and then use Excel's debugger to step through the function.** You then can access all the normal debugging tools.

Pasting Custom Functions

Excel's Paste Function dialog box is a handy tool that enables you to choose a worksheet function; you even can choose one of your custom worksheet functions. The Formula Palette prompts you for the function's arguments.

Function procedures that are defined with the `Private` keyword do not appear in the Paste Function dialog box.

You also can display a description of your custom function in the Paste Function dialog box. To do so, follow these steps:

1. Create the function in a module by using the VBE.
2. Activate Excel.
3. Choose the **Tools • Macro • Macros** command.

Excel displays its Macro dialog box (see Figure 36-4).

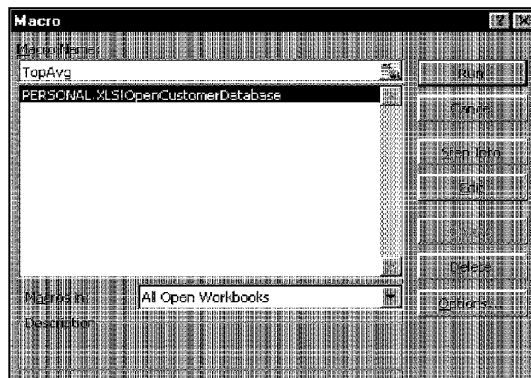


Figure 36-4: Excel's Macro dialog box doesn't list functions, so you must enter the function name yourself.

4. In the Macro dialog box, type the name of the function in the box labeled Macro Name. Notice that functions do not normally appear in this dialog box, so you must enter the function name yourself.
5. Click the Options button.

Excel displays its Macro Options dialog box. (See Figure 36-5.)

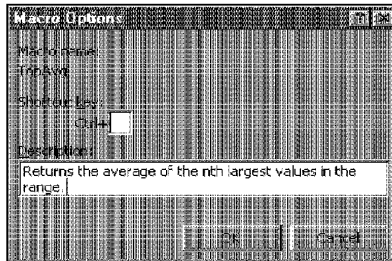


Figure 36-5: Entering a description for a custom function. This description appears in the Paste Function dialog box.

6. Enter a description of the function and then click OK. The Shortcut key field is irrelevant for functions.

The description that you enter appears in the Paste Function dialog box.

Custom functions are listed under the User Defined category, and no straightforward way exists to create a new function category for your custom functions.

Figure 36-6 shows the Paste Function dialog box, listing the custom functions that are in the User Defined category. In the second Function Wizard dialog box, the user is prompted to enter arguments for a custom function — just as in using a built-in worksheet function.

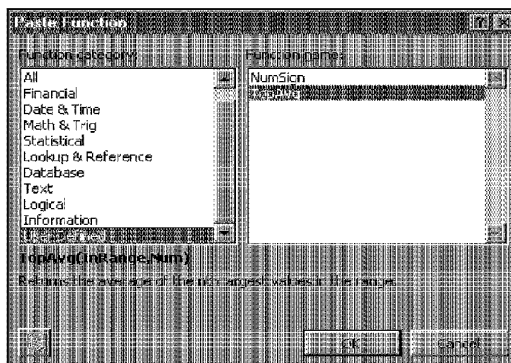


Figure 36-6: Using the Paste Function dialog box to insert a custom function.

When you access a *built-in* function from the Paste Function dialog box, the Formula Palette displays a description of each argument. Unfortunately, you can't provide such descriptions for custom functions.

Learning More

The information in this chapter only scratches the surface when it comes to creating custom functions. It should be enough to get you started, however, if you're interested in this topic. Refer to Chapter 38 for more examples of useful VBA functions. You may be able to use the examples directly or adapt them for your needs.

Summary

In this chapter, you read about how to create and use custom VBA functions. These functions can be used in worksheet formulas and in other VBA procedures. Several examples are provided, and you can refer to Chapter 38 for more examples.

•

Creating Custom Dialog Boxes

You can't use Excel very long without being exposed to dialog boxes. Excel, like most Windows programs, uses dialog boxes to obtain information, clarify commands, and display messages. If you develop VBA macros, you can create your own dialog boxes that work just like those that are built into Excel. This chapter introduces you to custom dialog boxes.



Note

Beginning with Excel 97, Microsoft introduced a new method for creating custom dialog boxes. Therefore, the information in this chapter does not apply to versions of Excel prior to Excel 97.

Why Create Custom Dialog Boxes?

Some macros that you create behave exactly the same every time that you execute them. For example, you may develop a macro that enters a list of your employees into a worksheet range. This macro always produces the same result and requires no additional user input. You may develop other macros, however, that you want to behave differently under different circumstances, or that offer some options for the user. In such cases, the macro may benefit from a custom dialog box.

The following is an example of a simple macro that makes each cell in the selected range uppercase (but it skips cells that have a formula). The subroutine uses VBA's built-in `StrConv` function.

```
Sub ChangeCase()  
    For Each cell In Selection  
        If Not cell.HasFormula Then  
            cell.Value = StrConv(cell.Value,  
vbUpperCase)  
        End If  
    Next cell  
End Sub
```

37

CHAPTER

In This Chapter

Why Create Custom Dialog Boxes?

Custom Dialog Box Alternatives

Creating Custom Dialog Boxes: An Overview

A Custom Dialog Box Example

Another Custom Dialog Box Example

More on Creating Custom Dialog Boxes

This macro is useful, but it could be even more useful. For example, the macro would be more helpful if it could also change the cells to lowercase or initial capitals (only the first letter of each word is uppercase). This modification is not difficult to make, but if you make this change to the macro, you need some method of asking the user what type of change to make to the cells. The solution is to present a dialog box like the one shown in Figure 37-1. This dialog box is a UserForm that was created by using the Visual Basic Editor, and it is displayed by a VBA macro.

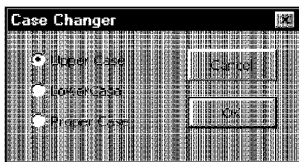


Figure 37-1: A custom dialog box that asks the user for an option.

Another solution would be to develop three macros—one for each type of text case change. Combining these three operations into a single macro and using a dialog box represents a more efficient approach, however. This example, including how to create the dialog box, is discussed later in the chapter.

Custom Dialog Box Alternatives

Although developing custom dialog boxes isn't difficult, sometimes using the tools that are built into VBA is easier. For example, VBA includes two functions (`InputBox` and `MsgBox`) that enable you to display simple dialog boxes, without having to create a UserForm in the VBE. These dialog boxes can be customized in some ways, but they certainly don't offer the options that are available in a custom dialog box.

The InputBox Function

The `InputBox` function is useful for obtaining a single input from the user. A simplified version of the function's syntax follows:

```
InputBox(prompt[,title][,default])
```

The elements are defined as follows:

- `prompt`: Text that is displayed in the input box. (Required)
- `title`: Text that appears in the input box's title bar. (Optional)
- `default`: The default value. (Optional)

The following is an example of how you can use the `InputBox` function:

```
Rate = InputBox("Commission rate?","Commission Worksheet")
```

When this VBA statement is executed, Excel displays the dialog box that is shown in Figure 37-2. Notice that this example uses only the first two arguments and does not supply a default value. When the user enters a value and clicks OK, the value is assigned to the variable Rate.

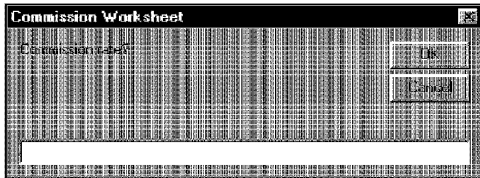


Figure 37-2: This dialog box is displayed by VBA's InputBox function.

VBA's InputBox function always returns a string, so you may need to convert the results to a value. You can use the Val function to convert a string to a value, as follows:

```
Rate = Val(InputBox("Commission rate?","Commission Worksheet"))
```

The MsgBox Function

VBA's MsgBox function is a handy way to display information and to solicit simple input from users. I use VBA's MsgBox function in many of this book's examples, to display a variable's value. A simplified version of the MsgBox syntax is as follows:

```
MsgBox(prompt[, buttons][, title])
```

The elements are defined as follows:

- **prompt:** Text that is displayed in the message box. (Required)
- **buttons:** The code for the buttons that are to appear in the message box. (Optional)
- **title:** Text that appears in the message box's title bar. (Optional)

You can use the MsgBox function by itself or assign its result to a variable. If you use it by itself, don't include parentheses around the arguments. The following example displays a message and does not return a result:

```
Sub MsgBoxDemo()  
    MsgBox "Click OK to continue"  
End Sub
```

Figure 37-3 shows how this message box appears.

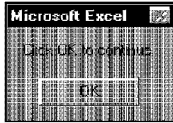


Figure 37-3: A simple message box, displayed with VBA's MsgBox function.

To get a response from a message box, you can assign the result of the `MsgBox` function to a variable. The following code uses some built-in constants (described later) to make it easier to work with the values that are returned by `MsgBox`:

```
Sub GetAnswer()
    Ans = MsgBox("Continue?", vbYesNo)
    Select Case Ans
        Case vbYes
            ' ...[code if Ans is Yes]...
        Case vbNo
            ' ...[code if Ans is No]...
    End Select
End Sub
```

When this procedure is executed, the `Ans` variable contains a value that corresponds to `vbYes` or `vbNo`. The `Select Case` statement determines the action to take based on the value of `Ans`.

You can easily customize your message boxes, because of the flexibility of the buttons argument. Table 37-1 lists the built-in constants that you can use for the button argument. You can specify which buttons to display, whether an icon appears, and which button is the default.

Table 37-1
Constants That Are Used in the MsgBox Function

<i>Constant</i>	<i>Value</i>	<i>Description</i>
<code>vbOKOnly</code>	0	Display OK button
<code>vbOKCancel</code>	1	Display OK and Cancel buttons
<code>vbAbortRetryIgnore</code>	2	Display Abort, Retry, and Ignore buttons
<code>vbYesNoCancel</code>	3	Display Yes, No, and Cancel buttons
<code>vbYesNo</code>	4	Display Yes and No buttons
<code>vbRetryCancel</code>	5	Display Retry and Cancel buttons
<code>vbCritical</code>	16	Display Critical Message icon

Constant	Value	Description
vbQuestion	32	Display Warning Query icon
vbExclamation	48	Display Warning Message icon
vbInformation	64	Display Information Message icon
vbDefaultButton1	0	First button is default
vbDefaultButton2	256	Second button is default
vbDefaultButton3	512	Third button is default
vbSystemModal	4096	System modal; all applications are suspended until the user responds to the message box

The following example uses a combination of constants to display a message box with a Yes button, a No button (vbYesNo), and a question mark icon (vbQuestion); the second button is designated as the default button (vbDefaultButton2)—which is the button that is executed if the user presses Enter. For simplicity, these constants are assigned to the Config variable and Config is then used as the second argument in the MsgBox function.

```
Sub GetAnswer()
    Config = vbYesNo + vbQuestion + vbDefaultButton2
    Ans = MsgBox("Process the monthly report?", Config)
    If Ans = vbYes Then RunReport
    If Ans = vbNo Then End
End Sub
```

Figure 37-4 shows how this message box appears when the GetAnswer subroutine is executed. If the user clicks the Yes button (or presses Enter), the routine executes the procedure named RunReport (which is not shown). If the user clicks the No button, the routine is ended with no action. Because the title argument was omitted in the MsgBox function, Excel uses the default title ("Microsoft Excel").



Figure 37-4: The second argument of the MsgBox function determines what appears in the message box.

The routine that follows is another example of using the MsgBox function:

```
Sub GetAnswer2()
    Msg = "Do you want to process the monthly report?"
    Msg = Msg & vbLf & vbLf
    Msg = Msg & "Processing the monthly report will take
approximately "
    Msg = Msg & "15 minutes. It will generate a 30-page report
for all "
    Msg = Msg & "sales offices for the current month."
    Title = "XYZ Marketing Company"
    Config = vbYesNo + vbQuestion
    Ans = MsgBox(Msg, Config, Title)
    If Ans = vbYes Then RunReport
    If Ans = vbNo Then End
End Sub
```

This example demonstrates an efficient way to specify a longer message in a message box. A variable (Msg) and the concatenation operator (&) are used to build the message in a series of statements. In the second statement, vbLf is a constant that represents a line feed character (using two line feeds inserts a blank line). The title argument is also used to display a different title in the message box. Figure 37-5 shows how this message box appears when the procedure is executed.



Figure 37-5: A message box with a longer message and a title.

Creating Custom Dialog Boxes: An Overview

The InputBox and MsgBox functions do just fine for many cases, but if you need to obtain more information, then you need to create a custom dialog box. A custom dialog box is created on a UserForm in the Visual Basic Editor.

The following is a list of the general steps that you typically take to create a custom dialog box:

1. Determine exactly how the dialog box is going to be used and where it is to fit into your VBA macro.
2. Activate the Visual Basic Editor and insert a new UserForm (select Insert • UserForm).

3. Add the appropriate controls to the dialog box.
4. Create a macro to display the dialog box.
5. Create “event-handler” VBA subroutines that are executed when the user manipulates the controls (for example, clicks the OK button).

The following sections provide more details on creating a custom dialog box.

Working with UserForms

Excel stores custom dialog boxes on UserForms (one dialog box per form). To create a dialog box, you must first insert a new UserForm in the Visual Basic Editor window.

To activate the Visual Basic Editor, select **Tools • Macro • Visual Basic Editor** (or press **Alt+F11**). Make sure that the current workbook is selected in the Project window and then select **Insert • UserForm**. The Visual Basic Editor displays an empty form, as shown in Figure 37-6. When you activate a form, the Visual Basic editor displays the Toolbox, which is used to add controls to the dialog box.

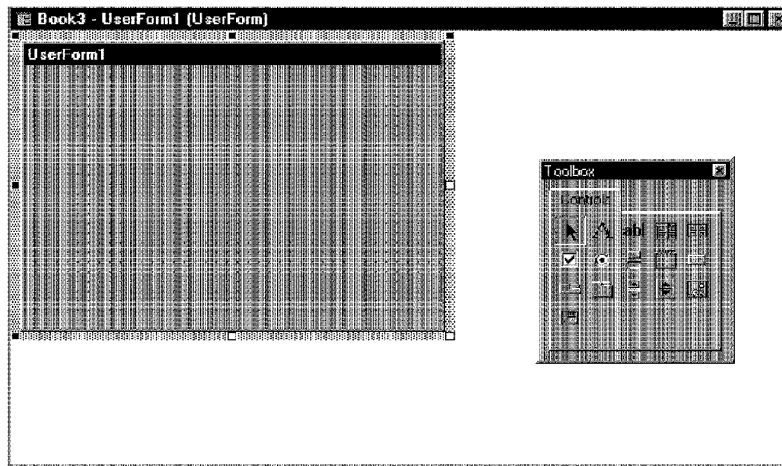


Figure 37-6: An empty form.

Adding Controls

The Toolbox, shown in Figure 37-7, contains various ActiveX controls that you can add to your dialog box.

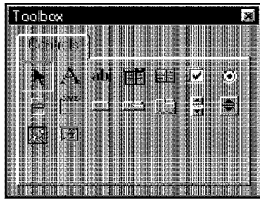


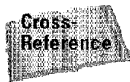
Figure 37-7: The Toolbox contains the controls that you add to your dialog box.

When you move the mouse pointer over a control in the Toolbox, the control's name is displayed. To add a control, click and drag it in the form. After adding a control, you can move it or change its size.

Table 37-2 lists the Toolbox controls.

Table 37-2
Toolbox Controls

Control	Description
Select Objects	Lets you select other controls by dragging
Label	Adds a label
TextBox	Adds a text box
ComboBox	Adds a combo box
ListBox	Adds a list box
CheckBox	Adds a check box
OptionButton	Adds an option button
ToggleButton	Adds a toggle button
Frame	Adds a frame (a container for other objects)
CommandButton	Adds a command button
TabStrip	Adds a tab strip
MultiPage	Adds a multipage control (a container for other objects)
ScrollBar	Adds a scrollbar
SpinButton	Adds a spin button
Image	Adds a control that can contain an image
RefEdit	Adds a reference edit control (lets the user select a range)



You can also place some of these controls directly on your worksheet. Refer to Chapter 38 for details.

Changing the Properties of a Control

Every control that you add to a UserForm has several properties that determine how the control looks and behaves. You can change some of these properties (such as `Height` and `Width`) by clicking and dragging the control's border. To change other properties, use the Properties window.

To display the Properties window, select `View • Properties Window` (or press F4). The Properties window displays a list of properties for the selected control (each control has a different set of properties). If you click the form itself, the Properties window displays properties for the form. Figure 37-8 shows the Properties window for a `CommandButton` control.

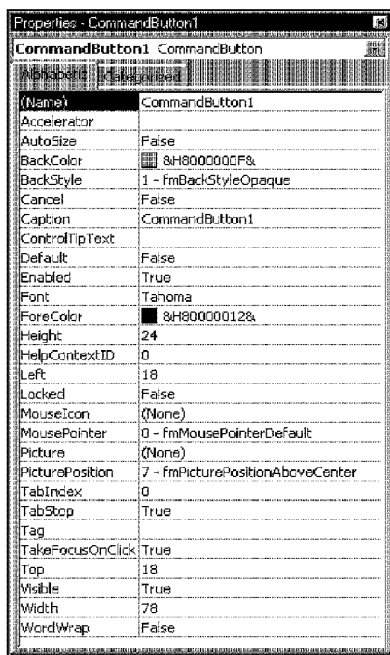


Figure 37-8: The Properties window for a `CommandButton` control.

To change a property, select the property in the Property window and then enter a new value. Some properties (such as `BackColor`) enable you to select a property from a list. The top of the Properties window contains a drop-down list that enables you to select a control to work with. You can also click a control to select it and display its properties.

When you set properties by using the Property window, you're setting properties at *design time*. You can also use VBA to change the properties of controls while the dialog box is displayed (that is, at *run time*).

A complete discussion of all the properties is well beyond the scope of this book. To find out about a particular property, select it in the Property window and press F1. The online Help for UserForm controls is extremely thorough.

Handling Events

When you insert a UserForm, that form can also hold VBA subroutines to handle the events that are generated by the form. An *event* is something that occurs when the user manipulates a control. For example, clicking a button is an event. Selecting an item in a list box control is an event. To make a dialog box useful, you must write VBA code to do something when an event occurs.

Event-handler subroutines have names that combine the control with the event. The general form is the control's name, followed by an underscore, and then the event name. For example, the subroutine that is executed when the user clicks a button named MyButton is MyButton_Click.

Displaying Custom Dialog Boxes

You also need to write a subroutine to display a custom dialog box. You use the Show method of the UserForm object. The following procedure displays the dialog box that is located on the UserForm1 form:

```
Sub ShowDialog()  
    UserForm1.Show  
End Sub
```

This subroutine should be stored in a regular VBA module (not the code module for the UserForm).

When this subroutine is executed, the dialog box is displayed. What happens next depends on the event-handler subroutines that you create.

A Custom Dialog Box Example

The preceding section is, admittedly, rudimentary. However, this section demonstrates how to develop a custom dialog box. This example is rather simple. The UserForm displays a message to the user—something that could be accomplished more easily by using the MsgBox function. However, the custom dialog box gives you a lot more flexibility in terms of formatting and layout of the message.

Creating the Dialog Box

If you're following along on your computer, start with a new workbook. Then, follow these steps:

1. Choose **Tools • Macro • Visual Basic Editor** (or press **Alt+F11**) to activate the VBE window.

2. In the VBE window, choose **Insert • UserForm**.

The VBE adds an empty form named **UserForm1** and displays the **Toolbox**.

3. Press **F4** to display the **Properties** window and then change the following properties of the **UserForm** object:

<i>Property</i>	<i>Change To</i>
Name	AboutBox
Caption	About This Workbook

4. Use the toolbar to add a **Label** object to the dialog box.
5. Select the **Label** object. In the **Properties** window, enter any text that you want for the label's **Caption**.
6. In the **Properties** window, click the **Font** property and adjust the font. You can change the typeface, size, and so on. The changes then appear in the form. **Figure 37-9** shows an example of a formatted **Label** control.

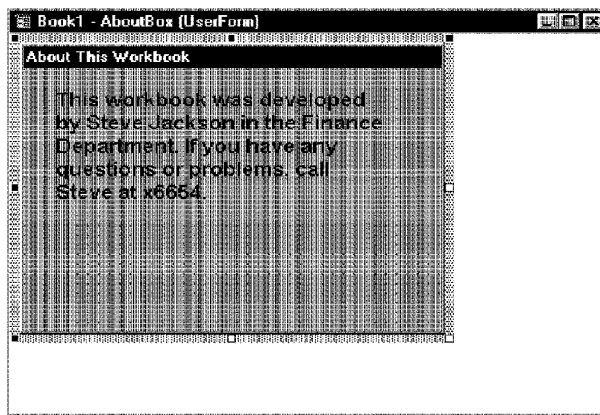


Figure 37-9: A **Label** control, after changing its **Font** properties.

7. Add a `CommandButton` object to the dialog box, and change the following properties for the `CommandButton`:

<i>Property</i>	<i>Change To</i>
Name	OKButton
Caption	OK
Default	True

8. Make other adjustments so that the form looks good to you. You can change the size of the form, or move or resize the controls.

Testing the Dialog Box

At this point, the dialog box has all the necessary controls. What's missing is a way to display the dialog box. This section explains how to write a VBA subroutine to display the custom dialog box.

1. Insert a module by selecting **Insert • Module**.
2. In the empty module, enter the following code:

```
Sub ShowAboutBox()  
    AboutBox.Show  
End Sub
```
3. Activate Excel.
4. Choose **Tools • Macro • Macros** (or press **Alt+F8**).
5. In the **Macros** dialog box, select `ShowAboutBox` from the list of macros and click **OK**.

The custom dialog box then appears.

If you click the **OK** button, notice that it doesn't close the dialog box as you may expect. This button needs to have an event-handler subroutine. You can dismiss the dialog box by clicking the close button in its title box.

Creating an Event-Handler Subroutine

An event-handler subroutine is executed when an event occurs. In this case, you need a subroutine to handle the `Click` event that's generated when the user clicks the `OK` button.

1. Activate the Visual Basic Editor (pressing **Alt+F11** is the fastest way).
2. Activate the `AboutBox` form by double-clicking its name in the Project window.
3. Double-click the `OKButton` control.

4. VBE activates the module for the UserForm and inserts some code, as shown in Figure 37-10.

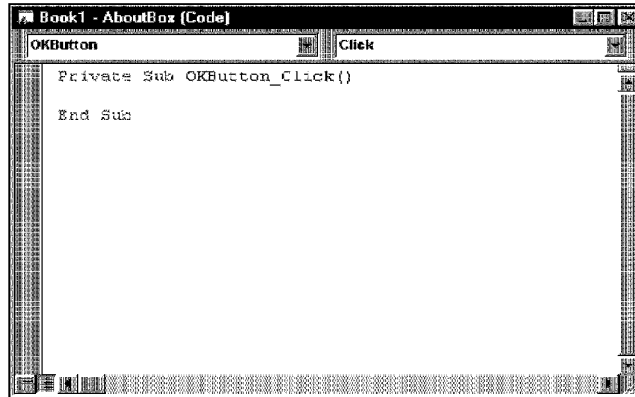


Figure 37-10: The module for the UserForm.

5. Insert the following statement before the End Sub statement:

```
Unload AboutBox
```

This statement simply dismisses the UserForm. The complete event-handler subroutine is listed below:

```
Private Sub OKButton_Click()  
    Unload AboutBox  
End Sub
```

Attaching the Macro to a Button

This section describes how to attach the `ShowAboutBox` subroutine to a Button object on a worksheet. Follow these steps:

1. Activate Excel.
2. Right-click any toolbar and select Forms from the shortcut menu.
The Forms toolbar is displayed.
3. Click the Button tool on the Forms toolbar.
4. Drag the Button tool into the worksheet to create a Button object.

When you release the mouse button, Excel displays its Assign Macro dialog box (see Figure 37-11).

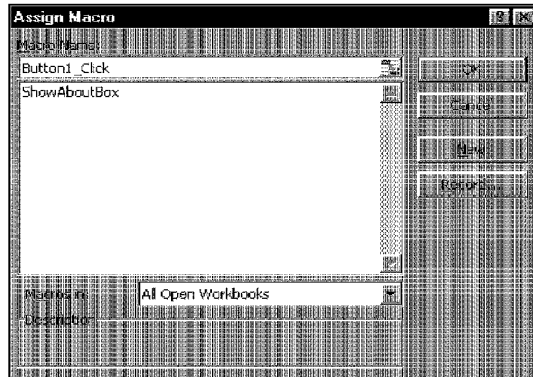


Figure 37-11: The Assign Macro dialog box.

5. Select the ShowAboutBox macro from the list.
6. Click OK to close the Assign Macro dialog box.
7. Change the caption of the button to **About...**

After you perform these steps, click the button to execute the ShowAboutBox subroutine — which displays your custom dialog box.

Another Custom Dialog Box Example

The example in this section is an enhanced version of the ChangeCase example presented at the beginning of the chapter. Recall that the original version of this macro changes the text in the selected cells to uppercase characters. This modified version asks the user what type of case change to make: uppercase, lowercase, or initial capitals.



This workbook is available on the companion CD-ROM.

Creating the Dialog Box

This dialog box needs one piece of information from the user: the type of change to make to the text. Because only one option can be selected, `OptionButton` controls are appropriate. Follow these steps to create the custom dialog box. Start with an empty workbook:

1. Choose **Tools • Macro • Visual Basic Editor** (or press **Alt-F11**) to activate the VBE window.
2. In the VBE window, choose **Insert • UserForm**.
VBE adds an empty form named `UserForm1` and displays the Toolbox.

3. Press F4 to display the Properties window and then change the following properties of the UserForm object:

<i>Property</i>	<i>Change To</i>
Name	CaseChangerDialog
Caption	Case Changer

4. Add a CommandButton object to the dialog box and then change the following properties for the CommandButton:

<i>Property</i>	<i>Change To</i>
Name	OKButton
Caption	OK
Default	True

5. Add another CommandButton object and then change the following properties:

<i>Property</i>	<i>Change To</i>
Name	CancelButton
Caption	Cancel
Cancel	True

6. Add an OptionButton control and then change the following properties (this option is the default, so its Value property should be set to True):

<i>Property</i>	<i>Change To</i>
Name	OptionUpper
Caption	Upper Case
Value	True

7. Add a second OptionButton control and then change the following properties:

<i>Property</i>	<i>Change To</i>
Name	OptionLower
Caption	Lower Case

8. Add a third OptionButton control and then change the following properties:

<i>Property</i>	<i>Change To</i>
Name	OptionProper
Caption	Proper Case

9. Adjust the size and position of the controls and the form until your screen resembles Figure 37-12. Make sure that the controls do not overlap.

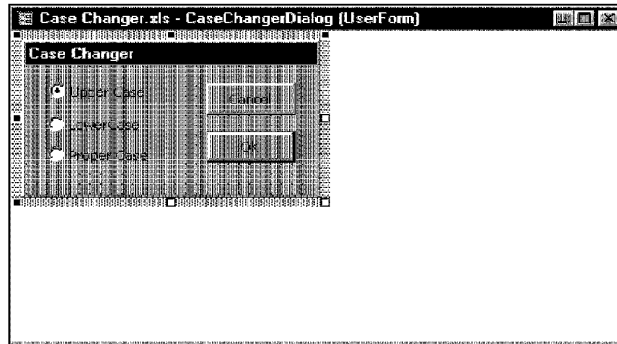


Figure 37-12: The dialog box after adding controls and adjusting some properties.



Tip

The Visual Basic Editor provides several useful commands to help you size and align the controls. Select the controls that you want to work with, and then choose a command from the Format menu. These commands are fairly self-explanatory, and the online Help has complete details.

Testing the Dialog Box

At this point, the dialog box has all the necessary controls. What's missing is a way to display the dialog box. This section explains how to write a VBA subroutine to display the custom dialog box. Make sure that the VBE window is activated.

1. Insert a module by selecting **Insert • Module**.
2. In the empty module, enter the following code:


```
Sub ChangeCase()
    CaseChangerDialog.Show
End Sub
```
3. Select **Run • Sub/UserForm** (or press F5).

The Excel window is then activated, and the new dialog box is displayed, as shown in Figure 37-13. The **OptionButtons** work, but clicking the **OK** and **Cancel** buttons has no effect. These two buttons need to have event-handler subroutines. Click the **Close** button in the title bar to dismiss the dialog box.

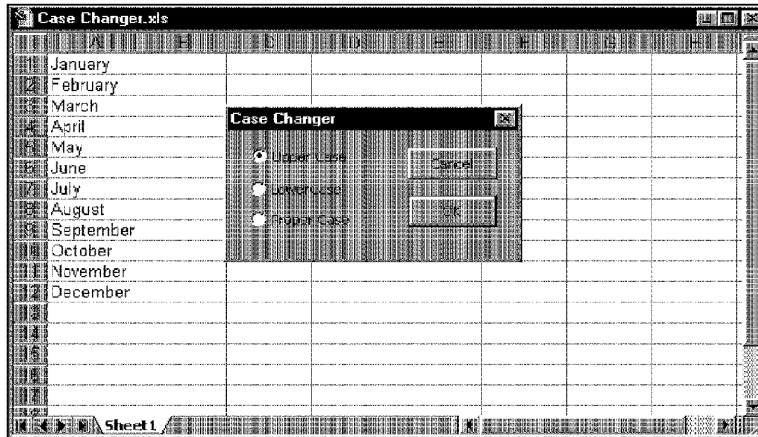


Figure 37-13: Displaying the custom dialog box.

Creating Event-Handler Subroutines

This section explains how to create two event-handler subroutines: one to handle the Click event for the CancelButton CommandButton and the other to handle the Click event for the OKButton CommandButton. Event handlers for the OptionButtons are not necessary. The VBA code can determine which of the three OptionButtons is selected.

Event-handler subroutines are stored in the form module. To create the subroutine to handle the Click event for the CancelButton, follow these steps:

1. Activate the CaseChangerDialog form by double-clicking its name in the Project window.
2. Double-click the CancelButton control.
3. VBE activates the module for the form and inserts some code, as shown in Figure 37-14.
4. Insert the following statement before the End Sub statement:
`Unload CaseChangerDialog`

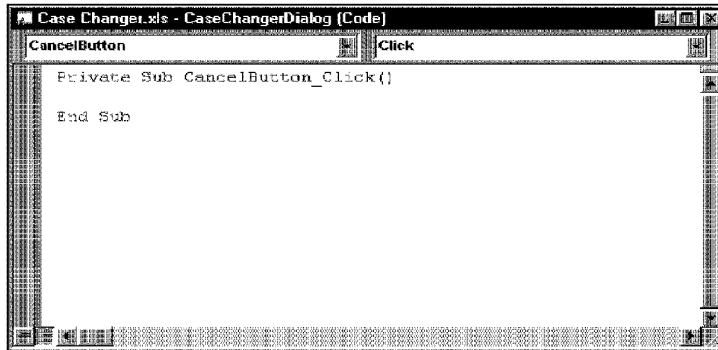


Figure 37-14: VBE sets up an empty subroutine to handle the Click event for the CancelButton control.

That's all there is to it. The following is a listing of the entire subroutine:

```
Private Sub CancelButton_Click()
    Unload CaseChangerDialog
End Sub
```

This subroutine is executed when the CancelButton is clicked. It consists of a single statement that unloads the CaseChangerDialog form.

The next step is to add the code to handle the Click event for the OKButton control. Follow these steps:

1. Select OKButton from the drop-down list at the top of the module. VBE begins a new subroutine called OKButton_Click.
2. Enter the following code (the first and last statements have already been entered for you by VBE):

```
Private Sub OKButton_Click()
    Application.ScreenUpdating = False
    ' Exit if a range is not selected
    If TypeName(Selection) <> "Range" Then Exit Sub
    ' Upper case
    If OptionUpper Then
        For Each cell In Selection
            If Not cell.HasFormula Then
                cell.Value = StrConv(cell.Value, vbUpperCase)
            End If
        Next cell
    End If
    ' Lower case
    If OptionLower Then
        For Each cell In Selection
            If Not cell.HasFormula Then
```

```
        cell.Value = StrConv(cell.Value, vbLowerCase)
    End If
    Next cell
End If
' Proper case
If OptionProper Then
    For Each cell In Selection
        If Not cell.HasFormula Then
            cell.Value = StrConv(cell.Value, bProperCase)
        End If
    Next cell
End If
Unload CaseChangerDialog
End Sub
```

The macro starts by turning off screen updating (this makes the macro run faster). Next, the code checks the type of the selection. If a range is not selected, the procedure ends. The remainder of the subroutine consists of three separate blocks. Only one block is executed, determined by which `OptionButton` is selected. The selected `OptionButton` has a value of `True`. Finally, the `UserForm` is unloaded (dismissed).

Testing the Dialog Box

To try out the dialog box, follow these steps:

1. Activate Excel.
2. Enter some text into some cells.
3. Select the range with the text.
4. Choose `Tools • Macro • Macros` (or press `Alt+F8`).
5. In the `Macros` dialog box, select `ChangeCase` from the list of macros and then click `OK`. The custom dialog box appears.
6. Make your choice, and click `OK`.

Try it with a few more selections. Notice that if you click `Cancel`, the dialog box is dismissed and no changes are made.

Making the Macro Available from a Toolbar Button

At this point, everything should be working properly. However, you have no quick and easy way to execute the macro. A good way to execute this macro would be from a toolbar button. You can use the following steps:

1. Right-click any toolbar, and select `Customize` from the shortcut menu.
Excel displays its `Customize` dialog box.

2. Click the **Commands** tab and then select **Macros** from the **Categories** list.
3. Click the **Custom Button** in the **Commands** list and drag it to a toolbar.
4. Right-click the new toolbar button and then select **Assign Macro** from the shortcut menu.
5. Choose **ChangeCase** from the list of macros, and click **OK**.

You can also change the button image and add a tool tip by using other commands that are on the shortcut menu.

6. Click **Close** to close the **Customize** dialog box.

After performing the preceding steps, clicking the toolbar button executes the macro and displays the dialog box.



If the workbook that contains the macro is not open already, it is opened. You may want to hide the workbook window (select **Window • Hide**) so that it isn't displayed. Another option is to create an add-in. See Chapter 40 for specifics.

More on Creating Custom Dialog Boxes

Creating custom dialog boxes can make your macros much more versatile. You can create custom commands that display dialog boxes that look exactly like those that Excel uses. This section contains some additional information to help you develop custom dialog boxes that work like those that are built into Excel.

Adding Accelerator Keys

Dialog boxes should not discriminate against those who want to use the keyboard rather than a mouse. All of Excel's dialog boxes work equally well with a mouse and a keyboard, because each control has an associated accelerator key. The user can press **Alt** plus the accelerator key to work with a specific dialog box control.

Adding accelerator keys to your custom dialog boxes is a good idea. You do this in the **Properties** window by entering a character for the **Accelerator** property.

Obviously, the letter that you enter as the accelerator key must be a letter that is contained in the caption of the object. It can be any letter in the text (not necessarily the first letter). You should make sure that an accelerator key is not duplicated in a dialog box. If you have duplicate accelerator keys, the accelerator key acts on the first control in the "tab order" of the dialog box (explained shortly).

Some controls (such as edit boxes) don't have a caption property. You can assign an accelerator key to a label that describes the control. Pressing the accelerator key then activates the next control in the tab order (which should be the edit box).

Controlling Tab Order

The previous section refers to a dialog box's *tab order*. When you're working with a dialog box, pressing Tab and Shift+Tab cycles through the dialog box's controls. When you create a custom dialog box, you should make sure that the tab order is correct. Usually, this means that tabbing should move to the controls in a logical sequence.

To view or change the tab order in a custom dialog box, use the Properties window. If the `TabStop` property is `True`, the selected control is selectable when the user clicks Tab. Change the value of the `TabIndex` property. These values range from 0 (first in the tab order) to 1 less than the number of controls that have a `TabIndex` property. When you change the `TabIndex`, VBE automatically adjusts the `TabIndex` of all subsequent controls in the tab order.

Learning More

Mastering custom dialog boxes takes practice. You should closely examine the dialog boxes that Excel uses; these are examples of well-designed dialog boxes. You can duplicate nearly every dialog box that Excel uses.

The best way to learn more about creating dialog boxes is by using the online Help system.

Summary

This chapter describes how to create dialog boxes and use them with your VBA macros. It also covers two VBA functions—`InputBox` and `MsgBox`—which can sometimes take the place of a custom dialog box. The chapter includes several examples to help you understand how to use this feature.

•

Using Dialog Box Controls in Your Worksheet

Chapter 37 presented an introduction to custom dialog boxes. If you like the idea of using dialog box controls — but don't like the idea of creating a dialog box — this chapter is for you. It explains how to enhance your worksheet with a variety of interactive controls, such as buttons, ListBoxes, and OptionButtons.

Why Use Controls on a Worksheet?

The main reason to use dialog box controls on a worksheet is to make it easier for the user to provide input. For example, if you create a model that uses one or more input cells, you can create controls to allow the user to select values for the input cells.

Adding controls to a worksheet requires much less effort than creating a dialog box. In addition, you may not have to create any macros, because you can link a control to a worksheet cell. For example, if you insert a CheckBox control on a worksheet, you can link it to a particular cell. When the CheckBox is selected, the linked cell displays TRUE. When the CheckBox is not selected, the linked cell displays FALSE.

Figure 38-1 shows a simple example that uses OptionButtons and a ScrollBar control.

38

CHAPTER

In This Chapter:

Why Use Controls on a Worksheet?

Using Controls

The Controls Toolbox Controls

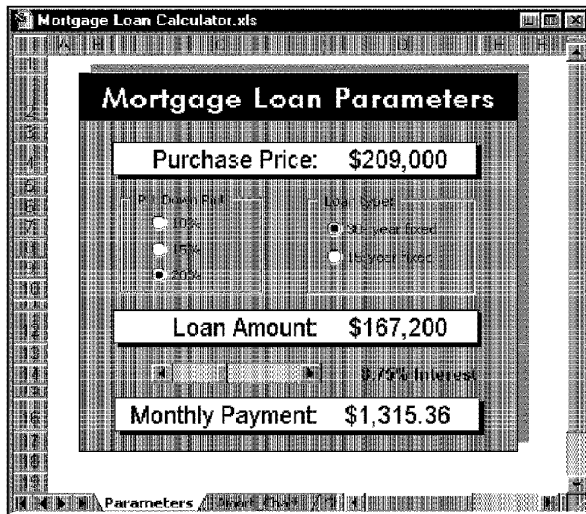


Figure 38-1: This worksheet uses dialog box controls.

Controls That Are Available to You

Adding controls to a worksheet can be a bit confusing, because these controls have two sources. The controls that you can insert on a worksheet come from two toolbars:

- **Forms toolbar:** These controls are insertable objects (and are compatible with Excel 5 and Excel 95).
- **Control Toolbox toolbar:** These are ActiveX controls. These controls are a subset of those that are available for use on UserForms. These controls work only with Excel 97 and Excel 2000, and are not compatible with Excel 5 and Excel 95.

To add to the confusion, most of the controls are available on both toolbars. For example, the Forms toolbar and the Control Toolbox toolbar both have a control named `ListBox`. However, these are two entirely different controls. In general, the ActiveX controls (those on the Control Toolbox toolbar) provide more flexibility, and you should use those controls. However, if you need to save your workbook so that it can be opened by Excel 5 or Excel 95, you should use the controls that are on the Forms toolbar.



This chapter focuses exclusively on the controls that are available in the Control Toolbox toolbar, as shown in Figure 38-2.

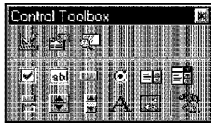


Figure 38-2: The Control Toolbox toolbar.

A description of the buttons in the Control Toolbox appears in Table 38-1.

Table 38-1
Buttons on the Control Toolbox Toolbar

Button	What It Does
Design Mode	Toggles design mode
Properties	Displays the Properties window
View Code	Switches to the Visual Basic Editor so that you can write or edit VBA code for the selected control
CheckBox	Inserts a CheckBox control
TextBox	Inserts a TextBox control
CommandButton	Inserts a CommandButton control
OptionButton	Inserts an OptionButton control
ListBox	Inserts a ListBox control
ComboBox	Inserts a ComboBox control
ToggleButton	Inserts a ToggleButton control
SpinButton	Inserts a SpinButton control
ScrollBar	Inserts a ScrollBar control
Label	Inserts a Label control
Image	Inserts an Image control
More Controls	Displays a list of other ActiveX controls that are installed on your system

Using Controls

Adding ActiveX controls in a worksheet is easy. After you add a control, you can adjust its properties to modify the way that the control looks and works.

Adding a Control

To add a control to a worksheet, make sure that the Control Toolbox toolbar is displayed—and don't confuse it with the Forms toolbar. Then, click and drag the control that you want to use into the worksheet to create the control. You don't need to be too concerned about the exact size or position, because you can modify these properties at any time.

About Design Mode

When you add a control to a worksheet, Excel goes into *design mode*. In this mode, you can adjust the properties of any controls on your worksheet, add or edit macros for the control, or change the control's size or position.

When Excel is in design mode, you can't try out the controls. To test the controls, you must exit design mode by clicking the Exit Design Mode button on the Control Toolbox toolbar.

Adjusting Properties

Every control that you add has various properties that determine how it looks and behaves. You can adjust these properties only when Excel is in design mode. When you add a control to a worksheet, Excel enters design mode automatically. If you need to change a control after you exit design mode, simply click the Design Mode button on the Control Toolbox toolbar.

To change the properties for a control, select the control and then click the Properties button on the Control Toolbox toolbar. Excel displays its Properties window, as shown in Figure 38-3. The Properties window has two tabs. The Alphabetic tab displays the properties in alphabetical order. The Categorized tab displays the properties by category. Both tabs show the same properties; only the order is different.

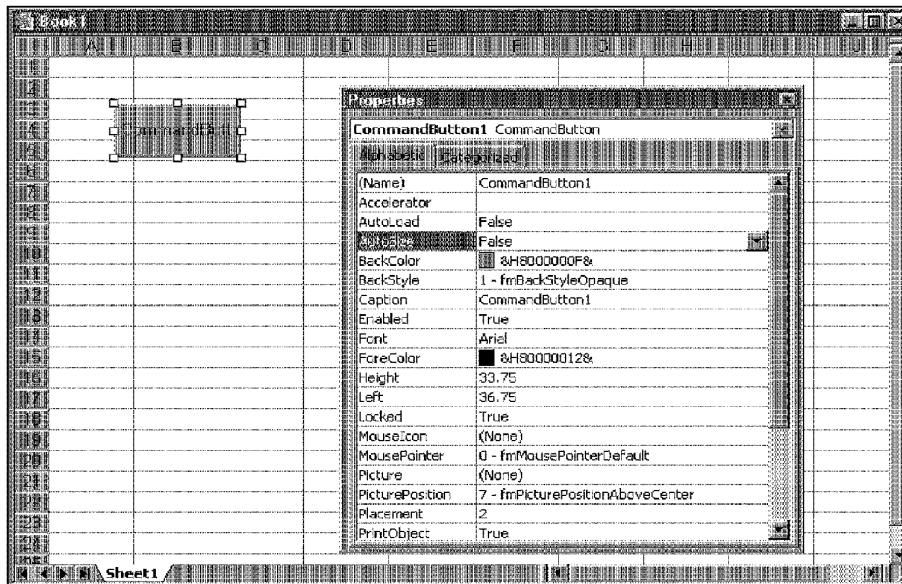


Figure 38-3: The Properties window lets you adjust the properties of a control.

To change a property, select it in the Properties window and then make the change. The manner in which you change a property depends on the property. Some properties display a drop-down list that lets you select from a list of options. Others (such as Font) provide a button that, when clicked, displays a dialog box. Other properties require you to type the property value. When you change a property, the change takes effect immediately.



Tip

To learn about a particular property, select the property in the Properties window and press F1.

Common Properties

Each control has its own unique set of properties. However, many controls share properties. This section describes some of the properties that are common to all or many controls, as set forth in Table 38-2.

Table 38-2
Properties Shared by Multiple Controls

<i>Property</i>	<i>Description</i>
Accelerator	The letter underlined in the control's caption.
AutoSize	If True , the control resizes itself automatically, based on the text in its caption.
BackColor	The background color of the control.
BackStyle	The style of the background (either transparent or opaque).
Caption	The text that appears on the control.
LinkedCell	A worksheet cell that contains the current value of a control.
ListFillRange	A worksheet range that contains items displayed in a ListBox or ComboBox control.
Value	The control's value.
Left and Top	Values that determine the control's position.
Width and Height	Values that determine the control's width and height.
Visible	If False , the control is hidden.
Name	The name of the control. By default, a control's name is based on the control type. You can change the name to any valid name. However, each control's name must be unique on the worksheet.
Picture	Enables you to specify a graphic image to display. The image must be contained in a file (it can't be copied from the Clipboard).

Linking Controls to Cells

Often, you can use ActiveX controls in a worksheet, without using any macros. Many of the controls have a `LinkedCell` property, which specifies a worksheet cell that is "linked" to the control.

For example, you might add a `SpinButton` control and specify `B1` as its `LinkedCell` property. After doing so, cell `B1` contains the value of the `SpinButton`, and clicking the `SpinButton` changes the value in cell `B1` (see Figure 38-4). You can, of course, use the value contained in the linked cell in your formulas.

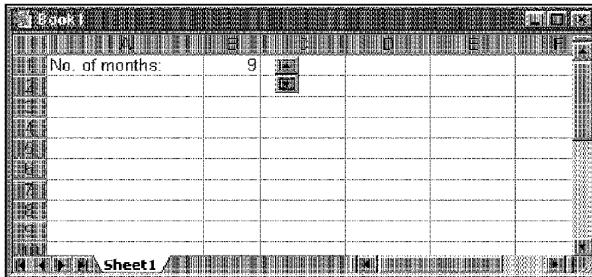


Figure 38-4: The SpinButton's `LinkedCell` property is set to cell B1, enabling the user to change the cell's value by using the SpinButton control.

Creating Macros for Controls

To create a macro for a control, you must use the Visual Basic Editor (VBE). The macros are stored in the code module for the sheet that contains the control. Each control can have a macro to handle any of its events. For example, a `CommandButton` control can have a macro for its `Click` event, its `Db1Click` event, and various other events.



Tip

The easiest way to access the code module for a control is to double-click the control while in design mode. Excel displays the VBE and creates an empty macro for the control's `Click` event. (See Figure 38-5.)

The control's name appears in the upper-left portion of the code window, and the event appears in the upper-right area. If you want to create a macro that executes when a different event occurs, select the event from the list in the upper-right area.

The following steps demonstrate how to insert a `CommandButton` and create a simple macro that displays a message when the button is clicked:

1. Make sure that the Control Toolbox toolbar is displayed.
2. Click the `CommandButton` tool in the Control Toolbox.
3. Click and drag in the worksheet to create the button.
4. Double-click the button. The VBE window is activated, and an empty subroutine is created.
5. Enter the following VBA statement before the `End Sub` statement:

```
MsgBox "You clicked on the command button."
```
6. Press `Alt+F11` to return to Excel.
7. Adjust any other properties for the `CommandButton`.
8. Click the `Exit Design Mode` button in the Control Toolbox toolbar.

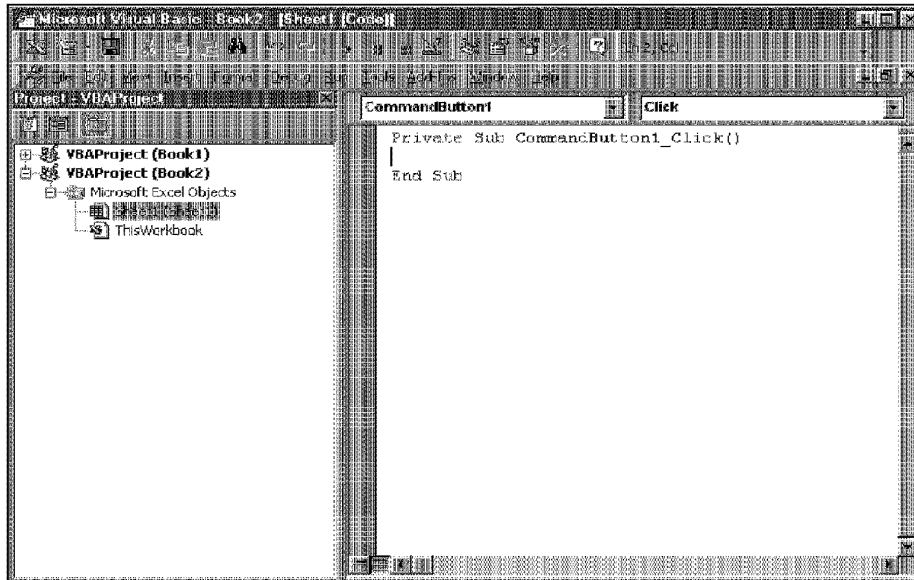


Figure 38-5: Double-clicking a control in design mode activates the Visual Basic Editor.

After performing the preceding steps, click the CommandButton to display the message box that is shown in Figure 38-6.

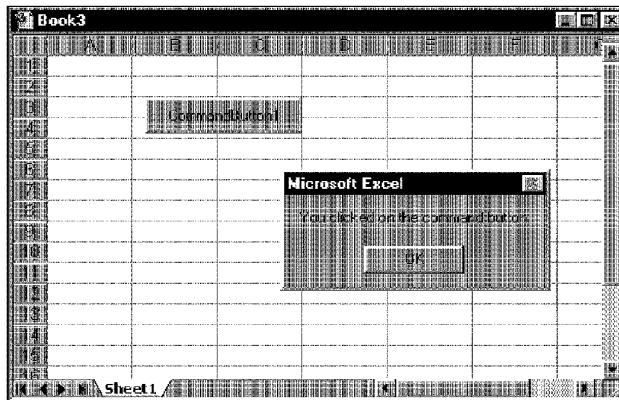


Figure 38-6: This message box is displayed by a simple macro.



When you use a CommandButton on a worksheet, setting its `TakeFocusOnClick` property to `False` is recommended. Otherwise, you may run into problems if the macro tries to select cells on the worksheet. If the CommandButton has the focus, the cells can't be selected!

The Controls Toolbox Controls

The sections that follow describe the ActiveX controls that are available on the Controls Toolbox toolbar.



The companion CD-ROM contains a file that includes examples of all the ActiveX controls.

CheckBox Control

A CheckBox control is useful for getting a binary choice: yes or no, true or false, on or off, and so on. Figure 38-7 shows some examples of CheckBox controls. Each of these controls displays its value in a cell (in A1:A4).

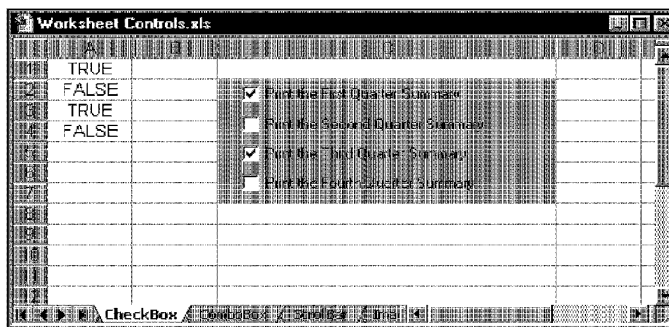


Figure 38-7: CheckBox controls on a worksheet.

The following is a description of the most useful properties of a CheckBox control:

- **Accelerator:** A letter that enables the user to change the value of the control by using the keyboard. For example, if the accelerator is A, pressing Alt+A changes the value of the CheckBox control.
- **LinkedCell:** The worksheet cell that's linked to the CheckBox. The cell displays TRUE if the control is checked or FALSE if the control is not checked.

ComboBox Control

A ComboBox control is similar to a ListBox control. A ComboBox, however, is a drop-down box, and it displays only one item at a time. Another difference is that the user may be allowed to enter a value that does not appear in the list of items.

Figure 38-8 shows a few ComboBox controls. One of these controls uses two columns for its ListFill range.

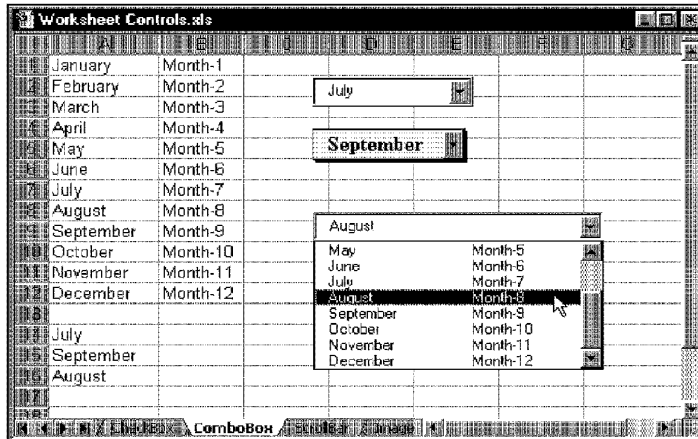


Figure 38-8: ComboBox controls.

The following is a description of the most useful properties of a ComboBox control:

- **BoundColumn:** If the list contains multiple columns, this property determines which column contains the returned value.
- **ColumnCount:** The number of columns in the list.
- **LinkedCell:** The worksheet cell that displays the selected item.
- **ListFillRange:** The worksheet range that contains the list items.
- **ListRows:** The number of items to display when the list drops down.
- **ListStyle:** Determines the appearance of the list items.
- **MultiSelect:** Determines whether the user can select multiple items from the list.
- **Style:** Determines whether the control acts like a drop-down list or a ComboBox. A drop-down list doesn't allow the user to enter a new value.



If you use a multiselect ListBox, you cannot specify a LinkedCell; you need to write a macro to determine which items are selected.

CommandButton Control

A CommandButton is useless if you don't provide a macro to execute when the button is clicked. Figure 38-9 shows a worksheet that uses several CommandButtons. One of these CommandButtons uses a picture.

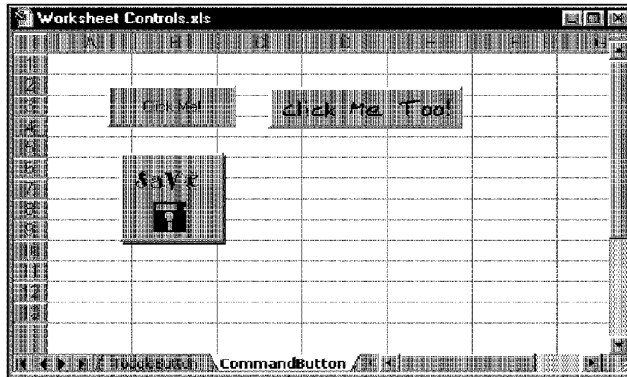


Figure 38-9: CommandButtons on a worksheet.

When a button is clicked, it executes a macro with a name that is made up of the CommandButton's name, an underscore, and the word *Click*. For example, if a CommandButton is named MyButton, clicking it executes the macro named MyButton_Click.

Image Control

An Image control is used to display an image that is contained in a file. This control offers no significant advantages over using standard imported images (as described in Chapter 14).

Label Control

A Label control simply displays text. This is not a useful control for use on worksheets, and a standard TextBox AutoShape gives you more versatility.

ListBox Controls

The ListBox control presents a list of items, and the user can select an item (or multiple items). Figure 38-10 shows a worksheet with several ListBox controls. As you can see, you have a great deal of control over the appearance of ListBox controls. One of the ListBoxes uses two columns as its ListFill range.

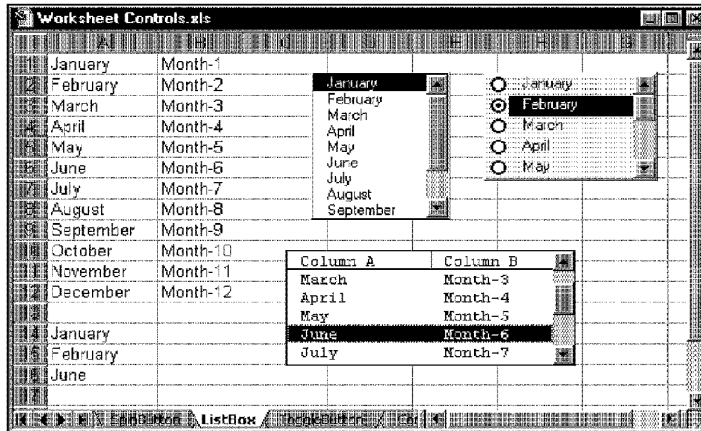


Figure 38-10: ListBox controls on a worksheet.

You can specify a range that holds the ListBox items, and this range can consist of multiple columns.

The following is a description of the most useful properties of a ListBox control:

- **BoundColumn:** If the list contains multiple columns, this property determines which column contains the returned value.
- **ColumnCount:** The number of columns in the list.
- **IntegralHeight:** This is `True` if the height of the ListBox adjusts automatically to display full lines of text when the list is scrolled vertically. If `False`, the ListBox may display partial lines of text when it is scrolled vertically.
- **LinkedCell:** The worksheet cell that displays the selected item.
- **ListFillRange:** The worksheet range that contains the list items.
- **ListStyle:** Determines the appearance of the list items.
- **MultiSelect:** Determines whether the user can select multiple items from the list.



If you use a multiselect ListBox, you cannot specify a `LinkedCell`; you need to write a macro to determine which items are selected.

OptionButton Controls

OptionButtons are useful when the user needs to select from a small number of items. OptionButtons are always used in groups of at least two. Figure 38-11 shows two sets of OptionButtons. One set uses graphic images (set with the `Picture` property).

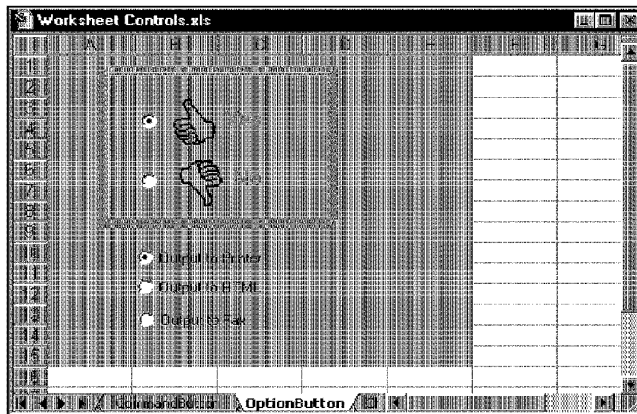


Figure 38-11: Two sets of OptionButtons.

The following is a description of the most useful properties of an OptionButton control:

- **Accelerator:** A letter that lets the user select the option by using the keyboard. For example, if the accelerator for an OptionButton is C, pressing Alt+C selects the control.
- **GroupName:** A name that identifies an OptionButton as being associated with other OptionButtons with the same GroupName property.
- **LinkedCell:** The worksheet cell that's linked to the OptionButton. The cell displays TRUE if the control is selected or FALSE if the control is not selected.



If your worksheet contains more than one set of OptionButtons, you *must* change the GroupName property for all OptionButtons in a particular set. Otherwise, all OptionButtons become part of the same set.

ScrollBar Control

The ScrollBar control is similar to a SpinButton control (discussed next). The difference is that the user can drag the ScrollBar's button to change the control's value in larger increments. Figure 38-12 shows a worksheet with three ScrollBar controls. These ScrollBars are used to change the color in the rectangle objects. The value of the ScrollBars determines the red, green, or blue component of the rectangle's color. This example uses a few simple macros to change the colors.

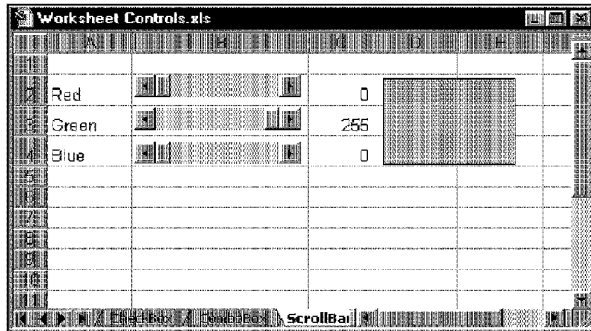


Figure 38-12: This worksheet has several ScrollBar controls.

The following is a description of the most useful properties of a ScrollBar control:

- **Value:** The current value of the control.
- **Min:** The minimum value for the control.
- **Max:** The maximum value for the control.
- **LinkedCell:** The worksheet cell that displays the value of the control.
- **SmallChange:** The amount that the control's value is changed by a click.
- **LargeChange:** The amount that the control's value is changed by clicking either side of the button.

The ScrollBar control is most useful for selecting a value that extends across a wide range of possible values.

SpinButton Control

The SpinButton control lets the user select a value by clicking the control, which has two arrows (one to increase the value and the other to decrease the value). Figure 38-13 shows a worksheet that uses several SpinButton controls. Each control is linked to the cell to the right. As you can see, a SpinButton can display either horizontally or vertically.

The following is a description of the most useful properties of a SpinButton control:

- **Value:** The current value of the control.
- **Min:** The minimum value of the control.
- **Max:** The maximum value of the control.
- **LinkedCell:** The worksheet cell that displays the value of the control.
- **SmallChange:** The amount that the control's value is changed by a click. Usually, this property is set to 1, but you can make it any value.

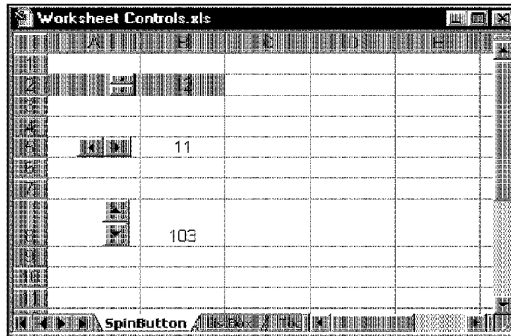


Figure 38-13: SpinButton controls in a worksheet.

If you use a linked cell for a SpinButton, you need to understand that the worksheet is recalculated every time the value of the control is changed. Therefore, if the user changes the value from 0 to 12, the worksheet gets calculated 12 times. If your worksheet takes a long time to calculate, you may want to reconsider using this control.

TextBox Controls

On the surface, a TextBox control may not seem useful. After all, it simply contains text—you can usually use worksheet cells to get text input. In fact, TextBox controls are useful not so much for input control but for output control. Because a TextBox can have ScrollBars, you can use a TextBox to display a great deal of information in a small area.

Figure 38-14 shows an example of a TextBox that is used to provide help information. The user can use the ScrollBar to read the text. The advantage is that the text uses only a small amount of screen space. The example in this figure uses three controls: the TextBox, a Label control, and a disabled CommandButton control (which provides a backdrop for the other two controls).

The following is a description of the most useful properties of a TextBox control:

- **AutoSize:** Determines whether the control adjusts its size automatically, depending on the amount of text.
- **IntegralHeight:** If `True`, the height of the TextBox adjusts automatically to display full lines of text when the list is scrolled vertically. If `False`, the ListBox may display partial lines of text when it is scrolled vertically.
- **MaxLength:** The maximum number of characters allowed in the TextBox. If 0, no limit exists on the number of characters.
- **MultiLine:** If `True`, the TextBox can display more than one line of text.
- **TextAlign:** Determines how the text is aligned in the TextBox.

- **WordWrap:** Determines whether the control allows word wrap.
- **ScrollBars:** Determines the type of ScrollBars for the control: horizontal, vertical, both, or none.

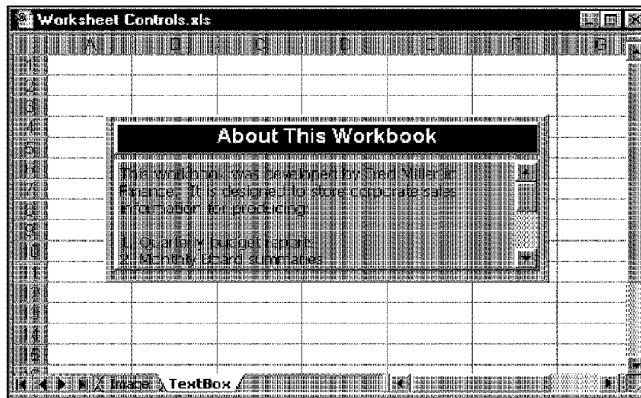


Figure 38-14: This worksheet uses a TextBox to display help information.

ToggleButton Control

A ToggleButton control has two states: on or off. Clicking the button toggles between these two states, and the button changes its appearance. Its value is either `True` (pressed) or `False` (not pressed). You can often use a ToggleButton in place of a CheckBox control.

Summary

This chapter describes how to add ActiveX controls to a worksheet and how to use these controls to enable users easily to provide data that's used in a worksheet.

• • • • •

VBA Programming Examples

My philosophy about learning to write Excel macros places heavy emphasis on examples. I've found that a well-thought-out example often communicates a concept much better than a lengthy description of the underlying theory. In this book, I chose to avoid a painstaking description of every nuance of VBA. I take this approach for two reasons. First, space limitations prohibit such a discussion. But more to the point, the VBA language is described very well in Excel's online Help system.

This chapter consists of several examples that demonstrate common VBA techniques. You may be able to use some of the examples directly, but in most cases, you must adapt them to your own needs. These examples are organized into the following categories:

- Working with ranges
- Changing Excel's settings
- Working with graphic objects
- Working with charts
- Learning ways to speed your VBA code



All subroutines and functions in this chapter can be found in a workbook that's included on the companion CD-ROM.

Working with Ranges

Most of what you do in VBA probably involves worksheet ranges. When you work with range objects, keep the following points in mind:

39 CHAPTER

In This Chapter

Working with Ranges

Changing Excel's Settings

Working with Graphic Objects (Shapes)

Working with Charts

Modifying Properties

VBA Speed Tips

- Your VBA code doesn't need to select a range to do something with the range.
- If your code does select a range, its worksheet must be active.
- The macro recorder doesn't always generate the most efficient code. Often, you can use the recorder to create your macro and then edit the code to make it more efficient.
- Using named ranges in your VBA code is recommended. For example, a reference such as Range("Total") is better than Range("D45"). In the latter case, you need to modify the macro if you add a row above row 45.
- When you record macros that select ranges, pay close attention to "relative vs. absolute" recording mode. The recording mode that you choose can drastically affect the way the macro operates.
- If you create a macro that loops through each cell in the current range selection, be aware that the user can select entire columns or rows. In most cases, you don't want to loop through every cell in the selection. You need to create a subset of the selection that consists only of nonblank cells.
- Be aware that Excel allows multiple selections. For example, you can select a range, press Ctrl, and then select another range. You can test for this in your macro and take appropriate actions.

The examples in the following sections demonstrate these points.

Copying a Range

Copying a range is a frequent activity in macros. When you turn on the macro recorder (using absolute recording mode) and copy a range from A1:A5 to B1:B5, you get a VBA macro like this:

```
Sub CopyRange()  
    Range("A1:A5").Select  
    Selection.Copy  
    Range("B1").Select  
    ActiveSheet.Paste  
    Application.CutCopyMode = False  
End Sub
```

This macro works, but it's not the most efficient way to copy a range. You can accomplish exactly the same result with the following one-line macro:

```
Sub CopyRange2()  
    Range("A1:A5").Copy Range("B1")  
End Sub
```

This takes advantage of the fact that the Copy method can use an argument that specifies the destination. Information such as this is available in the online Help system.

The example demonstrates that the macro recorder doesn't always generate the most efficient code. As you see, you don't have to select an object to work with it. Note that Macro2 doesn't select a range; therefore, the active cell doesn't change when this macro is executed.

Copying a Variable-Size Range

Often, you want to copy a range of cells in which the exact row and column dimensions are unknown.

Figure 39-1 shows a range on a worksheet. This range consists of a number of rows, and the number of rows can change daily. Because the exact range address is unknown at any given time, writing a macro to copy the range can be challenging.

Week Ending	Calls	Orders
01/02/99	462	94
01/09/99	546	132
01/16/99	687	144
01/23/99	733	193
01/30/99	632	113
02/06/99	708	98
02/13/99	813	203
02/20/99	882	183
02/27/99	634	187
03/06/99	654	103
03/13/99	770	90

Figure 39-1: This range can consist of any number of rows.

The macro that follows demonstrates how to copy this range from Sheet1 to Sheet2 (beginning at cell A1). It uses the `CurrentRegion` property, which returns a `Range` object that corresponds to the active block of cells. This is equivalent to choosing `Edit • Go To`, clicking the `Special` button, and then selecting the `Current Region` option.

```
Sub CopyCurrentRegion()
    Range("A1").CurrentRegion.Copy
    Sheets("Sheet2").Select
    Range("A1").Select
    ActiveSheet.Paste
    Sheets("Sheet1").Select
    Application.CutCopyMode = False
End Sub
```

Selecting to the End of a Row or Column

You probably are in the habit of using key combinations, such as Ctrl+Shift+right-arrow key and Ctrl+Shift+down-arrow key, to select from the active cell to the end of a row or column. When you record these actions in Excel (using relative recording mode), you'll find that the resulting code works as you would expect it to.



In previous versions of Excel, the macro recorder always recorded absolute cell addresses when making these types of selections. This problem has been fixed in Excel 2000.

The following VBA subroutine selects the range that begins at the active cell and extends down to the last cell in the column (or to the first empty cell, whichever comes first). When the range is selected, you can do whatever you want with it—copy it, move it, format it, and so on.

```
Sub SelectDown()  
    Range(ActiveCell, ActiveCell.End(xlDown)).Select  
End Sub
```

This example uses the `End` method of the `Range` object, which returns a `Range` object. The `End` method takes one argument, which can be any of the following constants: `xlUp`, `xlDown`, `xlToLeft`, or `xlToRight`.

Selecting a Row or Column

The macro that follows demonstrates how to select the column of the active cell. It uses the `EntireColumn` property, which returns a range that consists of a column.

```
Sub SelectColumn()  
    ActiveCell.EntireColumn.Select  
End Sub
```

As you may suspect, an `EntireRow` property also is available, which returns a range that consists of a row.

If you want to perform an operation on all cells in the selected column, you don't need to select the column. For example, the following subroutine makes all cells bold in the row that contains the active cell:

```
Sub MakeRowBold()  
    ActiveCell.EntireRow.Font.Bold = True  
End Sub
```

Moving a Range

Moving a range consists of cutting it to the Clipboard and then pasting it to another area. If you record your actions while performing a move operation, the macro recorder generates code as follows:

```
Sub MoveRange()  
    Range("A1:C6").Select  
    Selection.Cut  
    Range("A10").Select  
    ActiveSheet.Paste  
End Sub
```

As demonstrated with copying earlier in this chapter, this is not the most efficient way to move a range of cells. In fact, you can do it with a single VBA statement, as follows:

```
Sub MoveRange2()  
    Range("A1:C6").Cut Range("A10")  
End Sub
```

This statement takes advantage of the fact that the `Cut` method can use an argument that specifies the destination.

Looping Through a Range Efficiently

Many macros perform an operation on each cell in a range, or they may perform selective actions based on the content of each cell. These operations usually involve a `For-Next` loop that processes each cell in the range.

The following example demonstrates how to loop through all the cells in a range. In this case, the range is the current selection. In this example, `Cell` is a variable name that refers to the cell being processed. Within the `For-Next` loop, the single statement evaluates the cell and changes its font color if the cell value is negative (`vbRed` is a built-in constant that represents the color red).

```
Sub ProcessCells()  
    For Each Cell In Selection  
        If Cell.Value < 0 Then Cell.Font.Color = vbRed  
    Next Cell  
End Sub
```

The preceding example works, but what if the selection consists of an entire column or an entire range? This is not uncommon, because Excel lets you perform operations on entire columns or rows. But in this case, the macro seems to take forever, because it loops through each cell—even those that are blank. What's needed is a way to process only the nonblank cells.

This can be accomplished by using the `SelectSpecial` method. In the following example, the `SelectSpecial` method is used to create two new objects: the subset of the selection that consists of cells with constants, and the subset of the selection that consists of cells with formulas. Each of these subsets is processed, with the net effect of skipping all blank cells.

```
Sub SkipBlanks()  
  * Ignore errors  
  On Error Resume Next  
  
  * Process the constants  
  Set ConstantCells = Selection.SpecialCells(xlConstants, 23)  
  For Each cell In ConstantCells  
    If cell.Value > 0 Then cell.Font.Color = vbRed  
  Next cell  
  
  * Process the formulas  
  Set FormulaCells = Selection.SpecialCells(xlFormulas, 23)  
  For Each cell In FormulaCells  
    If cell.Value > 0 Then cell.Font.Color = vbRed  
  Next cell  
End Sub
```

The `SkipBlanks` subroutine works fast, regardless of what is selected. For example, you can select the range, select all columns in the range, select all rows in the range, or even select the entire worksheet. In all of these cases, only the cells that contain constants or values are processed. This is a vast improvement over the `ProcessCells` subroutine presented earlier.

Notice that the following statement is used in the subroutine:

```
On Error Resume Next
```

This statement causes Excel to ignore any errors that occur and simply to process the next statement. This is necessary because the `SpecialCells` method produces an error if no cells qualify. Normal error checking is resumed when the subroutine ends. To tell Excel explicitly to return to normal error-checking mode, use the following statement:

```
On Error GoTo 0
```

Prompting for a Cell Value

As discussed in Chapter 37, you can take advantage of VBA's `InputBox` function to solicit a value from the user. Figure 39-2 shows an example.

You can assign this value to a variable and use it in your subroutine. Often, however, you want to place the value into a cell. The following subroutine demonstrates how to ask the user for a value and place it into cell A1 of the active worksheet, using only one statement:

```
Sub GetValue()
    Range("A1").Value = InputBox("Enter the value for cell A1")
End Sub
```

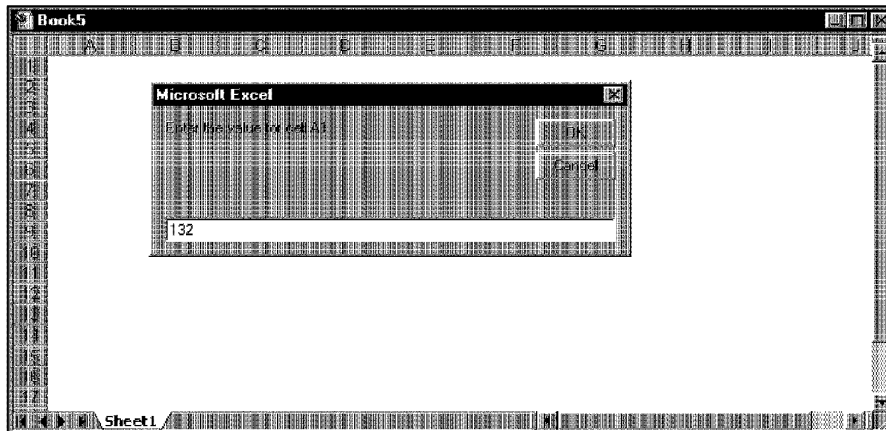


Figure 39-2: Using VBA's InputBox function to get a value from the user.

Determining the Type of Selection

If your macro is designed to work with a range selection, you need to determine that a range is actually selected. Otherwise, the macro most likely fails. The following subroutine identifies the type of object that is currently selected:

```
Sub SelectionType()
    MsgBox TypeName(Selection)
End Sub
```

If a Range object is selected, the MsgBox displays *Range*. If your macro is designed to work only with ranges, you can use an If statement to ensure that a range is actually selected. The following is an example that beeps, displays a message, and exits the subroutine if the current selection is not a Range object:

```
Sub CheckSelection()
    If TypeName(Selection) <> "Range" Then
        Beep
        MsgBox "Select a range."
        Exit Sub
    End If
    ' ... [Other statements go here]
End Sub
```

Another way to approach this is to define a custom function that returns True if the selection is a Range object, and False otherwise. The following function does just that:

```
Function IsRange(sel) As Boolean
    IsRange = False
    If TypeName(sel) = "Range" Then IsRange = True
End Function
```

If you enter the IsRange function in your module, you can rewrite the CheckSelection subroutine as follows:

```
Sub CheckSelection()
    If IsRange(Selection) Then
        ' ... [Other statements go here]
    Else
        Beep
        MsgBox "Select a range."
        Exit Sub
    End If
End Sub
```

Identifying a Multiple Selection

As you know, Excel enables you to make a multiple selection by pressing Ctrl while you select objects or ranges. This can cause problems with some macros; for example, you can't copy a multiple selection that consists of nonadjacent ranges. The following macro demonstrates how to determine whether the user has made a multiple selection:

```
Sub MultipleSelection()
    If Selection.Areas.Count > 1 Then
        MsgBox "Multiple selections not allowed."
        Exit Sub
    End If
    ' ... [Other statements go here]
End Sub
```

This example uses the Areas method, which returns a collection of all objects in the selection. The Count property returns the number of objects that are in the collection.

The following is a VBA function that returns True if the selection is a multiple selection:

```
Function IsMultiple(sel) As Boolean
    IsMultiple = False
    If Selection.Areas.Count > 1 Then IsMultiple = True
End Function
```

Changing Excel's Settings

Some of the most useful macros are simple subroutines that change one or more of Excel's settings. For example, it takes quite a few actions simply to change the Recalculation mode from automatic to manual.

This section contains two examples that demonstrate how to change settings in Excel. These examples can be generalized to other operations.

Boolean Settings

A Boolean setting is one that is either on or off. For example, you may want to create a macro that turns on and off the row and column headings. If you record your actions while you access the Options dialog box, you find that Excel generates the following code if you turn off the headings:

```
ActiveWindow.DisplayHeadings = False
```

It generates the following code if you turn on the headings:

```
ActiveWindow.DisplayHeadings = True
```

This may lead you to suspect that the heading display requires two macros: one to turn on the headings and one to turn them off. Actually, this isn't true. The following subroutine uses the `Not` operator effectively to toggle the heading display from True to False and from False to True:

```
Sub ToggleHeadings()  
    If TypeName(ActiveSheet) <> "Worksheet" Then Exit Sub  
    ActiveWindow.DisplayHeadings = Not  
    ActiveWindow.DisplayHeadings  
End Sub
```

The first statement ensures that the active sheet is a worksheet; otherwise, an error occurs (chart sheets don't have row and column headers). This technique can be used with any other settings that take on Boolean (True or False) values. For example, you can create macros to toggle sheet tab display, gridlines, and so on. The best way to find out which properties control these items is to turn on the macro recorder while you change them. Then, examine the VBA code.

Non-Boolean Settings

For non-Boolean settings, you can use the following `Select Case` structure. This example toggles the Calculation mode and displays a message indicating the current mode:

```

Sub ToggleCalcMode()
  Select Case Application.Calculation
    Case xlManual
      Application.Calculation = xlAutomatic
      MsgBox "Automatic Calculation Mode"
    Case xlAutomatic
      Application.Calculation = xlManual
      MsgBox "Manual Calculation Mode"
  End Select
End Sub

```

Working with Graphic Objects (Shapes)

VBA subroutines can work with any type of Excel object, including graphic objects that are embedded on a worksheet's draw layer. This section provides a few examples of using VBA to manipulate graphic objects.

Creating a Text Box to Match a Range

The following example creates a text box that is positioned precisely over the selected range of cells. This is useful if you want to make a text box that covers up a range of data.

```

Sub CreateTextBox()
  If TypeName(Selection) <> "Range" Then Exit Sub
  Set RangeSelection = Selection
  * Get coordinates of range selection
  SelLeft = Selection.Left
  SelTop = Selection.Top
  SelWidth = Selection.Width
  SelHeight = Selection.Height
  * Create a text box

  ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal, _
    SelLeft, SelTop, SelWidth, SelHeight).Select
  RangeSelection.Select
End Sub

```

The macro first checks to make sure that a range is selected. If not, the subroutine is exited with no further action. If a range is selected, the coordinates (Left, Top, Width, and Height) are assigned to four variables. These variables are then used as the arguments for the AddTextbox method of the Shapes collection.

The following is a more sophisticated version of this macro that works with a multiple selection of cells. The subroutine creates a text box for each area in the multiple selection. It uses a For-Next loop to cycle through each area in the range selection. If the range has only one area (not a multiple selection), the For-Next loop is activated only one time.


```

Sub CreateTextBox2()
    If TypeName(Selection) <> "Range" Then Exit Sub
    Set RangeSelection = Selection
    For Each Part In Selection.Areas
        ' Get coordinates of range selection
        SelLeft = Part.Left
        SelTop = Part.Top
        SelWidth = Part.Width
        SelHeight = Part.Height
        ' Create a text box

        ActiveSheet.Shapes.AddTextbox(msoTextOrientationHorizontal,
        _
        SelLeft, SelTop, SelWidth, SelHeight).Select
    Next Part
    RangeSelection.Select
End Sub

```

Drawing Attention to a Range

The example in this section is a macro that draws an AutoShape around the selected range. Figure 39-3 shows an example.

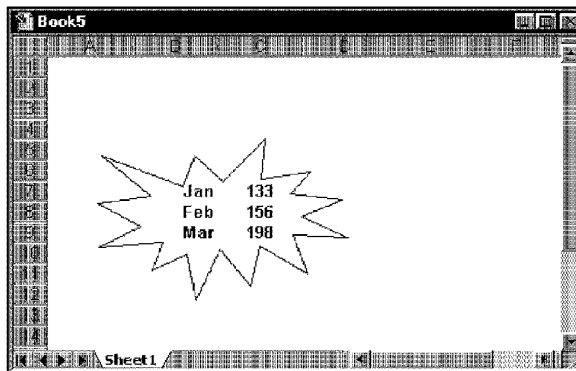


Figure 39-3: A macro draws the AutoShape around a selected range of cells.

```

Sub AddExplosion()
    If TypeName(Selection) <> "Range" Then Exit Sub
    SelLeft = Selection.Left - (Selection.Width * 0.2)
    SelTop = Selection.Top - (Selection.Height * 0.5)
    SelWidth = Selection.Width + (Selection.Width * 0.4)
    SelHeight = Selection.Height + Selection.Height
    ActiveSheet.Shapes.AddShape (msoShapeExplosion1, _
    SelLeft, SelTop, SelWidth, SelHeight).Select
    Selection.ShapeRange.Fill.Visible = msoFalse
End Sub

```

The macro begins by determining the location and size of the shape, using the selected range. The shape needs to be larger than the selected range and must be offset to the left and to the top. Therefore, the macro performs some calculations to determine the left, top, width, and height of the shape. In this example, the shape's height is twice as large as the height of the selection and 40 percent wider than the width of the selection. These calculations were determined by trial and error. In most cases, the shape is drawn in such a way that the contents of the underlying cells are completely visible. In other cases, slight adjustments are required.

After the parameters are calculated, the AutoShape is added to the active sheet. The AutoShape that's drawn by the macro is identified by a constant (`msoShapeExplosion1`). The final statement makes the shape transparent.

Working with Charts

Manipulating charts with VBA can be confusing, mainly because of the large number of objects involved. To get a feel for this, turn on the macro recorder, create a chart, and perform some routine chart editing. You may be surprised by the amount of code that's generated.

After you understand the objects in a chart, however, you can create some useful macros. This section presents a few macros that deal with charts. When you write macros that manipulate charts, you need to understand some terminology. An embedded chart on a worksheet is a `ChartObject` object. Before you can do anything to a `ChartObject`, you must activate it. The following statement activates the `ChartObject` named `Chart 1`.

```
ActiveSheet.ChartObjects("Chart 1").Activate
```

After you activate the `ChartObject`, you can refer to it in your VBA code as the `ActiveChart`. If the chart is on a separate chart sheet, it becomes the active chart as soon as the chart sheet is activated.

Modifying the Chart Type

The following example changes the chart type of every embedded chart on the active sheet. It makes each chart an area chart by adjusting the `Type` property of the `ActiveChart` object. A built-in constant, `xlArea`, represents an area chart.

```
Sub ChartType()  
    For Each cht In ActiveSheet.ChartObjects  
        cht.Activate  
        ActiveChart.Type = xlArea  
    Next cht  
End Sub
```

The preceding example uses a For-Next loop to cycle through all the ChartObject objects on the active sheet. Within the loop, the chart is activated and then the chart type is assigned a new value.

The following macro performs the same function but works on all chart sheets in the active workbook:

```
Sub ChartType2()  
  For Each cht In ThisWorkbook.Charts  
    cht.Activate  
    ActiveChart.Type = xlArea  
  Next cht  
End Sub
```

Modifying Properties

The following example changes the legend font for all charts that are on the active sheet. It uses a For-Next loop to process all ChartObject objects, and uses the On Error statement to ignore the error that occurs if a chart does not have a legend.

```
Sub LegendMod()  
  On Error Resume Next  
  For Each cht In ActiveSheet.ChartObjects  
    cht.Activate  
    With ActiveChart.Legend.Font  
      .Name = "Arial"  
      .FontStyle = "Bold"  
      .Size = 8  
    End With  
  Next cht  
End Sub
```

Applying Chart Formatting

This example applies several different formatting types to the active chart. A chart must be activated before executing this macro. You activate an embedded chart by selecting it. Activate a chart on a chart sheet by activating the chart sheet.

```
Sub ChartMods()  
  On Error Resume Next  
  With ActiveChart  
    .Type = xlArea  
    .ChartArea.Font.Name = "Arial"  
    .ChartArea.Font.FontStyle = "Regular"  
    .ChartArea.Font.Size = 9  
    .PlotArea.Interior.ColorIndex = xlNone  
    .Axes(xlValue).TickLabels.Font.Bold = True  
    .Axes(xlCategory).TickLabels.Font.Bold = True  
    .Legend.Position = xlBottom  
  End With  
End Sub
```

I created this macro by recording my actions as I formatted a chart. Then, I cleaned up the recorded code by removing irrelevant lines.

VBA Speed Tips

VBA is fast, but it's often not fast enough. This section presents some programming examples that you can use to help speed your macros.

Turning Off Screen Updating

You've probably noticed that when you execute a macro, you can watch everything that occurs in the macro. Sometimes this is instructive, but after you get the macro working properly, it can be annoying and slow things considerably.

Fortunately, a way exists to disable the normal screen updating that occurs when you execute a macro. Insert the following statement to turn off screen updating:

```
Application.ScreenUpdating = False
```

If, at any point during the macro, you want the user to see the results of the macro, use the following statement to turn back on screen updating:

```
Application.ScreenUpdating = True
```

Preventing Alert Messages

One of the benefits of using a macro is that you can perform a series of actions automatically. You can start a macro and then get a cup of coffee while Excel does its thing. Some operations cause Excel to display messages that must be attended to, however. For example, if your macro deletes a sheet, you see the message that is shown in the dialog box in Figure 39-4. These types of messages mean that you can't execute your macro unattended.

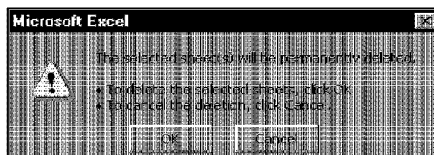


Figure 39-4: You can instruct Excel not to display these types of alerts while a macro is running.

To avoid these alert messages, insert the following VBA statement:

```
Application.DisplayAlerts = False
```

When the subroutine ends, the `DisplayAlerts` property is automatically reset to `True` (its normal state).

Simplifying Object References

As you probably have discovered, references to objects can get very lengthy — especially if your code refers to an object that's not on the active sheet or in the active workbook. For example, a fully qualified reference to a `Range` object may look like this:

```
Workbooks("MyBook").Worksheets("Sheet1").Range("IntRate")
```

If your macro uses this range frequently, you may want to create an object variable by using the `Set` command. For example, to assign this `Range` object to an object variable named `Rate`, use the following statement:

```
Set Rate = Workbooks("MyBook").Worksheets("Sheet1"). _  
    Range("IntRate")
```

After this variable is defined, you can use the variable `Rate` instead of the lengthy reference.

Besides simplifying your coding, using object variables also speeds your macros quite a bit. I've seen some macros execute twice as fast after creating object variables.

Declaring Variable Types

Usually, you don't have to worry about the type of data that's assigned to a variable. Excel handles all these details behind the scenes. For example, if you have a variable named `MyVar`, you can assign a number or any type to it. You can even assign a text string to it later in the procedure.

But if you want your procedures to execute as fast as possible, you should tell Excel in advance what type of data is going to be assigned to each of your variables. This is known as *declaring* a variable's type.

Table 39-1 lists all the data types that are supported by VBA. This table also lists the number of bytes that each type uses and the approximate range of possible values.

**Table 39-1
Data Types**

<i>Data Type</i>	<i>Bytes Used</i>	<i>Approximate Range of Values</i>
Byte	1	0 to 255
Boolean	2	True or False
Integer	2	-32,768 to 32,767
Long (long integer)	4	-2,147,483,648 to 2,147,483,647
Single (single-precision floating-point)	4	-3.4E38 to -1.4E-45 for negative values; 1.4E-45 to 4E38 for positive values
Double (double-precision floating-point)	8	-1.7E308 to -4.9E-324 for negative values; 4.9E-324 to .7E308 for positive values
Currency (scaled integer)	8	-9.2E14 to 9.2E14
Decimal	14	+/-7.9E28 with no decimal point
Date	8	January 1, 100, to December 31, 9999
Object	4	Any Object reference
String (variable-length)	10 + string length	0 to approximately 2 billion
String (fixed-length)	Length of string	1 to approximately 65,400
Variant (with numbers)	16	Any numeric value up to the range of a Double
Variant (with characters)	22 + string length	Same range as for variable-length String
User-defined (using Type)	Number required by elements	The range of each element is the same as the range of its data type

If you don't declare a variable, Excel uses the `Variant` data type. In general, the best technique is to use the data type that uses the smallest number of bytes yet can still handle all the data assigned to it. When VBA works with data, execution speed is a function of the number of bytes that VBA has at its disposal. In other words, the fewer bytes that are used by data, the faster VBA can access and manipulate the data.

To declare a variable, use the `Dim` statement before you use the variable for the first time. For example, to declare the variable `Units` as an integer, use the following statement:

```
Dim Units as Integer
```

To declare the variable `UserName` as a string, use the following statement:

```
Dim UserName as String
```

If you know that `UserName` can never exceed 20 characters, you can declare it as a fixed-length string, as follows:

```
Dim UserName as String * 20
```

If you declare a variable within a procedure, the declaration is valid only within that procedure. If you declare a variable outside of any procedures (but before the first procedure), the variable is valid in all procedures in the module.

If you use an object variable (as described previously), you can declare the variable as an object data type. The following is an example:

```
Dim Rate as Range  
Set Rate = Workbooks("MyBook").Worksheets("Sheet1").  
Range("Intrate")
```

To force yourself to declare all the variables that you use, insert the following statement at the top of your module:

```
Option Explicit
```

If you use this statement, Excel displays an error message if it encounters a variable that hasn't been declared.

Summary

This chapter presents several examples of VBA code that work with ranges, Excel's settings, graphic objects, and charts. It also discusses techniques that you can use to make your VBA macros run faster.

•

Creating Custom Excel Add-Ins

For developers, one of the most useful features in Excel is the capability to create add-ins. This chapter discusses this concept and provides a practical example of creating an add-in.

What Is an Add-In?

Generally speaking, a spreadsheet *add-in* is something that's added to the spreadsheet to give it additional functionality. Excel 2000 has several add-ins, including the Analysis ToolPak, AutoSave, and Solver. Some add-ins (such as the Analysis ToolPak, discussed in Chapter 28) provide new worksheet functions that can be used in formulas. Usually, the new features blend in well with the original interface, so they appear to be part of the program.

Excel's approach to add-ins is quite powerful, because any knowledgeable Excel user can create add-ins from XLS workbooks. An Excel add-in is basically a different form of an XLS workbook file. Any XLS file can be converted into an add-in, but not every workbook is a good candidate for an add-in. Add-ins are always hidden, so you can't display worksheets or chart sheets that are contained in an add-in. But, you can access its VBA subroutines and functions and display dialog boxes that are contained on dialog sheets.

The following are some typical uses for Excel add-ins:

- **To store one or more custom worksheet functions.** When the add-in is loaded, the functions can be used like any built-in worksheet function.
- **To store Excel utilities.** VBA is ideal for creating general-purpose utilities that extend the power of Excel. The Power Utility Pak that I created is an example of such a function.

40

CHAPTER

In This Chapter

What Is an Add-In?

Working with Add-Ins

Why Create Add-Ins?

Creating Add-Ins

An Add-In Example

Setting Up the Workbook

Testing the Workbook

Adding Descriptive Information

Protecting the Project

Creating the Add-In

Opening the Add-In

- **To store proprietary macros.** If you don't want end users to see (or modify) your macros, store the macros in an add-in. The macros can be used, but they can't be viewed or changed.

As previously noted, Excel ships with several useful add-ins (see the sidebar “Add-Ins That Are Included with Excel”), and you can acquire other add-ins from third-party vendors or the Internet. In addition, Excel includes the tools that enable you to create your own add-ins. This process is explained later in the chapter, but first, some background is required.

Working with Add-Ins

The best way to work with add-ins is to use Excel's add-in manager, which you access by selecting **Tools • Add-Ins**. This command displays the Add-Ins dialog box, shown in Figure 40-1. The list box contains all the add-ins that Excel knows about. Those that are checked are currently open. You can open and close add-ins from this dialog box by selecting or deselecting the check boxes.

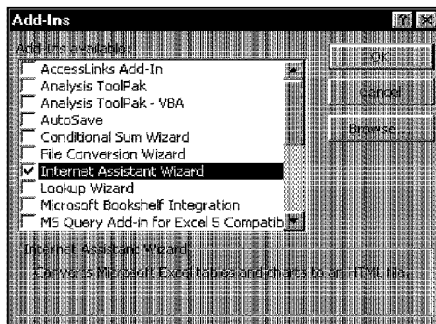


Figure 40-1: The Add-Ins dialog box.



Note

Most add-in files can also be opened by selecting **File • Open**. You'll find that after an add-in is opened, however, you can't choose **File • Close** to close it. The only way to remove the add-in is to exit and restart Excel or to write a macro to close the add-in.

When an add-in is opened, you may or may not notice anything different. In nearly every case, however, some change is made to the menu — either a new menu or one or more new menu items on an existing menu. For example, when you open the Analysis ToolPak add-in, a new menu item appears on the Tools menu: **Data Analysis**. When you open my Power Utility Pak add-in, you get a new **Utilities** menu, which is located between the **Data** and **Window** menus.

Add-Ins That Are Included with Excel

The following is a list of the add-ins that are included with Excel. Some of these add-ins may not have been installed. If you try to use one of these add-ins and it's not installed, you receive a prompt asking whether you want to install it.

- **Analysis ToolPak:** Statistical and engineering tools, plus new worksheet functions.
- **Analysis ToolPak – VBA:** VBA functions for the Analysis ToolPak.
- **AutoSave:** Automatically saves your workbook at a time interval that you specify.
- **Conditional Sum Wizard:** Helps you to create formulas that add values based on a condition.
- **File Conversion Wizard:** Converts a group of files to Excel format.
- **Lookup Wizard:** Helps you to create formulas that look up data in a list.
- **Microsoft AccessLinks Add-In:** Lets you use Microsoft Access forms and reports with Excel worksheets (Access 97 must be installed on your system).
- **Microsoft Bookshelf Integration:** Lets you access Microsoft Bookshelf from Excel.
- **MS Query Add-In for Excel 5 Compatibility:** Works with Microsoft Query to bring external data into a worksheet.
- **ODBC Add-In:** Lets you use ODBC functions to connect to external data sources directly.
- **Report Manager:** Prints reports that consist of a set sequence of views and scenarios.
- **Solver Add-In:** A tool that helps you to use a variety of numeric methods for equation solving and optimization.
- **Template Utilities:** Utilities that are used by the Spreadsheet Solutions templates. This is loaded automatically when you use one of these templates.
- **Template Wizard with Data Tracking:** Helps you to create custom templates.
- **Update Add-in Links:** Updates links to MS Excel 4.0 add-ins to directly access the new built-in functionality.
- **Web Form Wizard:** Sets up a form on a Web server to send data to a database.

The Internet Assistant Wizard, included with Excel 97, is no longer necessary, because Excel 2000 can use HTML as a native file format.

Why Create Add-Ins?

Most Excel users have no need to create add-ins. But if you develop spreadsheets for others — or if you simply want to get the most out of Excel — you may be interested in pursuing this topic further.

The following are several reasons why you may want to convert your XLS application to an add-in:

- **To prevent access to your VBA code.** When you distribute an application as an add-in, the end users can't view the sheets in the workbook. If you use proprietary techniques in your VBA code, this can prevent it from being copied (or at least make it more difficult to copy).
- **To avoid confusion.** If an end user loads your application as an add-in, the file is not visible and, therefore, is less likely to confuse novice users or get in the way. Unlike a hidden XLS workbook, an add-in can't be unhidden.
- **To simplify access to worksheet functions.** Custom worksheet functions that are stored in an add-in don't require the workbook name qualifier. For example, if you have a custom function named MOVAVG stored in a workbook named Newfunc.xls, you would have to use a syntax such as the following to use this function in a different workbook:

```
=NEWFUNC.XLS!MOVAVG(A1:A50)
```

But if this function is stored in an add-in file that's open, the syntax is much simpler, because you don't need to include the file reference:

```
=MOVAVG(A1:A50)
```

- **To provide easier access.** After you identify the location of your add-in, it appears in the Add-Ins dialog box with a friendly name and a description of what it does.
- **To permit better control over loading.** Add-ins can be opened automatically when Excel starts, regardless of the directory in which they are stored.
- **To omit prompts when unloading.** When an add-in is closed, the user never sees the *Save change in...?* prompt.

Creating Add-Ins

Although any workbook can be converted to an add-in, not all workbooks benefit by this. In fact, workbooks that consist only of worksheets (that is, not macros or custom dialog boxes) become unusable, because add-ins are hidden.



To convert a workbook to an add-in, the workbook must have at least one worksheet. Therefore, if your workbook consists only of Excel 5/95 dialog sheets or Excel 4 macro sheets, you can't convert it to an add-in.

The only types of workbooks that benefit from conversion to an add-in are those with macros. For example, you may have a workbook that consists of general-purpose macros (subroutines and functions). This type of workbook makes an ideal add-in.

Creating an add-in is quite simple. These steps describe how to create an add-in from a normal workbook file:

1. Develop your application and make sure that everything works properly. Don't forget to include a method to execute the macro or macros. You may want to add a new menu item (described later in the chapter).
2. Test the application by executing it when a *different* workbook is active. This simulates its behavior when it's an add-in, because an add-in is never the active workbook. You may find that some references no longer work. For example, the following statement works fine when the code resides in the active workbook, but fails when a different workbook is active:

```
x = Worksheets("Data").Range("A1")
```

You could qualify the reference with the name of the workbook object, like this:

```
x = Workbooks("MYBOOK.XLS").Worksheets("Data").Range("A1")
```

This method is not recommended, because the name of the workbook changes when it's converted to an add-in. The solution is to use the `ThisWorkbook` qualifier, as follows

```
x = ThisWorkbook.Worksheets("Data").Range("A1")
```

3. Select **File • Summary Info**, enter a brief descriptive title in the Title field, and then enter a longer description in the Comments field. This step is not required, but it makes using the add-in easier.
4. Lock the project. This is an optional step that protects the VBA code and UserForms from being viewed. You do this in the Visual Basic Editor, using the **Tools • Properties** command. Click the Protection tab and make the appropriate choices.
5. Save the workbook as an XLA file by selecting **File • Save As**. Select Microsoft Excel Add-In from the Save as type drop-down list.

After you create the add-in, you need to test it. Select **Tools • Add-Ins** and use the Browse button in the Add-Ins dialog box to locate the XLA file that you created in Step 5. This installs the add-in. The Add-Ins dialog box uses the descriptive title that you provided in Step 3.



You can continue to modify the macros and UserForms in the XLA version of your file, and save your changes in the Visual Basic Editor. In versions prior to Excel 97, the changes have to be made to the XLS version and then the workbook has to be resaved as an add-in.

An Add-In Example

This section discusses the steps that are used to create a useful add-in that displays a dialog box (see Figure 40-2) in which the user can quickly change several Excel settings. Although these settings can be changed in the Options dialog box, the add-in makes these changes interactively. For example, if the Grid Lines check box is deselected, the gridlines are removed immediately.

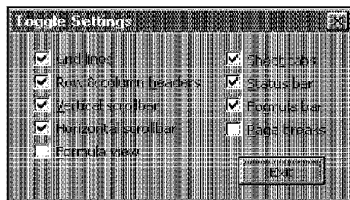


Figure 40-2: This dialog box enables the user to change various Excel settings interactively.



This file is available on the companion CD-ROM. The file is not locked, so you have full access to the VBA code and UserForm.

Setting Up the Workbook

This workbook consists of one worksheet, which is empty. Although the worksheet is not used, it must be present, because every workbook must have at least one sheet.

Use the Visual Basic Editor to insert a VBA module (named Module1) and a UserForm (named UserForm1).

Module1

The following macro is contained in the Module1 module. This subroutine ensures that a worksheet is active. If the active sheet is not a worksheet, a message box is displayed and nothing else happens. If a worksheet is active, the subroutine displays the dialog box that is contained in UserForm1.

```
Sub ShowToggleSettingsDialog()
    If TypeName(ActiveSheet) <> "Worksheet" Then
        MsgBox "A worksheet must be active.", vbInformation
    Else
        UserForm1.Show
    End If
End Sub
```

ThisWorkbook

The `ThisWorkbook` object contains a macro that adds a menu item to the Tools menu when the workbook (add-in) is opened. Another macro removes the menu item when the workbook (add-in) is closed. These two subroutines, which appear in the following syntax, are explained next:

```
Private Sub Workbook_Open()  
    Set NewMenuItem = Application.CommandBars _  
        ("Worksheet Menu Bar").Controls("Tools").Controls.Add  
    With NewMenuItem  
        .Caption = "Toggle Settings..."  
        .BeginGroup = True  
        .OnAction = "ShowToggleSettingsDialog"  
    End With  
End Sub  
  
Private Sub Workbook_BeforeClose(Cancel As Boolean)  
    On Error Resume Next  
    Application.CommandBars("Worksheet Menu Bar")._  
        Controls("Tools").Controls("Toggle Settings...").Delete  
End Sub
```

The `Workbook_Open` subroutine adds a menu item (Toggle Settings) to the bottom of the Tools menu on the Worksheet Menu Bar. This subroutine is executed when the workbook (or add-in) is opened.

The `Workbook_BeforeClose` subroutine is executed when the add-in is closed. This subroutine removes the Toggle Settings menu item from the Tools menu.

UserForm1

Figure 40-3 shows the `UserForm1` form, which has ten controls: nine check boxes and one command button. The controls have descriptive names, and the `Accelerator` property is set so that the controls display an accelerator key (for keyboard users).

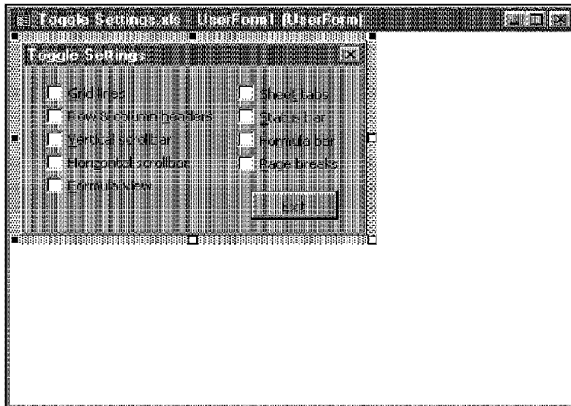


Figure 40-3: The custom dialog box.

The `UserForm1` object contains the event-handler subroutines for the objects that are on the form. The following subroutine is executed before the dialog box is displayed:

```
Private Sub UserForm_Initialize()
    cbGridlines = ActiveWindow.DisplayGridlines
    cbHeaders = ActiveWindow.DisplayHeadings
    cbVerticalScrollbar = ActiveWindow.DisplayVerticalScrollbar
    cbHorizontalScrollbar =
ActiveWindow.DisplayHorizontalScrollbar
    cbFormulaView = ActiveWindow.DisplayFormulas
    cbSheetTabs = ActiveWindow.DisplayWorkbookTabs
    cbStatusBar = Application.DisplayStatusBar
    cbFormulaBar = Application.DisplayFormulaBar
    cbPageBreaks = ActiveSheet.DisplayPageBreaks
End Sub
```

The `UserForm_Initialize` subroutine adjusts the settings of the `CheckBox` controls in the dialog box to correspond to the current settings. For example, if the worksheet is displaying gridlines, `ActiveWindow.DisplayGridlines` returns `True`. This value is assigned to the `cbGridlines` `CheckBox`—which means that the `CheckBox` is displayed with a check mark.

Each `CheckBox` also has an event-handler subroutine, listed in the following code, that is executed when the control is clicked. Each subroutine makes the appropriate changes. For example, if the `Grid lines` `CheckBox` is selected, the `DisplayGridlines` property is set to correspond to the `CheckBox`.


```
Private Sub cbGridlines_Click on()  
    ActiveWindow.DisplayGridlines = cbGridlines  
End Sub  
  
Private Sub cbHeaders_Click on()  
    ActiveWindow.DisplayHeadings = cbHeaders  
End Sub  
  
Private Sub cbVerticalScrollbar_Click on()  
    ActiveWindow.DisplayVerticalScrollbar = cbVerticalScrollbar  
End Sub  
  
Private Sub cbHorizontalScrollbar_Click on()  
    ActiveWindow.DisplayHorizontalScrollbar =  
    cbHorizontalScrollbar  
End Sub  
  
Private Sub cbFormulaView_Click on()  
    ActiveWindow.DisplayFormulas = cbFormulaView  
End Sub  
  
Private Sub cbSheetTabs_Click on()  
    ActiveWindow.DisplayWorkbookTabs = cbSheetTabs  
End Sub  
  
Private Sub cbStatusBar_Click on()  
    Application.DisplayStatusBar = cbStatusBar  
End Sub  
  
Private Sub cbFormulaBar_Click on()  
    Application.DisplayFormulaBar = cbFormulaBar  
End Sub  
  
Private Sub cbPageBreaks_Click on()  
    ActiveSheet.DisplayPageBreaks = cbPageBreaks  
End Sub
```

The `UserForm1` object has one additional event-handler subroutine for the Exit button. This subroutine, listed as follows, simply closes the dialog box:

```
Private Sub ExitButton_Click on()  
    Unload UserForm1  
End Sub
```

Testing the Workbook

Before you convert this workbook to an add-in, you need to test it. You should test it when a different workbook is active, to simulate what happens when the workbook is an add-in. Remember, an add-in is never the active workbook and it never displays any of its worksheets.

Adding Descriptive Information

This step is recommended but not necessary. Choose File • Properties to bring up the Properties dialog box. Then, click the Summary tab, as shown in Figure 40-6.

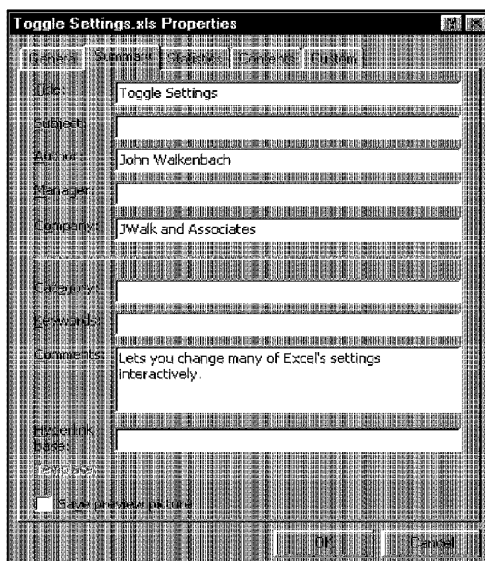


Figure 40-6: Use the Properties dialog box to enter descriptive information about your add-in.

Enter a title for the add-in in the Title field. This is the text that appears in the Add-Ins dialog box. In the Comments field, enter a description. This information appears at the bottom of the Add-Ins dialog box when the add-in is selected.

Protecting the Project

One advantage of an add-in is that it can be protected so that others can't see the source code. If you want to protect the project, follow these steps:

1. Activate the Visual Basic Editor.
2. In the Project window, click the project.
3. Select Tools • [project name] Properties.
VBE displays its Project Properties dialog box.
4. Click the Protection tab (see Figure 40-7).

5. Select the Lock project for viewing check box.
6. Enter a password (twice) for the project.
7. Click OK.

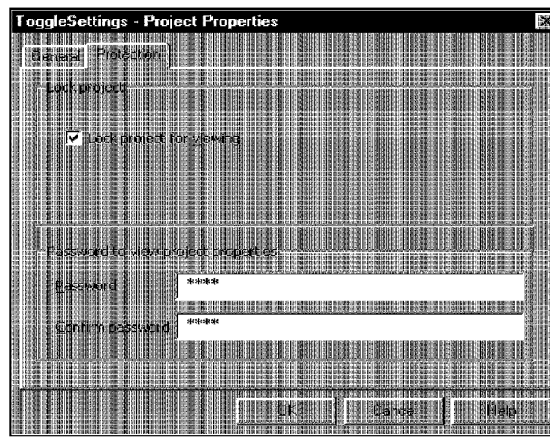


Figure 40-7: The Project Properties dialog box.

Creating the Add-In

To save the workbook as an add-in, activate Excel, make sure the workbook is active, and then choose **File • Save As**. Select **Microsoft Excel Add-In (*.xla)** from the **Save as Type** drop-down list. Enter a name for the add-in file and then click **OK**.

Opening the Add-In

To avoid confusion, close the XLS workbook before you open the add-in that was created from it. Then, select **Tools • Add-Ins**. Excel displays its **Add-Ins** dialog box. Click the **Browse** button and locate the add-in that you just created. After you do so, the **Add-Ins** dialog box displays the add-in in its list. Notice that the information that you provided in the **Properties** dialog box appears here (see Figure 40-8). Click **OK** to close the dialog box and open the add-in.

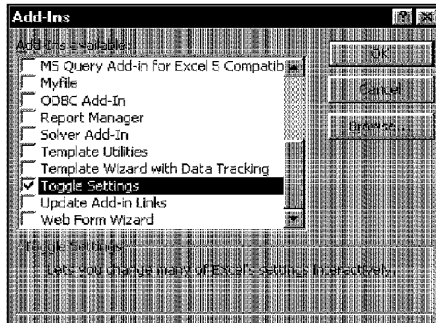


Figure 40-8: The Add-Ins dialog box, with the new add-in selected.

When the add-in is open, the Tools menu displays a new menu item (Toggle Settings) that executes the `ShowToggleSettingsDialog` subroutine in the add-in.

If you activate the VBE window, you find that the add-in is listed in the Project window. However, you can't make any modifications unless you provide the password.

Summary

This chapter discusses the concept of add-ins — files that add new capabilities to Excel — and explains how to work with add-ins and why you may want to create custom add-ins. The chapter closes with an example of an add-in that enables users easily to toggle on and off several Excel settings.

• •

Using Online Help: A Primer

Excel's online Help system has always been good. But the Help available with Excel 2000 is better than ever. However, the online Help system can be a bit intimidating for beginners, because you can get help in many ways. This appendix assists you in getting the most out of this valuable resource.

Why Online Help?

In the early days of personal computing, software programs usually came bundled with bulky manuals that described how to use the product. Some products included rudimentary help that could be accessed online. Over the years, that situation gradually changed. Now, online help is usually the *primary* source of documentation, which may be augmented by a written manual.

After you become accustomed to it, you'll find that online help (if it's done well) offers many advantages over written manuals:

- You don't have to lug around a manual—especially important for laptop users who do their work on the road.
- You don't have to thumb through a separate manual, which often has a confusing index.
- You can search for specific words and then select a topic that's appropriate to your question.
- In some cases (for example, writing VBA code), you can copy examples from the Help window and paste them into your application.
- Help sometimes includes embedded buttons that you can click to go directly to the command that you need.

A P P E N D I X

Types of Help

Excel offers several types of online Help:

- **Tooltips:** Move the mouse pointer over a toolbar button and the button's name appears.
- **Office Assistant:** The animated Office Assistant monitors your actions while you work. If a more efficient way to perform an operation exists, the Assistant can tell you about it.
- **Dialog box help:** When a dialog box is displayed, click the Help button in the title bar (it has a question mark on it) and then click any part of the dialog box. Excel pops up a description of the selected control. Figure A-1 shows an example.



Figure A-1: Getting a description of a dialog box control.

- **“What’s This” help:** Press Shift+F1, and the mouse pointer turns into a question mark. You can then click virtually any part of the screen to get a description of the object.
- **1-2-3 help:** The Help • Lotus 1-2-3 Help command provides help designed for those who are familiar with 1-2-3’s commands.
- **Internet-based help:** You can access a variety of Internet resources directly from Excel.
- **Detailed help:** This is what’s usually considered online help. As you’ll see, you have several ways to locate a particular Help topic.

Accessing Help

When you work with Excel 2000, you can access the online Help system by using the Help menu, shown in Figure A-2. The various options are described in the sections that follow.

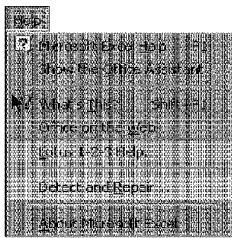


Figure A-2: The Help menu.

The Office Assistant

Selecting Microsoft Excel Help displays the Office Assistant, shown in Figure A-3. Type a brief description of the subject about which you want help, and the Assistant displays a list of Help topics. Chances are good that one of these topics will lead to the help that you need; click a list item to view a Help topic.

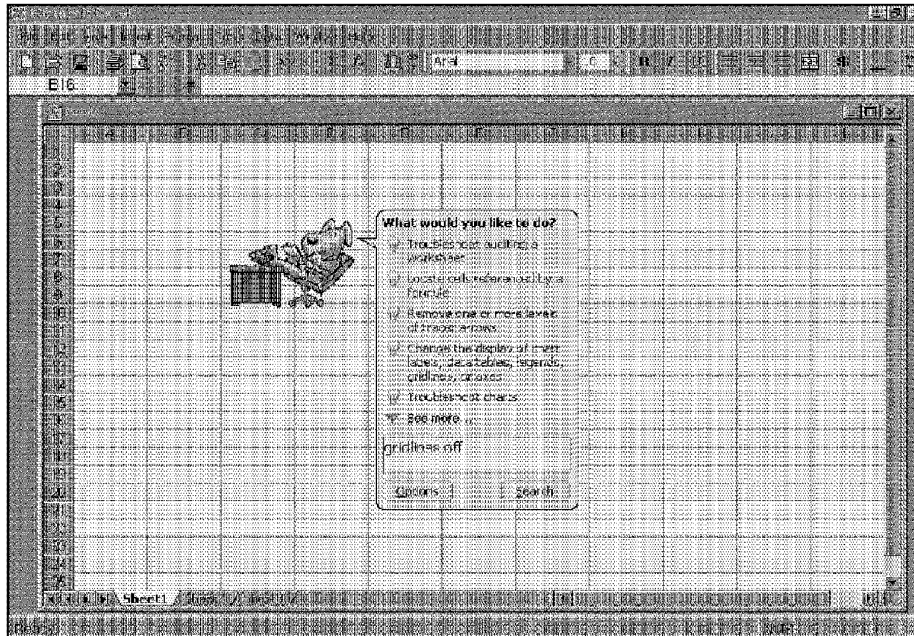


Figure A-3: The Office Assistant.

The information that you type doesn't have to be in the form of a question. Rather, you can simply enter one or more keywords that describe the topic. For example, if you want to find out how to turn off gridlines, you can type **gridlines off**.



You have a great deal of control over the Office Assistant. Right-click the Assistant and select Options from the shortcut menu. Excel displays the dialog box shown in Figure A-4. The Gallery tab lets you select a new character for the Assistant. The Options tab lets you determine whether to use the Assistant and, if you do, how the Assistant behaves. If you find that the Office Assistant is distracting, remove the check from the Use the Office Assistant check box. You can turn on the Office Assistant again by choosing the Show the Office Assistant command on the Help menu.

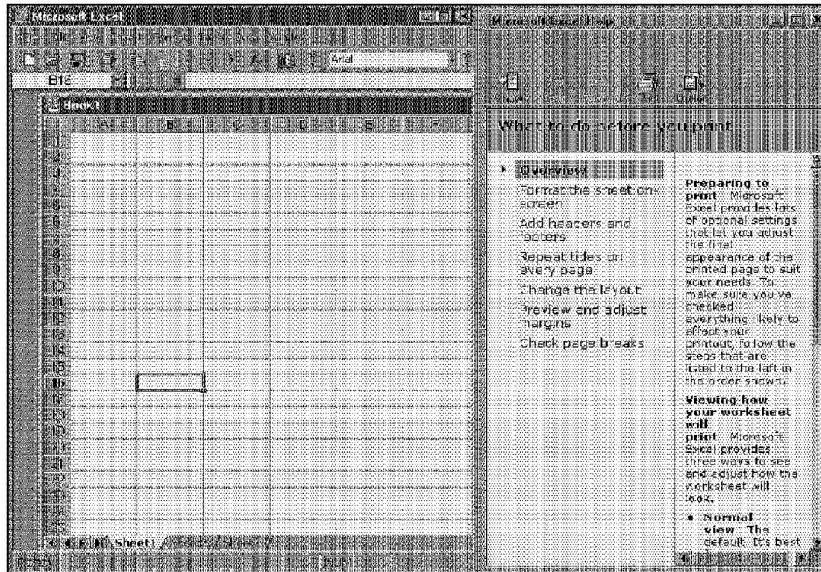


Figure A-5: The Help window tiles to the right of the program window so that you can view Help while working.

Contents Tab

Figure A-6 shows the Contents tab. This tab is arranged alphabetically by subject; you can compare the Contents tab to the table of contents in a book, because they both organize information by similar topic. When you double-click a book icon (or single-click the plus sign to the left of the book icon), the book expands to show Help topics (each with a question-mark icon). To close a book, double-click it again or single-click the minus sign to the left of the book. To display a Help topic, single-click the topic title.

The Help topic remains onscreen until you either close Help or select another Help topic.

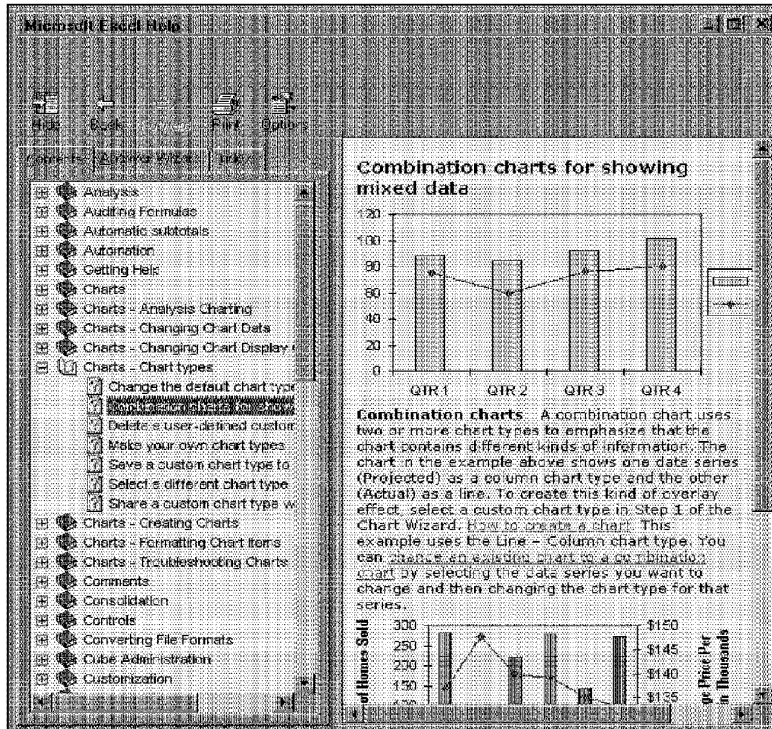


Figure A-6: The Contents tab.

Answer Wizard Tab

The Answer Wizard tab works in much the same way as the Office Assistant works. Type a question or some words related to the subject about which you want help, and then click the Search button (see Figure A-7). Topics appear at the bottom of the window. Double-click a topic in the bottom of the window and the Help topic appears in the right pane of the Help window.

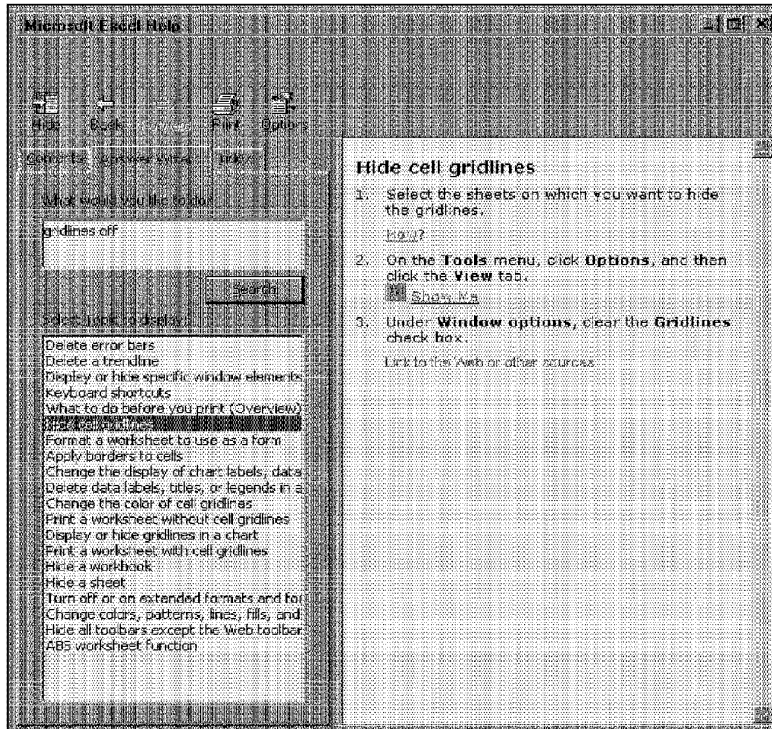


Figure A-7: The Answer Wizard tab of the Help window.

Index Tab

Figure A-8 shows the Index tab of the Help Topics dialog box. The keywords are arranged alphabetically, much like an index for a book. You can enter in the box at the top the first few letters of a keyword for which you'd like to search. Click the Search button to display related topics at the bottom of the box. Double-click a topic at the bottom of the box to display it in the right pane of the Help window.

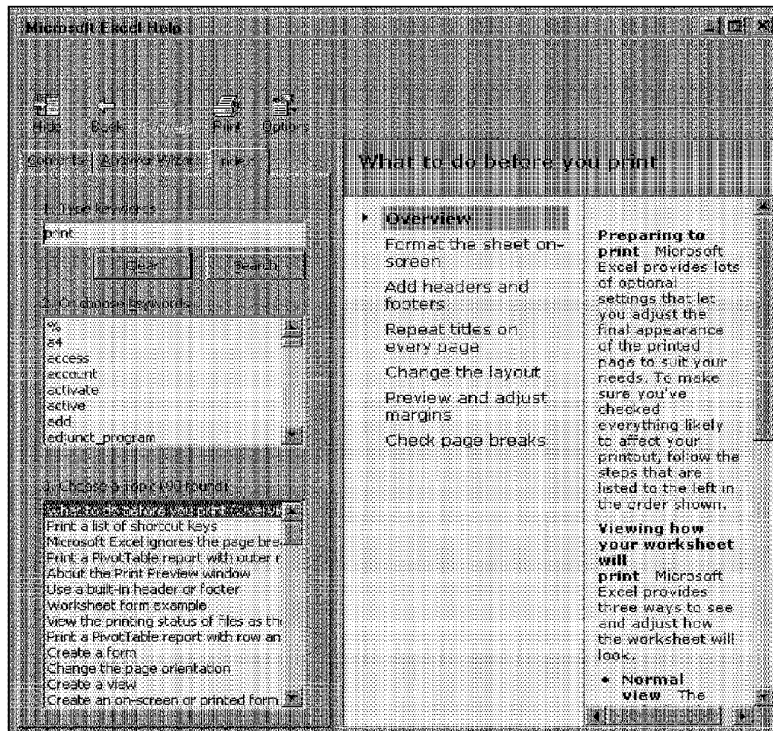


Figure A-8: The Index tab of the Help topics dialog box.

Mastering Help

After you select a Help topic, you can navigate through Help in the same way that you use a browser to navigate on the Web. The Back and Forward buttons let you view Help topics that you previously viewed, in the order that you viewed them. Use the Print button to print a Help topic. Click the Options button to display a drop-down menu that contains commands that perform the same functions as the Show, Hide, Back, Forward, and Print buttons. You'll also find a Stop command and a Refresh command; you can use these if you connect to the Web for Help and want to stop loading a page or refresh the Web page you're viewing.

The information provided in this appendix gets you started using Excel's online Help. Everyone develops his or her own style for using this help, and I urge you to explore this resource. Even if you think you understand a topic in Excel fairly well, you can often discover one or two subtle features that you didn't know about. A thorough understanding of how to use the online Help system will definitely make you a more productive Excel user.

• • • • •

Worksheet Function Reference

This appendix contains a complete listing of Excel's worksheet functions. The functions are arranged alphabetically by categories used by the Paste Function dialog box. Some of these functions (indicated in the lists that follow) are available only when a particular add-in is attached.

For more information about a particular function, including its arguments, select the function in the Function Wizard and click the Help button.

Table B-1
Database Category Functions

<i>Function</i>	<i>What It Does</i>
DAVERAGE	Returns the average of selected database entries
DCOUNT	Counts the cells containing numbers from a specified database and criteria
DCOUNTA	Counts nonblank cells from a specified database and criteria
DGET	Extracts from a database a single record that matches the specified criteria
DMAX	Returns the maximum value from selected database entries

Continued

B
A P P E N D I X

Table B-1 (continued)

Function	What It Does
DMIN	Returns the minimum value from selected database entries
DPRODUCT	Multiplies the values in a particular field of records that match the criteria in a database
DSTDEV	Estimates the standard deviation based on a sample of selected database entries
DSTDEVP	Calculates the standard deviation based on the entire population of selected database entries
DSUM	Adds the numbers in the field column of records in the database that match the criteria
DVAR	Estimates variance based on a sample from selected database entries
DVARP	Calculates variance based on the entire population of selected database entries
SQL.CLOSE**	Terminates a SQL.OPEN connection
SQL.BIND**	Specifies where to place SQL.EXEC.QUERY results
SQL.ERROR**	Returns error information on SQL* functions
SQL.EXEC.QUERY**	Executes a SQL statement on a SQL.OPEN connection
QUERYGETDATA***	Gets external data using Microsoft Query
QUERYGETDATADIALOG***	Displays a dialog box to get data using Microsoft Query
SQL.GET.SCHEMA**	Returns information on a SQL.OPEN connection
SQL.OPEN**	Makes a connection to a data source via ODBC
QUERYREFRESH***	Updates a data range using Microsoft Query
SQL.REQUEST**	Requests a connection and executes a SQL query
SQL.RETRIEVE**	Retrieves SQL.EXEC.QUERY results
SQL.RETRIEVE.TO.FILE**	Retrieves SQL.EXEC.QUERY results to a file

* Available only when the Analysis ToolPak add-in is attached

** Available only when the ODBC add-in is attached

*** Available only when the MS Query add-in is attached

**Table B-2
Date and Time Category Functions**

Function	What It Does
DATE	Returns the serial number of a particular date
DATEVALUE	Converts a date in the form of text to a serial number
DAY	Converts a serial number to a day of the month
DAYS360	Calculates the number of days between two dates, based on a 360-day year
EDATE*	Returns the serial number of the date that is the indicated number of months before or after the start date
EOMONTH*	Returns the serial number of the last day of the month before or after a specified number of months
HOUR	Converts a serial number to an hour
MINUTE	Converts a serial number to a minute
MONTH	Converts a serial number to a month
NETWORKDAYS*	Returns the number of whole workdays between two dates
NOW	Returns the serial number of the current date and time
SECOND	Converts a serial number to a second
TIME	Returns the serial number of a particular time
TIMEVALUE	Converts a time in the form of text to a serial number
TODAY	Returns the serial number of today's date
WEEKDAY	Converts a serial number to a day of the week
WEEKNUM*	Returns the week number in the year
WORKDAY*	Returns the serial number of the date before or after a specified number of workdays
YEAR	Converts a serial number to a year
YEARFRAC*	Returns the year fraction representing the number of whole days between start_date and end_date

* Available only when the Analysis ToolPak add-in is attached

**Table B-3
Engineering Category Functions**

<i>Function</i>	<i>What It Does</i>
BESSELI*	Returns the modified Bessel function $I_n(x)$
BESSELJ*	Returns the Bessel function $J_n(x)$
BESSELK*	Returns the modified Bessel function $K_n(x)$
BESSELY*	Returns the Bessel function $Y_n(x)$
BIN2DEC*	Converts a binary number to decimal
BIN2HEX*	Converts a binary number to hexadecimal
BIN2OCT*	Converts a binary number to octal
COMPLEX*	Converts real and imaginary coefficients into a complex number
CONVERT*	Converts a number from one measurement system to another
DEC2BIN*	Converts a decimal number to binary
DEC2HEX*	Converts a decimal number to hexadecimal
DEC2OCT*	Converts a decimal number to octal
DELTA*	Tests whether two values are equal
ERF*	Returns the error function
ERFC*	Returns the complementary error function
GESTEP*	Tests whether a number is greater than a threshold value
HEX2BIN*	Converts a hexadecimal number to binary
HEX2DEC*	Converts a hexadecimal number to decimal
HEX2OCT*	Converts a hexadecimal number to octal
IMABS*	Returns the absolute value (modulus) of a complex number
IMAGINARY*	Returns the imaginary coefficient of a complex number
IMARGUMENT*	Returns the argument theta, an angle expressed in radians
IMCONJUGATE*	Returns the complex conjugate of a complex number
IMCOS*	Returns the cosine of a complex number
IMDIV*	Returns the quotient of two complex numbers
IMEXP*	Returns the exponential of a complex number
IMLN*	Returns the natural logarithm of a complex number
IMLOG2*	Returns the base-2 logarithm of a complex number
IMLOG10*	Returns the base-10 logarithm of a complex number

Function	What It Does
IMPOWER*	Returns a complex number raised to an integer power
IMPRODUCT*	Returns the product of two complex numbers
IMREAL*	Returns the real coefficient of a complex number
IMSIN*	Returns the sine of a complex number
IMSQRT*	Returns the square root of a complex number
IMSUB*	Returns the difference of two complex numbers
ISUM*	Returns the sum of complex numbers
OCT2BIN*	Converts an octal number to binary
OCT2DEC*	Converts an octal number to decimal
OCT2HEX*	Converts an octal number to hexadecimal

* Available only when the Analysis ToolPak add-in is attached

Table B-4
Financial Category Functions

Function	What It Does
ACCRINT*	Returns the accrued interest for a security that pays periodic interest
ACCRINTM*	Returns the accrued interest for a security that pays interest at maturity
AMORDEGRC*	Returns the depreciation for each accounting period
AMORLINC*	Returns the depreciation for each accounting period
COUPDAYBS*	Returns the number of days from the beginning of the coupon period to the settlement date
COUPDAYS*	Returns the number of days in the coupon period that contains the settlement date
COUPDAYSNC*	Returns the number of days from the settlement date to the next coupon date
COUPNCD*	Returns the next coupon date after the settlement date
COUPNUM*	Returns the number of coupons payable between the settlement date and maturity date
COUPPCD*	Returns the previous coupon date before the settlement date
CUMIPMT*	Returns the cumulative interest paid between two periods

Continued

Table B-4 (continued)

Function	What It Does
CUMPRINC*	Returns the cumulative principal paid on a loan between two periods
DB	Returns the depreciation of an asset for a specified period, using the fixed-declining balance method
DDB	Returns the depreciation of an asset for a specified period, using the double-declining balance method or some other method that you specify
DISC*	Returns the discount rate for a security
DOLLARDE*	Converts a dollar price, expressed as a fraction, into a dollar price, expressed as a decimal number
DOLLARFR*	Converts a dollar price, expressed as a decimal number, into a dollar price, expressed as a fraction
DURATION*	Returns the annual duration of a security with periodic interest payments
EFFECT*	Returns the effective annual interest rate
FV	Returns the future value of an investment
FVSCHEDULE*	Returns the future value of an initial principal after applying a series of compound interest rates
INTRATE*	Returns the interest rate for a fully invested security
IPMT	Returns the interest payment for an investment for a given period
IRR	Returns the internal rate of return for a series of cash flows
ISPMT	Returns the interest associated with a specific loan payment.
MDURATION*	Returns the Macauley modified duration for a security with an assumed par value of \$100
MIRR	Returns the internal rate of return where positive and negative cash flows are financed at different rates
NOMINAL*	Returns the annual nominal interest rate
NPER	Returns the number of periods for an investment
NPV	Returns the net present value of an investment based on a series of periodic cash flows and a discount rate
ODDFPRICE*	Returns the price per \$100 face value of a security with an odd first period
ODDFYIELD*	Returns the yield of a security with an odd first period

Function	What It Does
ODDLPRICE*	Returns the price per \$100 face value of a security with an odd last period
ODDLYIELD*	Returns the yield of a security with an odd last period
PMT	Returns the periodic payment for an annuity
PPMT	Returns the payment on the principal for an investment for a given period
PRICE*	Returns the price per \$100 face value of a security that pays periodic interest
PRICEDISC*	Returns the price per \$100 face value of a discounted security
PRICEMAT*	Returns the price per \$100 face value of a security that pays interest at maturity
PV	Returns the present value of an investment
RATE	Returns the interest rate per period of an annuity
RECEIVED*	Returns the amount received at maturity for a fully invested security
SLN	Returns the straight-line depreciation of an asset for one period
SYD	Returns the sum-of-years' digits depreciation of an asset for a specified period
TBILLEQ*	Returns the bond-equivalent yield for a Treasury bill
TBILLPRICE*	Returns the price per \$100 face value for a Treasury bill
TBILLYIELD*	Returns the yield for a Treasury bill
VDB	Returns the depreciation of an asset for a specified or partial period using a declining balance method
XIRR*	Returns the internal rate of return for a schedule of cash flows that is not necessarily periodic
XNPV*	Returns the net present value for a schedule of cash flows that is not necessarily periodic
YIELD*	Returns the yield on a security that pays periodic interest
YIELDDISC*	Returns the annual yield for a discounted security; for example, a Treasury bill
YIELDMAT*	Returns the annual yield of a security that pays interest at maturity

* Available only when the Analysis ToolPak add-in is attached

**Table B-5
Information Category Functions**

<i>Function</i>	<i>What It Does</i>
CELL	Returns information about the formatting, location, or contents of a cell
COUNTBLANK	Counts the number of blank cells within a range
ERROR.TYPE	Returns a number corresponding to an error type
INFO	Returns information about the current operating environment
ISBLANK	Returns TRUE if the value is blank
ISERR	Returns TRUE if the value is any error value except #N/A
ISERROR	Returns TRUE if the value is any error value
ISEVEN*	Returns TRUE if the number is even
ISLOGICAL	Returns TRUE if the value is a logical value
ISNA	Returns TRUE if the value is the #N/A error value
ISNONTEXT	Returns TRUE if the value is not text
ISNUMBER	Returns TRUE if the value is a number
ISODD*	Returns TRUE if the number is odd
ISREF	Returns TRUE if the value is a reference
ISTEXT	Returns TRUE if the value is text
N	Returns a value converted to a number
NA	Returns the error value #N/A
TYPE	Returns a number indicating the data type of a value

* Available only when the Analysis ToolPak add-in is attached

**Table B-6
Logical Category Functions**

<i>Function</i>	<i>What It Does</i>
AND	Returns TRUE if all of its arguments are TRUE
FALSE	Returns the logical value FALSE
IF	Specifies a logical test to perform
NOT	Reverses the logic of its argument
OR	Returns TRUE if any argument is TRUE
TRUE	Returns the logical value TRUE

Table B-7
Lookup and Reference Category Functions

<i>Function</i>	<i>What It Does</i>
ADDRESS	Returns a reference as text to a single cell in a worksheet
AREAS	Returns the number of areas in a reference
CHOOSE	Chooses a value from a list of values
COLUMN	Returns the column number of a reference
COLUMNS	Returns the number of columns in a reference
GETPIVOTDATA	Returns data stored in a PivotTable
HLOOKUP	Looks in the top row of an array and returns the value of the indicated cell
HYPERLINK	Creates a shortcut that opens a document on your hard drive, a server, or the Internet
INDEX	Uses an index to choose a value from a reference or array
INDIRECT	Returns a reference indicated by a text value
LOOKUP	Looks up values in a vector or array
MATCH	Looks up values in a reference or array
OFFSET	Returns a reference offset from a given reference
ROW	Returns the row number of a reference
ROWS	Returns the number of rows in a reference
TRANSPOSE	Returns the transpose of an array
VLOOKUP	Looks in the first column of an array and moves across the row to return the value of a cell

Table B-8
Math and Trig Category Functions

<i>Function</i>	<i>What It Does</i>
ABS	Returns the absolute value of a number
ACOS	Returns the arccosine of a number
ACOSH	Returns the inverse hyperbolic cosine of a number
ASIN	Returns the arcsine of a number
ASINH	Returns the inverse hyperbolic sine of a number

Continued

Table B-8 (continued)

Function	What It Does
ATAN	Returns the arctangent of a number
ATAN2	Returns the arctangent from x and y coordinates
ATANH	Returns the inverse hyperbolic tangent of a number
CEILING	Rounds a number to the nearest integer or to the nearest multiple of significance
COMBIN	Returns the number of combinations for a given number of objects
COS	Returns the cosine of a number
COSH	Returns the hyperbolic cosine of a number
COUNTIF	Counts the number of nonblank cells within a range that meets the given criteria
DEGREES	Converts radians to degrees
EVEN	Rounds a number up to the nearest even integer
EXP	Returns e raised to the power of a given number
FACT	Returns the factorial of a number
FACTDOUBLE	Returns the double factorial of a number
FLOOR	Rounds a number down, toward 0
GCD*	Returns the greatest common divisor
INT	Rounds a number down to the nearest integer
LCM*	Returns the least common multiple
LN	Returns the natural logarithm of a number
LOG	Returns the logarithm of a number to a specified base
LOG10	Returns the base-10 logarithm of a number
MDTERM	Returns the matrix determinant of an array
MINVERSE	Returns the matrix inverse of an array
MMULT	Returns the matrix product of two arrays
MOD	Returns the remainder from division
MROUND*	Returns a number rounded to the desired multiple
MULTINOMIAL*	Returns the multinomial of a set of numbers
ODD	Rounds a number up to the nearest odd integer

Function	What It Does
PI	Returns the value of pi
POWER	Returns the result of a number raised to a power
PRODUCT	Multiplies its arguments
QUOTIENT*	Returns the integer portion of a division
RADIANS	Converts degrees to radians
RAND	Returns a random number between 0 and 1
RANDBETWEEN*	Returns a random number between the numbers that you specify
ROMAN	Converts an Arabic numeral to Roman, as text
ROUND	Rounds a number to a specified number of digits
ROUNDDOWN	Rounds a number down, toward 0
ROUNDUP	Rounds a number up, away from 0
SERIESSUM*	Returns the sum of a power series based on the formula
SIGN	Returns the sign of a number
SIN	Returns the sine of the given angle
SINH	Returns the hyperbolic sine of a number
SQRT	Returns a positive square root
SQRTPI*	Returns the square root of (<i>number</i> * pi)
SUBTOTAL	Returns a subtotal in a list or database
SUM	Adds its arguments
SUMIF	Adds the cells specified by a given criteria
SUMPRODUCT	Returns the sum of the products of corresponding array components
SUMSQ	Returns the sum of the squares of the arguments
SUMX2MY2	Returns the sum of the difference of squares of corresponding values in two arrays
SUMX2PY2	Returns the sum of the sum of squares of corresponding values in two arrays
SUMXMY2	Returns the sum of squares of differences of corresponding values in two arrays
TAN	Returns the tangent of a number
TANH	Returns the hyperbolic tangent of a number
TRUNC	Truncates a number to an integer

* Available only when the Analysis ToolPak add-in is attached

Table B-9
Statistical Category Functions

<i>Function</i>	<i>What It Does</i>
AVEDEV	Returns the average of the absolute deviations of data points from their mean
AVERAGE	Returns the average of its arguments
AVERAGEA	Returns the average of its arguments and includes evaluation of text and logical values
BETADIST	Returns the cumulative beta probability density function
BETAINV	Returns the inverse of the cumulative beta probability density function
BINOMDIST	Returns the individual term binomial distribution probability
CHIDIST	Returns the one-tailed probability of the chi-squared distribution
CHIINV	Returns the inverse of the one-tailed probability of the chi-squared distribution
CHITEST	Returns the test for independence
CONFIDENCE	Returns the confidence interval for a population mean
CORREL	Returns the correlation coefficient between two data sets
COUNT	Counts how many numbers are in the list of arguments
COUNTA	Counts how many values are in the list of arguments
COUNTBLANK	Counts the number of blank cells in the argument range
COUNTIF	Counts the number of cells that meet the criteria you specify in the argument
COVAR	Returns covariance, the average of the products of paired deviations
CRITBINOM	Returns the smallest value for which the cumulative binomial distribution is less than or equal to a criterion value
DEVSQ	Returns the sum of squares of deviations
EXPONDIST	Returns the exponential distribution
FDIST	Returns the F probability distribution
FINV	Returns the inverse of the F probability distribution
FISHER	Returns the Fisher transformation
FISHERINV	Returns the inverse of the Fisher transformation
FORECAST	Returns a value along a linear trend
FREQUENCY	Returns a frequency distribution as a vertical array
FTEST	Returns the result of an F-test

Function	What It Does
GAMMADIST	Returns the gamma distribution
GAMMAINV	Returns the inverse of the gamma cumulative distribution
GAMMALN	Returns the natural logarithm of the gamma function, $\Gamma(x)$
GEOMEAN	Returns the geometric mean
GROWTH	Returns values along an exponential trend
HARMEAN	Returns the harmonic mean
HYPGEOMDIST	Returns the hypergeometric distribution
INTERCEPT	Returns the intercept of the linear regression line
KURT	Returns the kurtosis of a data set
LARGE	Returns the k th largest value in a data set
LINEST	Returns the parameters of a linear trend
LOGEST	Returns the parameters of an exponential trend
LOGINV	Returns the inverse of the lognormal distribution
LOGNORMDIST	Returns the cumulative lognormal distribution
MAX	Returns the maximum value in a list of arguments, ignoring logical values and text
MAXA	Returns the maximum value in a list of arguments, including logical values and text
MEDIAN	Returns the median of the given numbers
MIN	Returns the minimum value in a list of arguments, ignoring logical values and text
MINA	Returns the minimum value in a list of arguments, including logical values and text
MODE	Returns the most common value in a data set
NEGBINOMDIST	Returns the negative binomial distribution
NORMDIST	Returns the normal cumulative distribution
NORMINV	Returns the inverse of the normal cumulative distribution
NORMSDIST	Returns the standard normal cumulative distribution
NORMSINV	Returns the inverse of the standard normal cumulative distribution
PEARSON	Returns the Pearson product moment correlation coefficient
PERCENTILE	Returns the k th percentile of values in a range
PERCENTRANK	Returns the percentage rank of a value in a data set

Continued

Table B-9 (continued)

Function	What It Does
PERMUT	Returns the number of permutations for a given number of objects
POISSON	Returns the Poisson distribution
PROB	Returns the probability that values in a range are between two limits
QUARTILE	Returns the quartile of a data set
RANK	Returns the rank of a number in a list of numbers
RSQ	Returns the square of the Pearson product moment correlation coefficient
SKEW	Returns the skewness of a distribution
SLOPE	Returns the slope of the linear regression line
SMALL	Returns the <i>k</i> th smallest value in a data set
STANDARDIZE	Returns a normalized value
STDEV	Estimates standard deviation based on a sample, ignoring text and logical values
STDEVA	Estimates standard deviation based on a sample, including text and logical values
STDEV.P	Calculates standard deviation based on the entire population, ignoring text and logical values
STDEVPA	Calculates standard deviation based on the entire population, including text and logical values
STEYX	Returns the standard error of the predicted <i>y</i> -value for each <i>x</i> in the regression
TDIST	Returns the student's <i>t</i> -distribution
TINV	Returns the inverse of the student's <i>t</i> -distribution
TREND	Returns values along a linear trend
TRIMMEAN	Returns the mean of the interior of a data set
TTEST	Returns the probability associated with a <i>student's t-Test</i>
VAR	Estimates variance based on a sample, ignoring logical values and text
VARA	Estimates variance based on a sample, including logical values and text
VARP	Calculates variance based on the entire population, ignoring logical values and text
VARPA	Calculates variance based on the entire population, including logical values and text
WEIBULL	Returns the Weibull distribution
ZTEST	Returns the two-tailed P-value of a <i>z</i> -test

Excel's Shortcut Keys

This appendix lists the most useful shortcut keys that are available in Excel. The shortcuts are arranged by context.

The keys listed assume that you are not using the Transition Navigation Keys, which are designed to emulate Lotus 1-2-3. You can select this option in the Transition tab of the Options dialog box.

**Table C-1
Moving Through a Worksheet**

<i>Key(s)</i>	<i>What It Does</i>
Arrow keys	Move left, right, up, or down one cell
Home	Moves to the beginning of the row
Home*	Moves to the upper-left cell displayed in the window
End*	Moves to the lower-left cell displayed in the window
Arrow keys*	Scrolls left, right, up, or down one cell
PgUp	Moves up one screen
Ctrl+PgUp	Moves to the previous sheet
PgDn	Moves down one screen
Ctrl+PgDn	Moves to the next sheet
Alt+PgUp	Moves one screen to the left
Alt+PgDn	Moves one screen to the right
Ctrl+Home	Moves to the first cell in the worksheet (A1)
Ctrl+End	Moves to the last active cell of the worksheet

Continued

Table C-1 (continued)

Key(s)	What It Does
Ctrl+arrow key	Moves to the edge of a data block; if the cell is blank, moves to the first nonblank cell
Ctrl+Backspace	Scrolls to display the active cell
End+Home	Moves to the last nonempty cell on the worksheet
F5	Prompts for a cell address to go to
F6	Moves to the next pane of a workbook that has been split
Shift+F6	Moves to the previous pane of a workbook that has been split
Ctrl+Tab	Moves to the next window
Ctrl+Shift+Tab	Moves to the previous window

* With Scroll Lock on

Table C-2
Selecting Cells in the Worksheet

Key(s)	What It Does
Shift+arrow key	Expands the selection in the direction indicated
Shift+spacebar	Selects the entire row
Ctrl+spacebar	Selects the entire column
Ctrl+Shift+	Selects the entire worksheet spacebar
Shift+Home	Expands the selection to the beginning of the current row
Ctrl+*	Selects the block of data surrounding the active cell
F8	Extends the selection as you use navigation keys
Shift+F8	Adds other nonadjacent cells or ranges to the selection; pressing Shift+F8 again ends Add mode
F5	Prompts for a range or range name to select
Ctrl+G	Prompts for a range or range name to select
Ctrl+A	Selects the entire worksheet
Shift+Backspace	Selects the active cell in a range selection

Table C-3
Moving Within a Range Selection

<i>Key(s)</i>	<i>What It Does</i>
Enter	Moves the cell pointer down to the next cell in the selection
Shift+Enter	Moves the cell pointer up to the preceding cell in the selection
Tab	Moves the cell pointer right to the next cell in the selection
Shift+Tab	Moves the cell pointer left to the preceding cell in the selection
Ctrl+period (.)	Moves the cell pointer to the next corner of the current cell range
Ctrl+Tab	Moves the cell pointer to the next cell range in a nonadjacent selection
Ctrl+Shift+Tab	Moves the cell pointer to the previous cell range in a nonadjacent selection
Shift+Backspace	Collapses the cell selection to just the active cell

Table C-4
Editing Keys in the Formula Bar

<i>Key(s)</i>	<i>What It Does</i>
F2	Begins editing the active cell
F3	Pastes a name into a formula
Arrow keys	Moves the cursor one character in the direction of the arrow
Home	Moves the cursor to the beginning of the line
Esc	Cancel the editing
End	Moves the cursor to the end of the line
Ctrl+right arrow	Moves the cursor one word to the right
Ctrl+left arrow	Moves the cursor one word to the left
Del	Deletes the character to the right of the cursor
Ctrl+Del	Deletes all characters from the cursor to the end of the line
Backspace	Deletes the character to the left of the cursor

**Table C-5
Formatting Keys**

Key(s)	What It Does
Ctrl+I	Format • [Selected Object]
Ctrl+B	Sets or removes boldface
Ctrl+I	Sets or removes italic
Ctrl+U	Sets or removes underlining
Ctrl+5	Sets or removes strikethrough
Ctrl+Shift+~	Applies the general number format
Ctrl+Shift+!	Applies the comma format with two decimal places
Ctrl+Shift+#	Applies the date format (day, month, year)
Ctrl+Shift+@	Applies the time format (hour, minute, a.m./p.m.)
Ctrl+Shift+\$	Applies the currency format with two decimal places
Ctrl+Shift+%	Applies the percent format with no decimal places
Ctrl+Shift+&	Applies border to outline
Ctrl+Shift+_	Removes all borders
Alt+'	Selects Format • Style

**Table C-6
Other Shortcut Keys**

Key(s)	What It Does
Alt+=	Inserts the AutoSum formula
Alt+Backspace	Selects Edit • Undo
Alt+Enter	Starts a new line in the current cell
Ctrl+;	Enters the current date
Ctrl+0 (zero)	Hides columns
Ctrl+1	Displays the Format dialog box for the selected object
Ctrl+6	Cycles among various ways of displaying objects

Key(s)	What It Does
Ctrl+7	Toggles the display of the standard toolbar
Ctrl+8	Toggles the display of outline symbols
Ctrl+9	Hides rows
Ctrl+A	After typing a function name in a formula, displays the Formula Palette
Ctrl+C	Selects Edit • Copy
Ctrl+D	Selects Edit • Fill Left
Ctrl+Delete	Selects Edit • Cut
Ctrl+F	Selects Edit • Find
Ctrl+H	Selects Edit • Replace
Ctrl+Insert	Selects Edit • Copy
Ctrl+K	Selects Insert • Hyperlink
Ctrl+N	Selects File • New
Ctrl+O	Selects File • Open
Ctrl+P	Selects File • Print
Ctrl+R	Selects Edit • Fill Right
Ctrl+S	Selects File • Save
Ctrl+Shift+(Unhides rows
Ctrl+Shift+)	Unhides columns
Ctrl+Shift+=	Enters the current time
Ctrl+Shift+A	After typing a valid function name in a formula, inserts the argument names and parentheses for the function
Ctrl+V	Selects Edit • Paste
Ctrl+X	Selects Edit • Cut
Ctrl+Z	Selects Edit • Undo
Delete	Selects Edit • Clear
Shift+Insert	Selects Edit • Paste

**Table C-7
Function Keys**

Key(s)	What It Does
F1	Displays Help or the Office Assistant
Shift+F1	Displays the What's This cursor
Alt+F1	Inserts a chart sheet
Alt+Shift+ F1	Inserts a new worksheet
F2	Edits the active cell
Shift+F2	Edits a cell comment
Alt+F2	Issues Save As command
Alt+Shift+F2	Issues Save command
F3	Pastes a name into a formula
Shift+F3	Pastes a function into a formula
Ctrl+F3	Defines a name
Ctrl+Shift+F3	Displays the Creates Names dialog box, to create names using row and column labels
F4	Repeats the last action
Shift+F4	Repeats the last Find (Find Next)
Ctrl+F4	Closes the window
Alt+F4	Exits the program
F5	Displays the Go To dialog box
Shift+F5	Displays the Find dialog box
Ctrl+F5	Restores the window size
F6	Moves to the next pane
Shift+F6	Moves to the previous pane
Ctrl+F6	Moves to the next workbook window
Ctrl+ Shift+F6	Moves to the previous workbook window
F7	Issues Spelling command
Ctrl+F7	Moves the window
F8	Extends a selection
Shift+F8	Adds to the selection
Ctrl+F8	Resizes the window
Alt+F8	Displays the Macro dialog box

Key(s)	What It Does
F9	Calculates all sheets in all open workbooks
Shift+F9	Calculates the active worksheet
Ctrl+F9	Minimizes the workbook
F10	Makes the menu bar active
Shift+F10	Displays a shortcut menu
Ctrl+F10	Maximizes or restores the workbook window
F11	Creates a chart
Shift+F11	Inserts a new worksheet
Ctrl+F11	Inserts an Excel 4.0 macro sheet
Alt+F11	Displays Visual Basic Editor
F12	Issues Save As command
Shift+F12	Issues Save command
Ctrl+F12	Issues Open command
Ctrl+Shift+F12	Issues Print command

•

What's on the CD-ROM

This appendix describes the contents of the companion CD-ROM.

CD-ROM Overview

The CD-ROM consists of four components:

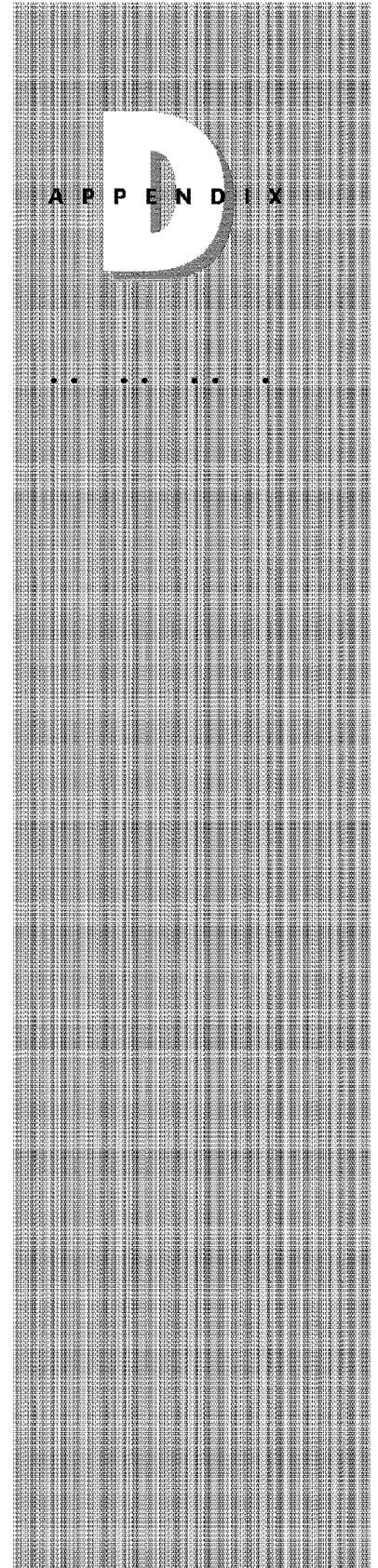
- **Chapter Examples:** Excel workbooks that were discussed in the chapter of this book.
- **Bonus Files:** Additional Excel workbooks and add-ins that you may find useful or instructive. These were all developed by the author
- **Power Utility Pak:** The shareware version of the author's popular Excel add-in. Use the coupon in this book to order the full version, and save \$30.
- **Sound-Proof:** The demo version of the author's audio proofreader add-in.

Chapter Examples

Each chapter of this book that contains example workbooks has its own subdirectory on the CD-ROM. For example, the example files for Chapter 32 will be found in the following directory:

```
chapters\chap32\
```

Following is a list of the chapter examples that follow a brief description of each.



Chapter 3

This workbook contains the end result of the hands on exercise.

handson.xls

Chapter 6

This workbook contains a variety of custom number formats.

formats.xls

Chapter 10

This workbook demonstrates the use of PMT, PPMT, and IPMT functions to calculate a fixed-rate amortization schedule.

amortize.xls

This workbook demonstrates the use of the INDEX and MATCH functions to display the mileage between various cities.

mileage.xls

This workbook demonstrates the use of the INDIRECT function.

indirect.xls

This workbook demonstrates the use of a lengthy “megaformula” to remove the middle names and middle initials from a list of names.

megaform.xls

Chapter 11

This workbook contains many examples of cell and range formatting.

fmtexamp.xls

This workbook contains custom style examples.

styles.xls

Chapter 16

This workbook demonstrates how to create a Gantt chart.

gantt.xls

This workbook demonstrates how to create a comparative histogram.

comphist.xls

This workbook contains a chart that updates automatically when you add new data to the data range.

autochart.xls

Chapter 18

This budgeting workbook demonstrates the use of row and column outlining.

outline.xls

This workbook demonstrates the use of an outline to display various levels of text.

textout.xls

Chapter 20

This workbook demonstrates some uses for array formulas.

arrays.xls

Chapter 24

The dBASE file is used for the examples in this chapter.

budget.dbf

Chapter 25

This workbook is used for several pivot table examples.

banking.xls

These four files are used in the pivot table consolidation example.

consolid.xls, file1.xls, file2.xls, file3.xls

This workbook demonstrates pivot charts.

pivchart.xls

This workbook demonstrates a survey data analysis using pivot tables.

survey.xls

This workbook demonstrates a geographic analysis using a pivot table.

geog.xls

This workbook demonstrates how to group pivot table data by dates.

pivdates.xls

Chapter 27

This workbook is set up to demonstrate the shipping costs example using Solver.

shipping.xls

This workbook is set up to demonstrate the staff scheduling example using Solver.

schedule.xls

This workbook is set up to demonstrate the resource allocation example using Solver.

allocate.xls

This workbook is set up to demonstrate the investment portfolio example using Solver.

invest.xls

Chapter 32

An Excel version of tick-tack-toe.

tictac.xls

An Excel version of the common moving tile puzzle.

movetile.xls

An Excel version of Keno.

keno.xls

This workbook calculates the odds of winning in Keno.

kenoodds.xls

This workbook contains some animated Shape objects.

animshap.xls

Create colorful symmetrical patterns in Excel.

pattern.xls

This workbook displays a guitar fretboard and the notes in various scales and keys.

guitar.xls

This workbook contains a macro which reverses the text in Excel's menus.

menushen.xls

This workbook creates word search puzzles.

wordsrch.xls

This workbook contains examples of ASCII art.

asciart.xls

This workbook lets you play sound files (WAV or MID format).

scunder.xls

This workbook displays interesting charts that use trigonometric functions.

trigfun.xls

This workbook lets you draw simple figures that are actually X-Y charts.

xysketch.xls

Chapter 33

This workbook contains a custom toolbar to assist with formatting.

tcolbar.xls

Chapter 36

This workbook contains several examples of custom worksheet functions written in VBA.

funcs.xls

Chapter 37

This workbook contains a utility (with a custom dialog box) to make it easy to change the case of text in cells.

`chngcase.xls`

Chapter 38

This workbook contains examples of Excel's ActiveX controls.

`activex.xls`

Chapter 39

This is VBA macros that demonstrate how to copy a range of cells.

`rngcopy.xls`

This is VBA macros that demonstrate various ways to select a range of cells.

`select.xls`

This is VBA macros that demonstrate how to loop through a range of cells.

`lloop.xls`

This is VBA macros that demonstrate how to prompt for a value and insert the value into a cell.

`Prompt.xls`

This is VBA macros that demonstrate how to determine the type of object that is selected.

`seltype.xls`

This is VBA macros that demonstrate how to create a text box.

`textbox.xls`

This is VBA macros that demonstrate how to call attention to a particular cell.

`explode.xls`

This is VBA macros that work with chart objects.

`chartmacs.xls`

Chapter 40

This add-in contains a utility to make it easy to toggle various settings in Excel. This add-in is not protected, so you can view or modify the code.

toggles.xla

Bonus Files

The files contained in the Bonus directory aren't discussed in the book, but you may find them helpful. These files consists of Excel add-ins and standard Excel workbooks. On the CD-ROM, they appear in the following directories:

```
bonus\addins\  
bonus\workbooks\
```

Add-Ins

This section contains a list of the add-ins on the companion CD-ROM, with a brief description of each.



To install an add-in, first copy it to a directory on your local hard drive. Then, in Excel, select Tools • Add-Ins. In the Add-Ins dialog box, click Browse and locate the *.xla file that you want to install.



The Add-Ins dialog box lists all add-ins that Excel knows about. The add-ins that are checked will be loaded each time Excel starts. To reduce the startup time for Excel, remove the checkmark from any add-ins that you don't use.

daterept.xla

An add-in that generates a useful report that describes all date cells in a worksheet. This may help you identify potential Year 2000 problems.

faceid.xla

An add-in that makes it very easy for developers to determine the FaceID value for a CommandBar image. Useful if you develop custom menus in Excel.

dataform.xla

An add-in that provides an alternative to Excel's Data • Form command.

Workbooks

Below is a list of workbooks that follow a brief description of each.

This workbook demonstrates a technique that makes it very easy to create a custom menu for an Excel workbook or add-in. VBA programming not required!

menumakr.xls

This workbook demonstrates a technique to display help topics in Excel.

helpmakr.xls

This workbook contains an easy-to-use time sheet for tracking daily hours worked.

timesht.xls

This workbook lets you generate and print daily appointment calendar pages.

apptcal.xls

This macro generates all possible permutations of a string. Uses a recursive VBA subroutine.

permute.xls

Power Utility Pak

Power Utility Pak is a collection of Excel add-ins developed by the author of this book. The companion CD-ROM contains a copy of the shareware version of this product. The shareware version contains a subset of the features.



The CD-ROM contains PUP97. PUP97 works with both Excel 97 and Excel 2000. A significantly enhanced version, PUP2000, was being finalized as this book went to press. If you would like to try the shareware version of PUP2000, download a copy from:

<http://www.j-walk.com/ss/pup>

Registering Power Utility Pak

The normal registration fee for Power Utility Pak is \$39.95. However, you can use the coupon in this book to get the full version for only \$9.95.

Installing the shareware version

To install the shareware version of Power Utility Pak:

1. Make sure Excel is not running.

2. Locate the pup97r3.exe file on the CD-ROM. This file is located in the pup\ directory.
3. Double-click pup97r3.exe. This will expand the files to a directory you specify on your hard drive.
4. Start Excel.
5. Select Tools • Add-Ins and click the Browse button. Locate the power97.xla file in the directory you specified in Step 3.
6. Make sure Power Utility Pak 97 is checked in the add-ins list.
7. Click OK to close the Add-Ins dialog box.

The procedure described above will install Power Utility Pak, and it will be available whenever you start Excel. When the product is installed, you'll have a new menu: Utilities. Access the Power Utility Pak features from the Utilities menu.



Power utility Pak includes extensive on-line help. Select Utilities • Help to view the Help file.

To uninstall Power Utility Pak

If you decide that you don't want Power Utility Pak, follow these instructions to remove it:

1. In Excel, select Tools • Add-Ins
2. In the Add-Ins dialog box, remove the checkmark from Power Utility Pak 97.
3. Click OK to close the Add-Ins dialog box

After performing these steps, you can re-install Power Utility Pak at any time by placing a checkmark next to the Power Utility Pak 97 item in the Add-Ins dialog box.



To permanently remove Power Utility Pak from your system, delete the directory into which you originally installed it.

Sound-Proof

Sound-Proof is an Excel add-in, developed by the author of this book. Sound-Proof uses a synthesized voice to read the contents of selected cells. It's the perfect proof-reading tool for anyone who does data entry in Excel.

Cells are read back using natural language format. For example, 154.78 is read as "One hundred fifty-four point seven eight." Date values are read as actual dates (for example, "June fourteen, nineteen ninety-eight") and time values are read as actual times (for example, "Six forty-five AM").

The companion CD-ROM contains a demo version of Sound-Proof. The full version is available for \$19.95. Ordering instructions are provided in the online Help file.



The only limitation in the demo version is that it reads no more than 12 cells at a time.

Installing the demo version

To install the demo version of Sound-Proof:

1. Make sure Excel is not running.
2. Locate the sp.exe file on the CD-ROM. This file is located in the sp\ directory.
3. Double-click sp.exe. This will expand the files to a directory you specify on your hard drive.
4. Start Excel.
5. Select Tools • Add-Ins, and click the Browse button. Locate the soundprf.xla file in the directory you specified in Step 3.
6. Make sure Sound-Proof is checked in the add-ins list.
7. Click OK to close the Add-Ins dialog box.

The procedure described above will install Sound-Proof, and it will be available whenever you start Excel. When the product is installed, you'll have a new menu command: Tools • Sound-Proof. This command will display the Sound-Proof toolbar.

To uninstall Sound-Proof

If you decide that you don't want Sound-Proof, follow these instructions to remove it:

1. In Excel, select Tools • Add-Ins
2. In the Add-Ins dialog box, remove the checkmark from Sound-Proof.
3. Click OK to close the Add-Ins dialog box

After performing these steps, you can re-install Sound-Proof at any time by placing a checkmark next to the Sound-Proof item in the Add-Ins dialog box.



To permanently remove Sound-Proof from your system, delete the directory into which you originally installed it.

•

Index

A

- ABS function, 471
- absolute cell referencing, 178, 179–180, 181, 191
- absolute recording, 768–769
- accelerator (hot) key, 56, 65, 820. *See also* shortcut keys
- accounting format, 107
- ACCRINT function, 660
- ACCRINTM function, 660
- activating worksheets, 122
- active area, 264
 - identifying, 354
- active cell, 48
 - custom views of, 278
- active cell indicator, 26
- active objects, 774
- active window, 45
- active workbooks, 43
- ActiveX controls, 824. *See also* Control Toolbox toolbar
 - adding to worksheet, 825–826
 - linking to cells, 828–829
- add-ins, 13, 84, 625, 857–869. *See also* Analysis ToolPak; Solver
 - creating, 860–868
 - adding descriptive information, 867
 - example of, 862
 - protecting project, 867–868
 - reasons for, 860
 - setting up workbook, 862–865
 - testing workbook, 863–866
 - custom, 757
 - custom functions in, 791
 - defined, 857
 - included with Excel, 859
 - opening, 868–869
 - uses of, 857–858
- addition operator, 170, 172
- addresses of cells, 48, 139
 - alternate, 140
- AutoText method, 848
- advanced filtering, 522–528
 - criteria range, 523–524
 - criteria types, 525–528
 - multiple criteria, 525
 - other operations, 528
- alert messages, preventing, 852–853
- alignment, 112
 - cell, 242–246
 - horizontal, 242–244
 - text control options, 244–245
 - vertical, 244
 - of drawing objects, 331
- All references list box (Consolidate dialog box), 453
- Alt+Enter, 118
- AMORDEGRC function, 660
- AMORLINC function, 660
- ampersand (&), 270
- Analysis of Variance, 642–643
- Analysis ToolPak, 10, 474, 639–662, 859
 - functions, 232, 641, 657–662
 - Date & Time category, 657–658
 - Engineering category, 658–659
 - Financial category, 660–661
 - Information category, 661–662
 - Math & Trig category, 662
 - overview of, 639–640
 - tools in, 640–657
 - Analysis of Variance, 642–643
 - correlation, 643–644
 - Covariance, 644
 - Descriptive Statistics, 645–646
 - Exponential Smoothing, 646–647
 - Fourier Analysis, 648
 - F-Test (Two-Sample Test for Variance), 647–648
 - Histogram, 648–649
 - Moving Average, 650–651
 - Random Number Generation, 651–652
 - Rank and Percentile, 652–653
 - Regression, 653–655
 - Sampling, 655
 - t-Test, 656, 657
 - z-Test (Two-Sample Test for Means), 656–657
- analytical tools, 10
- AND function, 218
- AND operator, 524
- animated GIF files, 320
- animated menu, 737
- animated shapes, 720–721
- annotation, 10. *See also* comments
 - of cells, 147–148
 - of charts, AutoShapes for, 341
- annuity functions, 224–225
- application
 - destination, 500, 666
 - multiuser, 480
 - source, 500, 666
- Application object, 777
- April Fool's prank (game), 722–723
- area charts, 285, 286, 287, 309
- Areas method, 846
- argument(s), 201–203
 - errors in, 182
 - expressions as, 202–203
 - Formula Palette to specify, 205–206
 - literal, 202
 - names as, 202
 - other functions as, 203
 - of VBA functions, 792–796
 - no argument, 792
 - one argument, 793–794
 - range argument, 795–796
 - two arguments, 794–795
- array(s), 457–458
 - formatting, 464
 - looping with, 461–462
 - one-dimensional, 457
 - selecting current, 146
 - selecting range of, 464
 - two-dimensional, 457
- array constants, 464–467
- array feature, 10
- array formulas, 198, 457–478
 - advantages of, 459
 - for calendar calculation, 476–477
 - for computing maximum and minimum changes, 469–470
 - for counting characters in range, 468–469
 - defined, 458
 - disadvantages of, 459
 - in dynamic crosstab table, 475–476
 - editing, 463–464
 - entering, 463
 - external links stored in, 670
 - for frequency distributions, 473–474
 - for identifying value in range, 467–468
 - in one cell, 461
 - for ranking, 472–473
 - for returning last value in column, 476
 - for returning last value in row, 476
 - standard formulas vs., 459–460
 - for summing digits in a value, 470–471
 - for summing nth value in a range, 471–472
 - tips for, 477–478
- arrow keys, during data entry, 116
- ASCII art, 724, 725
- “at” sign (@), 201
- attributes, cell, 113
- Auditing toolbar, 703–704
 - Trace Error button of, 705

auditing tools, 354, 706–707
 for tracing cell relationships,
 703–705
 auditing, worksheet, 10
 Australia, map of, 408, 423
 AutoComplete, 117, 514, 710–711
 AutoCorrect, 11, 119–120, 174, 709–710
 formula, 700
 AutoFill, 117
 to copy adjacent cells, 153
 instead of formulas, 187
 AutoFiltering, 517–522
 charting list data, 522
 custom, 520–521
 limits of, 518
 multicolumn, 519–520
 Top 10, 521
 AutoFormat, 32, 33–34, 253–256
 controlling, 256
 using, 254–255
 Automatic Calculation mode, 183,
 184
 automatic menu customization, 55
 AutoSave, 84, 859
 autosensing, toolbar, 737
 AutoShapes, 327–330
 for annotating charts, 341
 around range, 849–850
 calling attention to cell, 336, 337
 to change look of cell comments,
 338–339
 changing defaults, 333
 examples, 336–342
 flow diagrams using, 340
 formatting objects, 328–330
 inserting text in, 335
 linked to cell, 339–340
 organizational charts using, 338
 shadow and 3D effects, 333–334,
 336–337
 shape categories, 327
 AutoSum button, 31, 58, 212
 AVERAGE function, 206, 229, 461–462
 axes on charts, 289, 365, 367, 372–376
 modifying, 372–373
 patterns of, 373–374
 scales of, 374–376
 secondary, 389–390

B

background, 252–253
 backing up files, 90
 backsolving. *See* single-cell goal seeking
 Backspace key, 101
 BAK files, 72, 83
 bar charts, 285, 304–305
 stacked, 399–400

Begins with operator, 549
 Bernoulli distribution, 652
 BESSELI function, 658
 BESSELJ function, 658
 BESSELK function, 658
 BESSELY function, 658
 bin range, 648
 BIN2DEC function, 658
 BIN2HEX function, 658
 BIN2OCT function, 658
 binders, data sharing using, 682
 binding constraints, 626
 binomial distribution, 652
 bitmap files, 499
 bitmap images, 321
 pasting, 672, 674
 black-and-white printing, 272
 blank cells, selecting, 146
 blanks, skipping when pasting, 160
 BMP files, 322
 Bomb Hunt, 718–719
 book.xlt file, 748, 749
 Boolean settings, 847
 borders, 113–114, 249–252
 skipping when pasting, 159
 brackets in formula bar, 463
 browse button (Consolidate dialog box),
 453
 browser, defined, 686
 bubble charts, 285, 312–313
 budget spreadsheets, 347
 button controls, 65
 Button Editor, 744–745
 button(s), toolbar
 adding/removing, 738–739
 applying styles with, 259
 assigning macros to, 770
 attaching macros to, 813–814,
 819–820
 AutoSum, 31, 58, 212
 Bold, 34
 changing functionality of, 740–741
 changing image of, 740, 744–746
 Close, 23, 25
 on Control Toolbox Toolbar, 825
 Copy, 58
 copying ranges using, 149
 custom, 756
 Maximize, 25
 Minimize, 23, 24
 New Workbook, 74
 number-formatting, 103
 Open, 76
 option (radio), 65–66
 other operations on, 740–741
 Print, 263
 Restore, 23
 tab-scrolling, 122

C

calculated field, 582–583
 calculated items, 583–585
 Calculation modes, 183–184, 847–848
 calculator, Formula Bar as, 196–197
 calendar calculation, array formulas for,
 476–477
 Canada, map of, 408, 423
 case sensitive sort order, 534
 Case statement, 780
 category axis, 289
 category axis title, 369
 category, consolidating worksheets by,
 452
 category fields, 562
 category shading map format, 412–413
 CDR files, 322
 cell(s), 139–168
 active, 48
 addresses of, 48, 139
 alternate, 140
 alignment of, 242–246
 horizontal, 242–244
 text control options, 244–245
 vertical, 244
 annotating, 147–148
 array formulas in one, 461
 AutoShapes linked to, 339–340
 changing, 607, 614–615, 623
 color coding of, 703
 comments in, 10, 147–148
 AutoShapes to change look of,
 338–339
 pasting, 159
 copying ranges to adjacent, 153
 defined, 3, 139
 deleting contents of, 148
 drawing object to call attention to,
 336, 337
 editing contents of, 99–102
 erasing contents of, 99, 100
 forcing new line in, 118
 hiding before printing, 280–281
 linking dialog box controls to,
 828–829
 moving, 154–155
 multiple formatting worksheets in
 one, 241–242
 names/naming of, 161–168
 advantages of, 161–162
 automatic, 164–166
 changing, 168
 deleting, 167
 manual, 162–164
 redefining, 168
 table of names, 166–167
 valid, 162
 pasting pictures of, 341–342

- preformatting, 108, 353
- protection of, 13
- replacing contents of, 99
- selecting, during data entry, 116
- stylistic formatting of, 112–114
 - alignment, 112
 - attributes, 113
 - borders, 113–114
 - color, 114
 - font and text size, 113
 - pasting, 159
- target, 623
- tracing relationships of, 700–706
 - auditing tools for, 703–705
 - circular references, 706
 - Go To Special dialog box, 701–703
 - tracing error values, 705–706
- cell dependents, 701, 702–704
- CELL function, 219
- cell orientation, 245–246
- cell pointer, 115
- cell precedents, 701, 702–704
- cell references
 - absolute, 178, 179–180, 181, 191
 - applying names to existing, 194–196
 - as arguments, 202
 - circular, 184–188, 706
 - entering formulas by pointing to, 175
 - in formulas, 169
 - invalid, 182
 - mixed, 180–181
 - nonrelative, 181
 - relative, 178–179, 180, 191
 - outside worksheet, 177–178
- cell selection, *see-through*, 17
- cell tracers, 704
- centering text, 112
- change history, 484
- changing cells, 607, 614–615, 623
- CHAR function, 214
- chart(s), 283–316, 363–404
 - activating, 299, 363
 - automatically updating, 402–403
 - AutoShapes for annotating, 341
 - axes on, 289, 365, 367, 372–376
 - modifying, 372–373
 - patterns of, 373–374
 - scales of, 374–376
 - secondary, 389–390
 - changing location, 300
 - chart area, 366, 367–368
 - combination, 388
 - comparative histograms, 401–402
 - creating, 35–37, 292–299
 - chart options, 298
 - chart placement, 298–299
 - chart type selection, 296–297
 - with Chart Wizard, 293–299
 - data selection, 295–296
 - with one keystroke, 288–289
 - range verification and data orientation, 297–298
 - customizing, 363–364
 - data labels in, 365–366, 382–383
 - data series for, 377–388
 - adding, 378–379
 - changing, 379–382
 - controlling, by hiding data, 384–385
 - deleting, 378
 - data table, 367, 390, 391
 - elements of, 364–365
 - background, 367–368
 - deleting, 301
 - moving, 301
 - embedded, 284, 291, 293, 302, 363, 681
 - error bars in, 367, 385–386
 - Excel handling of, 289–290
 - Format dialog box, 365–367
 - formatting of, 851–852
 - fun with, 725–727
 - gridlines on, 364, 367, 371–372
 - handling missing data, 383–384
 - legends, 367, 370–371
 - linked, 587
 - modifying, 299–301
 - moving, 299
 - as object, 289
 - organizational, 338, 681
 - overview, 283–285
 - picture charts, 395–399
 - Clipboard to create, 397–399
 - graphic file to create, 397
 - from pivot tables, 587–588
 - placement of, 283–284
 - plot area, 367, 368–369
 - plotting trigonometric functions, 725–726
 - printing, 302–303
 - properties of, 851
 - resizing, 299
 - selecting, 286–287, 301
 - 3D, 290, 365, 382–394
 - modifying, 392
 - rotating, 392–394
 - from two-input data tables, 606–607
 - titles, 365, 369–370
 - trendlines in, 367, 386–388
 - tricks for making, 394–399
 - changing worksheet value by dragging, 394–395
 - unlinking chart from data range, 395, 396
 - types of, 285–286, 850–851
 - area charts, 285, 286, 287, 309
 - bar charts, 285, 304–305, 399–400
 - bubble charts, 285, 312–313
 - changing, 300–301
 - changing default, 301–302
 - column charts, 285, 286, 287, 289, 303–304, 396
 - cone charts, 285, 314–315
 - custom, 390–392
 - cylinder charts, 285, 314–315
 - doughnut charts, 285, 310
 - Gantt, 399–401
 - line chart, 285, 306
 - pie charts, 285, 287, 289, 307–308
 - pyramid charts, 285, 314–315
 - radar charts, 285, 311
 - stock charts, 285, 313–314
 - surface charts, 285, 311–312
 - XY (scatter) charts, 285, 308–309
 - VBA programming of, 850–852
 - from worksheet outlines, 440
- Chart Data Labeler utility, 393
- chart sheets, 284, 292, 294–295
- Chart toolbar, 293, 294
 - Chart Objects tool in, 365
- Chart Type tool, 293
- Chart Wizard, 36, 37, 287–288, 293–299
- charting list data, 522
- ChartObject objects, 851
- check box(es), 66
 - Fixed Decimal, 116
 - Precision as Displayed, 106
- Check Mark icon, 100
- CheckBox control, 831
- Circular Reference toolbar, 185, 186
- circular references, 184–188, 706
 - indirect, 186
 - intentional, 186–188
- client-server model, 479
- Clip Gallery, 318–320
- Clipboard
 - creating picture charts using, 397–399
 - data sharing using, 666–668
 - importing data through, 497–502
 - Office, 18–19, 150–151, 497
 - clearing contents of, 157
 - Office 2000, 155–158
 - Windows, 150–151, 158, 497
- Clipboard toolbar, 19, 60
- Clipboard Viewer, 150, 497, 498, 500
- Clips Online, 319
- closing windows, 47
- closing workbooks, 87–88

- code(s)
 - for CELL function, 219
 - for custom formatting values, 109–111
 - defined, 759
 - formatting, 109–111
 - for INFO function, 220
 - for WEEKDAY function, 222
- CODE function, 214
- coding. *See* Visual Basic for Applications (VBA); VBA programming
- coefficient, 622
 - of correlation, 643
- collect and paste, 18–19
- collection
 - defined, 773
 - VBA, 775–776
- color, 114, 246–249, 251
 - care in using, 353
 - for value-shading format, 412
- color coding of cells, 703
- color palette, 251
- column(s), 48, 133–137
 - array formulas for returning last value in, 476
 - changing widths of, 135–136
 - deleting, 135
 - hiding, 136–137
 - inserting, 133–135
 - in lists, 513
 - preformatting, 513
 - printing headings, 272
 - selecting complete, 141–142
 - selecting, for querying, 545
 - sorting on two or more, 530
 - widths of
 - adjusting, in print preview window, 275
 - pasting, 159
- column charts, 285, 286, 287, 289, 303–304, 396
- column differences, selecting cells with, 146
- column field, 562
- column headings, 28–30, 119, 131
 - using as names, 195
 - on workbook window, 26
- column-chart maps, 416, 417
- combination chart, 388
- ComboBox control, 831–832
- Comma Style button, 103
- command(s). *See also* shortcut keys
 - custom, 756
 - giving, 51–64
 - reversing, 53
 - using menus, 51–57
 - using shortcut keys, 64
 - using shortcut menus, 57–58
 - using toolbars, 58–63
- help for, 58
- menu
 - for annotating cells, 147–148
 - for applying names to cell references, 195
 - for cell or range operations, 154–155
 - for changing column widths, 136
 - for changing row heights, 136
 - for charts, 300, 302
 - for consolidating worksheets, 451–454
 - for copying ranges, 151, 153
 - for creating new workbook, 74
 - for creating picture of cell or range, 341
 - custom views, 278
 - for data tables, 390, 599
 - for deleting cell contents, 99
 - for embedding objects, 680
 - for freezing/unfreezing panes, 132
 - Goal Seek, 620
 - for hiding/unhiding rows and columns, 136–137
 - for hiding/unhiding worksheets, 126
 - hyperlink, 693
 - for importing from camera or scanner, 324
 - for importing graphics, 320
 - for inserting cells, 135
 - for inserting rows and columns, 134
 - for linking, 670
 - macro, 761
 - for mailing, 487
 - for making selections, 137, 145
 - for names, 165, 166, 167, 168
 - for opening existing workbook, 75
 - for outlines, 439
 - for page breaks, 276
 - for pasting, 158, 176
 - for queries, 554
 - for saving workbooks, 80
 - for saving workspace, 88
 - for splitting panes, 130
 - for styles, 259–260
 - for zooming, 127
 - What's This?, 58
- CommandBar. *See* toolbar(s)
- CommandButton control, 832–833
 - inserting, 829–830
- comma-separated text files, 496
- comma-separated value text file format, 73
- comments
 - in cells, 10, 147–148
 - AutoShapes to change look of, 338–339
 - pasting, 159
 - printing, 273
 - selecting cells with, 146
 - in VBA programs, 758
- commissions, calculating, 793–794
- comparative histograms, 401–402
- comparison operators, 526
- compatibility, file, 7
- COMPLEX function, 658
- computed criteria, 525–528
- concatenation, 171
- concatenation operator, 170, 172, 806
- Conditional Formatting, 109
 - selecting cells with, 147
- conditional formatting worksheets, 256–257
- Conditional Sum Wizard, 859
- cone charts, 285, 314–315
- conflicts, multiple-user, 484
- consistency, cross-platform, 14
- consolidating worksheets, 448–455
 - data sources for, 454–455
 - linking worksheets and, 442
 - pivot tables for, 585–587
 - shared workbooks for, 481
 - by using Data ⇄ Consolidate, 452–454
 - by using formulas, 449–451
 - by using Paste Special, 451
- constant(s)
 - array, 464–467
 - in MsgBox function, 804–805
 - naming, 190–191
 - selecting cells with, 146
 - smoothing (damping factor), 646
- constraints, 623, 625, 633–635
 - binding, 626
- contains operator, 549
- context menus. *See* shortcut menus (context menus)
- control(s), 12, 13
 - ActiveX. *See* ActiveX controls; Control Toolbox toolbar
 - buttons, 65
 - check boxes, 66
 - in Controls Toolbox, 831–838
 - CheckBox, 831
 - ComboBox, 831–832
 - CommandButton, 832–833
 - Image, 833
 - Label, 833
 - ListBox, 833–834

- OptionButton, 834–835
 - ScrollBar, 835–836
 - SpinButton, 836–837
 - TextBox, 837–838
 - ToggleButton, 838
 - custom dialog box, 807–810
 - properties of, 809–810
 - defined, 759
 - dialog box, 64, 65–68, 823–838
 - adding, 826
 - available, 824–825
 - design mode and, 826
 - linking to cells, 828–829
 - macros for, 829–830
 - properties, 826–828
 - reasons for using, 823–824
 - drop-down boxes, 68
 - list-boxes, 67
 - option buttons, 65–66
 - range selection boxes, 66, 67
 - spinners, 66–67
 - Control Toolbox toolbar, 824
 - CONVERT function, 658
 - copy tool, 157
 - copy-and-paste technique, 668. *See also*
 - Clipboard
 - copying
 - buttons, 738
 - formulas, 197
 - graphics images, 667
 - queries, 553
 - ranges, 149–154, 840–841
 - to adjacent cells, 153
 - to other sheets, 154
 - using drag and drop, 152–153
 - using menu commands, 151, 153
 - using shortcut keys, 152
 - using shortcut menus, 151
 - using toolbar buttons, 149
 - rows, 528
 - worksheets, 125
 - corners of 3D charts, 366
 - CORREL function, 8
 - Correlation tool, 643–644
 - corrupted files, 90
 - links to recover data from, 449
 - COUNT function, 230, 569
 - COUNTA function, 230, 403
 - COUNTBLANK function, 230
 - COUNTIF function, 229–230, 473–474
 - COUPDAYBS function, 660
 - COUPDAYS function, 660
 - COUPDAYSNC function, 660
 - COUPNCD function, 660
 - COUPNUM function, 660
 - COUPPCD function, 660
 - COVAR function, 644
 - Covariance tool, 644
 - Create links to source data check box
 - (Consolidate dialog box), 453
 - creating
 - charts, 35–37, 292–299
 - chart options, 298
 - chart placement, 298–299
 - chart type selection, 296–297
 - with Chart Wizard, 293–299
 - data selection, 295–296
 - with one keystroke, 288–289
 - picture charts, 395–399
 - range verification and data orientation, 297–298
 - formulas, 31
 - maps, 409–417
 - category shading, 412–413
 - column-chart maps, 416, 417
 - combined formats, 416
 - data setup, 409
 - dot density, 413–414
 - formats, 410–411
 - graduated symbol, 414–415, 417
 - pie-chart maps, 415–416
 - value shading, 411–412, 417
 - named styles, 260
 - new workbooks, 74–75
 - spreadsheets, 349–358
 - considering audience, 350–351
 - designing workbook layout, 351–352
 - developing a plan, 350
 - entering data and formulas, 352–353
 - formatting, 353–354
 - protection, 355–357
 - testing, 354
 - Criteria pane, 554, 555
 - criteria range, 523–524
 - criteria, validation, 159
 - cropping objects, 678
 - Cropping tool, 678
 - cross-platform consistency, 14
 - Ctrl+left/right arrow, 101
 - cubes, OLAP, 543, 566
 - CUMIPMT function, 660
 - CUMPRINC function, 660
 - currency, Euro symbols of, 19
 - currency format, 107
 - Currency Style button, 103
 - current region, selecting cells in, 146
 - custom error bar, 386
 - custom filtering, 520–521
 - custom functions, 234. *See also* VBA
 - functions
 - custom header/footer buttons, 270
 - custom number formats, 107, 108–112
 - to hide cells, 281
 - Custom Pin Map tool, 420
 - custom views, 278–279
 - custom workbook templates, 747, 750–754
 - customer geographic analysis, using
 - pivot tables for, 591–592
 - customer lists, 481
 - customizable toolbars, 8
 - customization mode, 734
 - customizing, 731–746
 - fields in pivot tables, 575–576
 - headers/footers, 269–270
 - maps, 417–418
 - menus, 17–18
 - automatic, 55
 - pivot table field, 575–576
 - shortcut menus, 733
 - toolbars, 18, 731–734
 - adding/removing buttons, 738–739
 - changing button functionality, 740–741
 - changing button image, 740, 744–746
 - creating new toolbars, 735, 741–744
 - Customize dialog box for, 734–737
 - moving toolbars, 733–734
 - types of, 733
 - Cut method, 843
 - cylinder charts, 285, 314–315
- ## D
- damping factor (smoothing constant), 646
 - data
 - appropriate for pivot tables, 562–564
 - controlling data series by hiding, 384–385
 - defined, 492
 - dummy, 354
 - entering. *See* data entry
 - formatting, 556
 - handling missing chart, 383–384
 - importing. *See* Importing data
 - "noisy," 387
 - sharing with other Windows
 - applications, 665–683
 - by linking data, 669–670
 - Microsoft Word, 671–674
 - using binders, 682
 - using Clipboards, 666–668
 - using OLE (object linking and embedding), 674–681
 - sorting, 556
 - source, 563

- data analysis. *See* Analysis ToolPak;
 - pivot tables; single-cell goal seeking; Solver; what-if analyses
- data area, 562
- data entry, 30, 114–120
 - arrow keys during, 116
 - AutoComplete feature, 117
 - AutoCorrect feature, 119–120
 - AutoFill feature, 117
 - automatic decimal points during, 116
 - cell pointer movement during, 115
 - current date or time, 117
 - forcing new line in cell, 118
 - forms for, 118–119
 - fractions, 118
 - into lists, 514–517
 - with Data Form dialog box, 515
 - Microsoft Access forms for, 516–517
 - repeating information, 116
 - selecting cells before, 116
 - into spreadsheets, 352–353
 - validating, 114–115
 - selecting cells set up for, 147
- data fields, 562
- data files, 72
- Data Interchange Format (DIF), 73, 496
- data labels in charts, 365–366, 382–383
- Data pane, 555
- data range
 - automatically updated, 402
 - unlinking chart from, 395, 396
- data series, 287, 377–388
 - adding, 378–379
 - changing, 379–382
 - Data Source dialog box for, 380
 - dragging range outline, 379–380
 - editing SERIES formula, 380–381
 - using names in SERIES formula, 381–382
 - deleting, 378
 - points in, 367
 - selecting, 377
- data tables, 367, 390, 391, 599–607
 - limitations of, 607
 - one-input, 600–603
 - two-input, 603–607
- data types, 93–95
 - formulas, 95
 - text, 94
 - values, 93–94, 95
 - VBA, 854
- data-analysis models, 346–348
- database(s). *See also* external data files;
 - file formats supported, 493, 495
 - OLAP, 543, 566
 - calculated fields and, 582
 - calculated items and, 584
 - relational, 542, 556
 - terminology of, 542
- database access, spreadsheets for, 348
- database functions, 231–232, 528–529.
 - See also specific names of functions*
 - with lists, 528–529
- database management, 11
- date(s), 97–98
 - custom formatting codes for, 110–111
 - entering current, 117
 - grouping by, 593
 - in pivot tables, grouping by, 593
 - sorting and, 535
- date format, 107
- DATE function, 222
- date functions, 221–222. *See also specific names of functions*
- date serial number system, 98
- Date & Time category functions, 657–658
- DAVERAGE function, 529
- DAY function, 222
- days, AutoFill to create series of, 187
- DB function, 224
- dBase file format, 73
- DBF files, 495, 539
- DCOUNT function, 529
- DCOUNTA function, 529
- DDB function, 224
- debugging
 - of formulas. *See* troubleshooting of VBA functions, 796
- DEC2BIN function, 659
- DEC2HEX function, 659
- DEC2OCT function, 659
- decimal points, automatic insertion
 - during data entry, 116
- Decision Support Services (DSS)
 - analysis, 543
- Decrease Decimal button, 103
- default(s)
 - AutoShapes, changing, 333
 - file formats, 86
 - printing
 - settings, 263–264
 - templates to change, 281
 - workbook location, 82
- default templates, 748, 749–750
 - workbook, 747, 748–749
 - changing, 748–749
 - resetting, 749
 - worksheet, 747, 749–750
- Delete key, 99, 101
- deleting
 - cell contents, 148
 - elements of charts, 301
 - named styles, 261
 - names, 167
 - queries, 553
 - rows and columns, 135
 - toolbars, 735
 - worksheets, 123
- delimited text files, 496, 503
- DELTA function, 659
- dependent workbook, 441
- dependents, cell, 701, 702–704
 - selecting cells with, 146
- depreciation functions, 223
- Descriptive Statistics tool, 645–646
- design mode, 826
- design time, 810
- destination application, 500, 666
- DGET function, 529
- dialog box(es), 64–69
 - Add Constraint, 625
 - integer option of, 634
 - Add Scenario, 609–610
 - Add Trendline, 387
 - Options tab of, 387, 388
 - Type tab of, 387
 - Add View, 279
 - Add-Ins, 858, 861, 869
 - Advanced Filter, 524, 528
 - Copy to Another Location option, 528
 - Unique records option, 528
 - Advanced Text Import Settings, 507
 - Anova: Single Factor, 642
 - Apply Names, 195–196
 - Arrange Windows, 45
 - Assign Hyperlink: Open, 746
 - Assign Macro, 814
 - AutoCorrect, 709–710
 - AutoSave, 84
 - Button Editor, 744, 745
 - Chart Area, 368
 - Font tab of, 368
 - Patterns tab of, 368
 - Properties tab of, 368
 - Chart Options
 - Data Labels tab of, 382, 383
 - Gridlines tab, 372
 - Chart Type, 300
 - Custom Types tab of, 390, 391
 - Chart Wizard, 36, 287–288, 297–299
 - ClipArt, 318
 - Consolidate, 452–453
 - controls, 64, 65–66, 823–838
 - adding, 826
 - available, 824–825
 - CheckBox, 831
 - ComboBox, 831–832
 - CommandButton, 832–833
 - design mode and, 826

- Image, 833
- Label, 833
- linking to cells, 828–829
- ListBox, 833–834
- macros for, 829–830
- OptionButton, 834–835
- properties, 826–828
- reasons for using, 823–824
- ScrollBar, 835–836
- SpinButton, 836–837
- TextBox, 837–838
- ToggleButton, 838
- Correlation, 643
- Create Names, 165
- Create New Data Source, 544
- custom, 12, 801–821
 - adding accelerator keys to, 820
 - alternatives to, 802–806
 - attaching macro to button, 813–814, 819–820
 - controls, 807–810
 - displaying, 810
 - event handling by, 810, 812–813, 817–819
 - examples of, 810–820
 - learning more about, 821
 - reasons for creating, 801–802
 - tab order in, 821
 - testing, 812, 816–817, 819
 - UserForms for, 807
- Custom AutoFilter, 520–521
- Customize, 61–62, 734–737
 - Commands tab of, 736, 738
 - Options tab of, 736–737
 - Toolbars tab of, 735–736
- Data Analysis, 641
- Data Form, 515–516
 - buttons on, 516
- Data Source, 380
- Data Validation, 115
- Define Name, 162–163, 167, 189–190, 191, 464, 465
- Descriptive Statistics, 645
- Edit Button, 744, 745
- External Data Range Properties, 551–552, 697
- file, 16
- File in Use, 480
- File Not Found, 445
- File Now Available, 481
- Find, 79
- Format, 365–367
 - Alignment tab of, 370
 - Font tab of, 370
 - to modify chart title properties, 370
 - Patterns tab of, 370
- Format AutoShape, 328–330
 - Alignment tab of, 330
 - Colors and Lines tab of, 328–329
 - Font tab of, 330
 - Margins tab of, 330
 - Properties tab of, 330
 - Protection tab of, 329–330
 - Size tab of, 329
 - Web tab of, 330
- Format Axis
 - Alignment tab of, 373
 - Font tab of, 373
 - Number tab of, 373
 - Patterns tab of, 372, 373–374
 - Scale tab of, 372, 375–376
- Format Cells, 68, 104–108, 237
 - Number tab of, 105–106
- Format Data Labels, 383
- Format Data Series, 377
 - Axis tab of, 377, 390
 - Data Labels tab of, 377
 - Options tab of, 377
 - Patterns tab of, 377, 397
 - Series Order tab of, 377
 - Shape tab, 377
 - X Error Bars tab of, 377
 - Y Error Bars tab of, 377, 385
- Format Properties
 - Dot Density Options tab of, 414
 - Legend Options tab of, 421
 - Pie Chart Options tab of, 416
 - Value Shading Options tab of, 411–412
- F-Test, 647
- Function Wizard, 641
- Go To Special, 145–146, 701–703
- Goal Seek, 395, 619, 622
- Graduated Symbol Options, 414
- Grouping, 580
- Histogram, 648–649
- Insert, 134
- Insert Calculated Field, 583
- Insert Hyperlink, 693
- Insert Picture, 320
- invoked through menus, 54
- Links, 445, 670
- Macro, 797
- Map Features, 422
- Map Labels, 419
- Merge Scenarios, 612
- Microsoft Map Control, 410
- Moving Average, 650
- Multiple Maps Available, 409
- navigating using keyboard, 65
- New, 74, 75
- New Web, 695
- Object, 679
- Open, 75–78, 493
- Options, 133, 184
 - Chart tab of, 384–385
 - Custom Lists tab of, 535
- Page Setup, 264, 266–273
 - Chart tab of, 303
 - Header/Footer tab of, 269–271
 - margin adjustments, 268
 - Margins tab of, 268
 - page settings, 267
 - Page tab of, 267
 - printer-specific options, 273
 - sheet options, 271–273
 - Sheet tab of, 271
- Paste, 797
- Paste Function, 205, 208, 232, 798
- Paste Name, 167, 176, 191
- Paste Special, 155, 158, 379, 451, 671–672, 673–674
- PivotTable and PivotChart Wizard, 565–569
- PivotTable Field, 576
- PivotTable Options, 572–573
 - Enable drilldown option, 581
- Print, 264–266
 - Options button, 273
- Project Properties, 867–868
- Properties, 85, 264, 357
 - Summary tab of, 357
- Protect Sheet, 355
- Query Wizard, 545–549
- Random Number Generation, 651–652
- Record Macro, 761
- Regression, 654
- Resolve Unknown Geographic Data, 425
- Sampling, 655
- Save As, 37, 38, 81, 82–83, 86, 690
- Save Options, 83
- Scenario Manager, 609
- Scenario Summary, 612–613
- Scenario Values, 610
- Settings, 438
- Shared Workbook, 482, 483
 - Advanced tab of, 483–484
- Show Pages, 581–582
- Solver Options, 627–628
- Solver Parameters, 624
 - Set Target Cell field of, 625
- Solver Results, 626
- Sort, 532–533
- Sort Options, 533–534
- Source Data, 378–379
- Specify Geographic Data, 424
- Spelling, 708
- Style, 260, 261
- Subtotal, 537
- tabbed, 68–69

continued

- dialog box(es), *(continued)*
 - Table, 602
 - Text Import Wizard, 504-508
 - t-Test: Paired Two Sample for Means, 656
 - UserForm, 20
 - Web Options, 691
 - WordArt Gallery, 334
 - Word's Object, 677
 - Worksheet Auditing, 706
 - Zoom, 127
 - DIB files, 322, 499
 - Dice Game, 718, 719
 - DIF (Data Interchange Format), 73, 496
 - digital camera, importing graphics from, 324-325
 - dimensions, OLAP, 543, 566
 - direct precedent, 701
 - DISC function, 666
 - discrete distribution, 652
 - display settings, custom views of, 278
 - distributions, random number, 652
 - #DIV/0!, 706
 - dividends, 637
 - division by zero, 182
 - division operator, 170, 172
 - DMAX function, 529
 - DMIN function, 529
 - documenting work, 356-357
 - documents, on Windows Taskbar, 18
 - does not begin with operator, 549
 - does not contain operator, 549
 - does not end with operator, 549
 - does not equal operator, 549
 - DOLLARDE function, 660
 - DOLLARFR function, 660
 - DOS window, copying contents into
 - Clipboard, 501-502
 - dot density map format, 413-414
 - doughnut charts, 285, 310
 - down-bars, 367
 - download, defined, 686
 - DPRODUCT function, 529
 - draft quality printing, 272
 - drag and drop, copying ranges using, 152-153
 - drag and drop fields, 570
 - dragging, changing worksheet value by, 394-395
 - draw layer, 93, 323, 325
 - drawing, symmetrical pattern, 721
 - drawing tips, 335-336
 - Drawing toolbar, 325-327
 - drawing tools, 325-334
 - AutoShapes, 327-330
 - for annotating charts, 341
 - calling attention to cell, 336, 337
 - to change look of cell comments, 338-339
 - changing defaults, 333
 - examples, 336-342
 - flow diagrams using, 340
 - formatting objects, 328-330
 - inserting text in, 335
 - linked to cell, 339-340
 - organizational charts using, 338
 - shadow and 3D effects, 333-334, 336-337
 - shape categories, 327
 - objects
 - aligning, 332
 - changing stack order of, 330-331
 - grouping, 331
 - spacing evenly, 332
 - pasting pictures of cells, 341-342
 - WordArt, 334-336
 - drawing tips, 335-336
 - example of, 335
 - drop-down boxes, 68
 - drop-down lists, 518
 - droplines, 367
 - DRW files, 322
 - DSS analysis, 543
 - DSTDEV function, 529
 - DSTDEVP function, 529
 - DSUM function, 231, 529
 - dumb terminals, 479
 - dummy data, 354
 - DURATION function, 660
 - DVAR function, 529
 - DVARP function, 529
 - dynamic crosstab table, array formulas
 - in, 475-476
- ## E
- EDATE function, 658
 - Edit menu, 52
 - editing
 - array formulas, 463-464
 - cell contents, 99-102
 - formulas, 182-183
 - selecting characters during, 183
 - functions, 200
 - macros, 765
 - records, 556
 - SERIES formula, 380-381
 - using Data Form dialog box, 516
 - EFFECT function, 660
 - ellipsis, menu items ending with, 64
 - e-mail attachments, workbook mailed
 - as, 485-486
 - e-mail, defined, 686
 - embedded charts, 284, 291, 293, 302, 363
 - organizational charts, 681
 - embedding objects. *See* object linking and embedding (OLE)
 - EMF files, 322
 - End key, 101
 - End method, 842
 - End Sub keyword, 757
 - ends with operator, 549
 - Engineering category functions, 658-659
 - Enter key, 101
 - EOMONTH function, 658
 - EOS files, 322
 - equal to operator, 170, 172, 526
 - equals operator, 549
 - erasing cell contents, 99, 100. *See also* deleting
 - ERF function, 659
 - ERFC function, 659
 - error(s). *See also* troubleshooting returned by formulas, 181-182
 - in spreadsheets, 358-359
 - syntax, 772
 - error bars, in chart, 367, 385-386
 - error values, tracing, 705-706
 - Euro currency symbols, 19
 - Europe, map of, 408, 423
 - event handling, 810, 812-813, 817-819
 - Excel 2, 4
 - Excel 3, 4
 - Excel 4, 4
 - Excel 5, 4, 86
 - Excel 7 (Excel for Windows 95), 4-5
 - Excel 8 (Excel 97), 5
 - Excel 9. *See* Excel 2000 (highlighted features)
 - Excel 97, 318
 - Excel 2000 (highlighted features), 4-5, 15-20
 - active windows in, 46
 - adding/removing buttons in, 739
 - animated GIF files in, 320
 - Assign Hyperlink feature in, 741
 - automatic formula adjustment in, 17
 - AutoShapes, 327
 - changeable range references in, 204
 - Clip Gallery, 318
 - Clipboard operations in, 666, 667
 - Clipboard toolbar in, 50
 - Clips Online, 319
 - default file formats in, 86
 - documents on Windows taskbar in, 18
 - enhanced Clipboard in, 18-19
 - Euro currency symbols in, 19
 - file dialog boxes in, 16
 - finding lost workbooks in, 79
 - fonts in, 113
 - header/footer limitations in, 271
 - help system in, 19

- HTML features in, 73, 348, 690
 - image import options, 19
 - importing images in, 324
 - Insert Clip Art and Line Color in, 327
 - installation improvements in, 15
 - Internet features of, 16
 - macro recorder bug fix in, 769, 842
 - menus in, 55
 - modeless UserForms, 20
 - multilingual features of, 20
 - native file format of, 492
 - Office Assistant in, 19–20
 - Open dialog box in, 75, 76
 - parts of, 21–27
 - close button, 23
 - formula bar, 23
 - menu bar, 23
 - minimize button, 23
 - name box, 23
 - restore button, 23
 - status bar, 22, 24
 - title bar, 22
 - toolbars, 23
 - window control menu button, 23
 - of workbook window, 24–27
 - personalized menus in, 17–18
 - Pivot chart reports in, 588
 - PivotTable enhancement, 17
 - PivotTable toolbar in, 570
 - refreshing queries in, 552
 - see-through cell selection in, 17
 - "See-through View" in, 141
 - Standard and Formatting toolbars in, 59
 - statistical functions in, 231
 - templates in, 89
 - toolbar customization in, 18
 - VBA in, 758
 - Web page activation from toolbar button, 746
 - Web tab feature, 330
- Excel 2000 Power Programming with VBA*, 771, 784
- Excel (all versions)**, 3–14
- competitors of, 5
 - defined, 3
 - evolution of, 4–5
 - features of, 5–14
 - add-in capability, 13
 - analytical tools, 10
 - built-in functions, 8
 - charts, 9
 - cross-platform consistency, 14
 - customizable toolbars, 8
 - database management, 11
 - dialog boxes, 12
 - drawing tools, 9
 - easy-to-use, 7
 - file compatibility, 7
 - integrated mapping, 9
 - interactive help, 7
 - Internet support, 14
 - list management, 7
 - multiple document interface, 6
 - multisheet files (workbooks), 6
 - OLE support, 13
 - pivot tables, 10
 - printing and print preview, 10
 - protection options, 13
 - scenario management, 11
 - spell checking and AutoCorrect, 11
 - templates, 11, 12
 - text formatting, 9
 - text handling, 8
 - Visual Basic for Applications (VBA), 12
 - worksheet auditing and annotation, 10
 - worksheet controls, 12, 13
 - worksheet outlining, 10
 - XLM macro compatibility, 11
 - file compatibility among versions of, 86
 - quitting, 38
 - starting, 21
- Expense State template, 89
- Exponential Smoothing tool, 646–647
- exponentiation operator, 170, 172
- expressions, as arguments, 202–203
- external data files, 539–557
- adding and editing records in, 556
 - for pivot table, 565
 - queries on, 540–550
 - changing, 553–554
 - copying or moving, 553
 - creating, 554–555
 - deleting, 553
 - external data ranges, 551–554
 - to get data, 541
 - multiple, 553
 - operators for, 549
 - without Query Wizard, 554–557
 - refreshing, 552–553
 - selecting data source, 542–545
 - starting, 542
 - using multiple database tables, 556
 - using Query Wizard, 545–550
 - reasons for using, 539–540
- external reference formulas, 442–446
- for consolidating worksheets, 449–451
 - creating by pasting, 443
 - creating by pointing, 443
 - data recovery using, 449
 - opening workbook with, 444–445
- potential problems with, 447–448
- syntax for, 442–443
- F**
- FACTDOUBLE function, 659
- features in Excel 2000, 16
- HTML as native file format, 690–691
 - hyperlinks, 693–694
 - information available on, 687
 - interactivity with Web documents, 691–692
 - mailing lists, 689
 - newsgroups, 688
 - terminology of, 686
 - Web queries, 694–697
 - Web toolbar, 692–693
- field(s)
- adding, 575
 - calculated, 582–583
 - category, 562
 - column, 562
 - data, 562
 - defined, 542
 - drag and drop, 570
 - page, 561, 562, 571
 - removing, 575
 - row, 563, 574
- field buttons, 574
- file(s), 71–90. *See also* external data files; table(s); workbook(s)
- add-in, 84
 - backing up, 90
 - corrupted, 90, 449
 - data, 72
 - defined, 71, 492
 - display preferences, 77–78
 - manipulation of, 71
 - multisheet. *See* workbook(s)
 - naming rules for, 82
 - properties of, 71–72
 - read-only, 480
 - template. *See* template(s)
 - workspace, 88
- file compatibility, 7
- file dialog boxes, 16
- file formats, 73, 492–496
- database, 493, 495
 - HTML, 16, 73, 493, 496, 686, 690–691
 - Lotus 1-2-3, 493, 494–495
 - other, 493
 - Quattro Pro, 73, 493, 495
 - text, 493, 495–496
- file reservation, 479–481
- File ⇨ Save As command, 447
- file security, 83
- file servers, 16, 479, 480
- backup copies on, 90

- File Transfer Protocol (FTP), 686
- Fill Color tool, 301
- fill handle, 29, 148
- filtering
 - by file type, 77
 - lists, 517–528
 - advanced, 522–528
 - AutoFiltering, 517–522
 - for queries, 546
- Financial category functions, 660–661
- financial functions, 223–225. *See also specific names of functions*
 - depreciation, 223–224
 - loan and annuity, 224–225
- financial models, 346–348
- finding files, 79
- finding lost workbooks, 79
- first key sort order, 533
- first page number, 267
- fixed value error bar, 385
- floating toolbars, 59
- floor of 3D charts, 367, 392
- flow diagrams, 340
- folder(s)
 - holding existing workbooks, 77
 - Personal, 82
 - XlStart, 80
- font(s), 113, 237–242
 - changing, 239–241
 - default, 238–239
 - displaying, 736
 - problems with, 279
 - in spreadsheets, 353
 - TrueType, 279
- footers, 269–271
 - multiline, 271
- Form Wizard, 516
- formatting
 - of arrays, 464
 - of AutoShape objects, 328–330
 - of cells, by pasting, 159
 - of charts, 851–852
 - Conditional, 109
 - selecting cells with, 147
 - of data, 556
 - of numbers, automatic, 103
 - of spreadsheet, 353–354
 - stylistic. *See stylistic formatting*
 - of tables, 31–34
 - pivot tables, 576–577
 - of text, 9
 - of values, 102–112
 - automatic, 103
 - custom, 107, 108–112
 - types of, 104–108
 - using shortcut keys, 104
 - using toolbar, 103–104
- formatting codes, 109–111
- Formatting toolbar, 38–59, 103–104, 237
- forms
 - data entry, 118–119
 - Microsoft Access, 516–517
- Forms toolbar, 824
- formula(s), 169–198. *See also array formulas; function(s)*
 - AutoCorrect feature for, 174, 700
 - AutoFill instead of, 187
 - automatic adjustment in, 17
 - calculation of, 183–184
 - cell referencing in
 - absolute, 178, 179–180, 181, 191
 - circular, 184–188, 706
 - invalid, 182
 - nonrelative, 181
 - relative, 178–179, 180, 191
 - outside worksheet, 177–178
 - color coding of cells with, 703
 - consolidating worksheets by using, 449–451
 - converting to values, 197–198
 - creating, 31
 - as data type, 95
 - debugging. *See troubleshooting*
 - editing, 182–183
 - selecting character during, 183
 - elements of, 169
 - entering, 174–176
 - manual, 175
 - pasting names, 176
 - by pointing, 175–176, 178
 - in spreadsheet, 352–353
 - error returned by, 181–182
 - examples of, 170, 171
 - exponential smoothing and, 646
 - external reference, 442–446
 - creating by pasting, 443
 - creating by pointing, 443
 - opening workbook with, 444–445
 - potential problems with, 447–448
 - syntax for, 442–443
 - filtered lists and, 519
 - “hard coding” values in, 196, 358
 - inserting, 756
 - intermediary, 232–234, 469
 - making exact copy of, 197
 - names in, 188–196
 - applied to existing references, 194–196
 - constants, 190–191
 - multisheet, 189–190
 - sheet-level, 188–189
 - naming, 191–192
 - operators in, 169, 170–174
 - precedence of, 172–174
 - outlines and, 434
 - pasting as values, 159
 - performing mathematical operations
 - without, 159–160
 - selecting cells with, 146
 - summary, 434
 - viewing, 712
- Formula Bar, 23, 97, 100, 101
 - brackets in, 463
 - as calculator, 196–197
- formula list, 707
- Formula Palette, 100–101, 205
 - entering formulas using, 175–176
 - for pasting functions, 204–209
 - to specify arguments, 205–206
- FOR-NEXT loop, 743, 779
- Fourier Analysis tool, 648
- fractions, 118
- freezing/unfreezing panes, 131–132
- “front end” for users, 756
- frozen panes, custom views of, 278
- F-Test (Two-Sample Test for Variance), 647–648
- FTP (File Transfer Protocol), 686
- FTP sites, 686, 687
- function(s), 106, 199–234
- Function keyword, 788
- Function list box (Consolidate dialog box), 452
- functions, 8, 174. *See also VBA functions; specific names of functions*
 - add-ins to simplify access to, 860
 - Analysis ToolPak, 641, 657–662
 - Date & Time category, 657–658
 - Engineering category, 658–659
 - Financial category, 660–661
 - Information category, 661–662
 - Math & Trig category, 662
 - arguments of, 201–203
 - expressions as, 202–203
 - literal, 202
 - names as, 202
 - other functions as, 203
 - database, 231–232, 528–529. *See also specific names of functions*
 - with lists, 528–529
 - date, 221–222
 - defined, 199, 759
 - editing and, 200
 - entering, 203–209
 - manual, 203–204
 - by pasting, 204–207
 - tips for, 208–209
 - examples of, 199–200
 - financial, 223–225
 - depreciation, 223–224
 - loan and annuity, 224–225
 - in formulas, 169

- information, 218–221
 - logical, 216–218
 - lookup, 225–229
 - mathematical and trigonometric, 209–213
 - nested, 203, 208
 - reference, 225–229
 - statistical, 229–231
 - text, 214–216
 - time, 221, 223
 - trigonometric, 725–726
 - volatile, 210
 - FV function, 225
 - FVSCHEDULE function, 661
- G**
- games, 715–720
 - Keno, 717
 - Moving Tile Puzzle, 716
 - in Power Utility Pak, 717–720
 - Bomb Hunt, 718–719
 - Dice Game, 718, 719
 - Hangman, 719–720
 - Video Poker, 717–718
 - Tick-Tack-Toe, 715–716
 - Gantt chart, 399–401
 - GCD function, 662
 - general number format, 107
 - GESTEP function, 659
 - GIF files, 322
 - animated, 320
 - goal seeking, single-cell, 10, 618–622
 - example of, 618–619
 - graphical, 620–622
 - graduated symbol map format, 414–415, 417
 - grand totals, 562
 - graphical goal seeking, 620–622
 - graphics, 317–325. *See also* drawing
 - tools
 - bitmap vs. vector images, 321
 - Clip Gallery, 318–320
 - copying, 667
 - copying, using Clipboard, 322–323
 - file formats supported, 322
 - importing, 320–322
 - from digital camera or scanner, 324–325
 - programming, 848–850
 - graphs. *See* chart(s)
 - greater than operator, 170, 172, 526
 - greater than or equal to operator, 170, 526
 - gridlines, 364
 - borders and, 114
 - on chart, 364, 367, 371–372
 - printing, 272
 - grouping
 - by dates, 593
 - drawing objects, 331
 - for outlines, 435–436
 - in pivot table, 562
 - guitar fret board application, 722
- H**
- Hangman, 719–720
 - hard coding values, 596
 - hard disk, 71
 - hard disk failure, 90
 - header row, 513
 - sorting and, 534
 - heading(s), 269–271
 - column, 119, 131
 - entering, 28–30
 - multiline, 271
 - printing row and column, 272
 - row, 131
 - using as names, 195
 - on workbook window, 26
 - help
 - for commands, 58
 - HTML system, 19
 - interactive, 7
 - online, 784
 - HEX2BIN function, 659
 - HEX2DEC function, 659
 - HEX2OCT function, 659
 - HGL files, 322
 - hidden rows and columns, custom
 - views of, 278
 - hiding
 - of cells before printing, 280–281
 - controlling data series by, 384–385
 - items in pivot tables, 576
 - outline symbols, 439
 - rows and columns, 136–137
 - toolbars, 61–62, 735
 - of worksheets, 125
 - high-low lines, 367
 - Histogram tool, 474, 648–649
 - histograms, comparative, 401–402
 - HLOOKUP function, 226
 - Home key, 101
 - horizontal cell alignment, 242–244
 - horizontal scrollbar, on workbook window, 26
 - hot (accelerator) key, 56, 65, 820. *See also* shortcut keys
 - hour function, 223
 - HTML files, 16, 73, 493, 496, 690–691
 - defined, 686
 - pasting, 672, 674
 - “HTML Help” system, 19
 - HTTP, defined, 686
 - hyperlinks, 686, 693–694
- I**
- icons, large, 736
 - IDG Books Web site, 689
 - IF function, 217–218, 462
 - If-Then construct, 778–779
 - IMABS function, 659
 - image control, 833
 - image import options, 19
 - image on toolbar button, 744–745
 - images. *See* graphics
 - IMAGINARY function, 659
 - IMARGUMENT function, 659
 - IMCONJUGATE function, 659
 - IMCOS function, 659
 - IMDIV function, 659
 - IMEXP function, 659
 - IMLN function, 659
 - IMLOG10 function, 659
 - IMLOG2 function, 659
 - importing data, 491–509
 - from another Windows application, 600–601
 - through Clipboard, 497–502
 - file formats supported, 492–496
 - database, 493, 495
 - HTML, 493, 496
 - Lotus 1-2-3, 493, 494–495
 - other, 493
 - Quattro Pro, 493, 495
 - text, 493, 495–496
 - graphics, 320–322
 - from digital camera or scanner, 324–325
 - methods of, 491
 - from non-Windows application, 501–502
 - overview, 491
 - from text files, 502–509
 - using Text Import Wizard, 504–509
 - IMPOWER function, 659
 - IMPRODUCT function, 659
 - IMREAL function, 659
 - IMSIN function, 659
 - IMSQRT function, 659
 - IMSUB function, 659
 - IMSUM function, 659
 - Increase Decimal button, 103
 - incremental values, AutoFill to create, 187
 - INDEX function, 208, 227, 228
 - indirect circular references, 186
 - INDIRECT function, 228–229
 - indirect precedent, 701
 - INFO function, 220
 - Information category functions, 661–662
 - information functions, 218–221. *See also specific names of functions*
 - InputBox function, 802–803, 844–845

- insert key, 101
- inserting. *See also* pasting
 - buttons, 738
 - CommandButton, 829–830
 - formulas, 756
 - page breaks, 276
 - rows and columns, 133–135
 - text in AutoShapes, 335
 - text strings, 756
 - WordArt image, 334
- insertion point, 101
- installation, 15
- INT function, 209
- intentional circular references, 186–188
- interactive help, 7
- interface, multiple document, 6
- intermediary formulas, 469
- intermediary link, 447
- Internet, 685–697
 - accessing, 687
 - defined, 685, 686
- Internet Assistant Wizard, 859
- Internet Explorer, 692
- Internet Service Provider (ISP), 687
- Internet support, 14
- intersection operator, 192–193
- intersection, range, 182, 192–194
- intranet, saving over, 16
- INTRATE function, 661
- investment portfolio, optimizing, 636–638
- invoice template, 89
- IPMT function, 225
- is between operator, 549
- is greater than operator, 549
- is greater than or equal to operator, 549
- is less than operator, 549
- is less than or equal to operator, 549
- is not between operator, 549
- is not Null operator, 549
- is not one of operator, 549
- is Null operator, 549
- is one of operator, 549
- ISBLANK function, 219
- ISERR function, 219
- ISError function, 219, 220–221
- ISEVEN function, 662
- ISLOGICAL function, 219
- ISNA function, 219
- ISNONTEXT, 219
- ISNUMBER function, 219
- ISODD function, 662
- ISP (Internet Service Provider), 687
- ISPMT function, 225
- ISREF function, 219
- ISTEXT function, 219
- items
 - calculated, 583–585
 - in pivot tables, 562, 576
- iteration setting, circular references and, 185, 188
- J**
- JPG files, 322
- K**
- Keno, 717
- keyboard
 - menu manipulation with, 55–56
 - navigation using
 - through dialog boxes, 65
 - through worksheets, 49–50
 - keyboard commands
 - to display shortcut menus, 58
 - to manipulate windows, 47
 - keyboard shortcuts. *See* shortcut keys
 - keys in chart legend, 370
- L**
- label(s)
 - data, in charts, 365–366, 382–383
 - in lists, 513
 - on maps, 419–420
 - tick mark, 374
- Label control, 833
- LARGE function, 231
- last cell, selecting, 146
- layout, flexible, 358
- LCM function, 662
- Left arrow, 101
- LEFT function, 202, 215
- legends
 - chart, 367, 370–371
 - map, 421–422
- LEN function, 215, 470
- less than operator, 170, 172, 526
- less than or equal to operator, 170, 526
- like operator, 549
- line break, 271
- line chart, 285, 306
- linear trends, 387
- lines, 249–252
- link(s)
 - for data recovery, 449
 - intermediary, 447
- link formulas. *See* external reference formulas
- linked charts, 587
- linking
 - data, data sharing by, 669–670
 - text in object to cell, 339–340
 - workbooks, 441–448
 - changing link source, 446
 - examining links, 445
 - external reference formulas for, 442–446
 - reasons for, 441–442
 - to recover data from corrupted files, 449
 - severing links, 446
 - to unsaved workbook, 444
 - updating links, 446
 - worksheets, consolidating
 - worksheets and, 442
- link-update requests, 670
- list(s), 511–38. *See also* database(s)
 - customer, 481
 - data entry into, 514–517
 - with Data Form dialog box, 515
 - Microsoft Access forms for, 516–517
 - database functions with, 528–529
 - database tables, 231
 - defined, 511–512
 - designing, 513
 - drop-down, 518
 - example of, 512
 - filtering, 517–528
 - advanced, 522–528
 - AutoFiltering, 517–522
 - formula, 707
 - mailing, 687, 689
 - of names, pasting, 713
 - pick, 514
 - pivot tables and, 565
 - size limits on, 512
 - sorting, 529–536
 - complex, 530–534
 - custom, 534–535
 - file list, 78
 - simple, 530
 - subtotals from, 536–538
 - uses of, 512–513
- list management, 7
 - spreadsheets for, 348, 349
- ListBox control, 833–834
- list-boxes, 67
 - Font Size, 34, 35
- literal arguments, 202
- loan and annuity functions, 224–225
- logical comparison operators, 170
- logical functions, 216–218. *See also* specific names of functions
- lookup functions, 225–229. *See also* specific names of functions

Lookup Wizard, 859
 looping
 with arrays, 461–462
 through ranges, 843–844
 Lotus 1-2-3, 5, 493, 494–495
 file formats, 73
 spreadsheets, 201
 Lotus Word Pro, 671
 LOWER function, 216

M

macro-assisted what-if analyses, 597–599
 macros, 161. *See also* VBA functions
 assigning to toolbar button, 770
 creating, 760
 defined, 597, 755, 759
 for dialog box controls, 829–830
 editing, 765
 examining, 762–764
 example of, 765–767
 non-recordable, 780–783
 proprietary, 858
 recording, 760–762, 767–769
 absolute vs. relative, 768–769
 storing, 769–770
 testing, 764–765
 XLM language, 755
 mailing lists, 687, 689
 mailing workbook as e-mail attachment, 485–486
 mainframe systems, 479
 maintenance
 of spreadsheet, 358
 of workbook, 358
 major gridlines, 372
 major tick marks, 373
 Manual Calculation mode, 183
 manual what-if analyses, 597
 MAPI (Messaging Application Programming Interface), 485
 maps/mapping, 405–427
 adding and removing features, 422–424
 adding data to, 426
 available in Microsoft Map, 407–409
 converting to picture, 426
 creating, 409–417
 category shading, 412–413
 column-chart maps, 416, 417
 combined formats, 416
 data setup, 409
 dot density, 413–414
 formats, 410–411
 graduated symbol, 414–415, 417
 pie-chart maps, 415–416
 value shading, 411–412, 417

 customizing, 417–418
 example of, 406–407
 integrated mapping, 9
 labels on, 419–420
 legends for, 421–422
 overview of, 405
 pins on, 420–421
 pivot table data to create, 591–592
 plotting U.S. zip codes, 424–425
 repositioning, 419
 templates, 426
 worksheet, 707
 zooming in and out, 418–419
 margin adjustments, 268
 in print preview window, 275
 MATH function, 227, 228
 Math & Trig category functions, 662
 mathematical functions, 209–213. *See also specific names of functions*
 mathematical operations, performing
 without formulas, 159–160
 matrix, covariance, 644
 MAX function, 230
 MAXA function, 231
 maximized state, 43
 MDURATION function, 661
 megaformulas, 232–234
 memory, virtual, 72
 menu(s), 51–57
 animated, 737
 automatic customization of, 55
 dialog boxes invoked through, 54
 grayed out items on, 54
 keyboard manipulation of, 55–56
 mouse manipulation of, 52–55
 moving, 56–57
 personalized, 17–18, 736
 shortcut (context menus), 57–58
 shortcut keys associated with items
 on, 54
 submenus, 53
 menu bar, 23, 51–52
 as toolbar, 56
 Merge and Center, 112
 Merge and Center button, 35
 merging named styles, 261–262
 messages, preventing, 852–853
 Messaging Application Programming Interface (MAPI), 485
 methods, 778
 defined, 759
 of objects, 775, 778
 Mexico, map of, 408, 423
 Microsoft Access forms, 516–517
 Microsoft AccessLinks Add-In, 859
 Microsoft Bookshelf Integration, 859
 Microsoft Clip Gallery Live, 319

Microsoft Equation, 680, 681
 Microsoft IntelliMouse, scrolling with, 51
 Microsoft Map. *See* maps/mapping
 Microsoft Map toolbar, 418
 Microsoft Query, 540. *See also* query(ies)
 running alone, 556
 Microsoft Windows, 21
 Microsoft Word
 copying Clipboard information to, 667–668
 data sharing, 671–674
 pasting with link, 673–674
 pasting without link, 671–673
 new Excel object in, 677–678
 range embedded in Word document, 675–677
 workbook embedded in, 679
 Microsoft WordArt, 680, 681
 MID function, 215, 470
 MIN function, 230
 MINA function, 231
 Minesweeper, 718
 minimized state, 44
 minor gridlines, 372
 minor tick marks, 373
 mixed cell references, 180–181
 MOD function, 471
 modeless UserForms, 20
 module, defined, 759
 month(s)
 AutoFill to create series of, 187
 grouping by, 593
 MONTH function, 222
 mouse
 deleting cells using, 148
 menu manipulation with, 52–55
 navigating through windows
 without, 47
 navigating through worksheets
 using, 50–51
 to select characters, 102
 moving
 buttons, 738
 cells, 154–155
 charts, 299
 elements of charts, 301
 menus, 56–57
 queries, 553
 ranges, 154–155
 toolbars, 62–63, 733–734
 windows, 44–45
 worksheets, 124–125
 Moving Average tool, 650–651
 Moving Tile Puzzle, 716
 MROUND function, 662
 MS Organization Chart, 680

- MS Query Add-in for Excel 5
Compatibility, 859
- MsgBox function, 803-806
- multicolumn filtering, 519-520
- MULTINOMIAL function, 862
- multiple document interface, 6
- multiple selections, 142-143
- multiplication operator, 170, 172
- multisheet files. *See* workbook(s)
- multisheet names, 189-190
- multisheet ranges, selecting, 143-145
- multiuser applications, 480
- N**
- #N/A, 706
- NA function, 182
- #NAME?, 706
- Name box, 23, 163-164, 189
- name of field button, 575
- named ranges, 381-382, 515
in VBA code, 840
zooming, 128
- named styles, 257-262, 358
applying, 259
controlling with templates, 262
creating, 260
deleting, 261
merging, 261-262
modifying, 261
overriding, 260
- named views, 133
- names/naming, 188-196
applying to existing references,
194-196
as arguments, 202
of cells and ranges, 161-168, 171
advantages of, 161-162
automatic, 164-166
changing, 168
deleting, 167
manual, 162-164
redefining, 168
table of names, 166-167
valid, 162
constants, 190-191
errors in, 182
of files, 82
formulas, 191-192
of functions, 790
liberal use of, 358
multisheet, 189-190
range intersections and, 193-194
row and column headings used as, 195
in SERIES formula, using, 381-382
sheet-level, 188-189
worksheet views, 133
worksheets, 123-124
- Navigate Circular Reference box, 185
- navigating, 43-70
through cell content, 101
dialog boxes, 64-69
controls in, 64, 65-68
by keyboard, 65
tabbed, 68-69
giving commands, 51-64
reversing, 53
using menus, 51-57
using shortcut keys, 64
using shortcut menus, 57-58
using toolbars, 58-63
windows, 43-47
closing, 47
mouseless manipulation of, 47
moving and resizing, 44-45
states of, 43-44
switching among, 45-47
worksheets, 48-51
using keyboard, 49-50
using mouse, 50-51
- nested function, 208
- nested parentheses, 173
- nesting
of custom functions, 786
of IF functions, 217
of SEARCH function, 216
- nesting functions, 203
- network server, backup copies on, 90
- networks, Excel over, 479-488
file reservations, 479-481
mailing workbook as e-mail
attachment, 485-486
routing workbook, 486-488
shared workbooks, 481-484
advanced settings for, 483-484
appropriate sharing, 481
conflicting changes between
users, 484
designating, 482-483
limitations of, 482
personal views, 484
tracking changes, 484
updating changes, 484
- newsgroups, 687, 688
- "noisy" data, 387
- NOMINAL function, 661
- noncontiguous ranges, selecting,
142-143
- nondelimited text files, 503
- nonrelative cell references, 181
- normal distribution, 652
- Normal style, 258
- North America, map of, 408
- not equal to operator, 170, 526
- not like operator, 549
- notes. *See* comments
- NPER function, 225
- #NULL!, 706
- #NUM!, 706
- Num Lock key, 49
- number format, 107
- numbers, formatting. *See also* value(s)
automatic, 103
using shortcut keys, 104
using toolbar, 103-104
- numerical limitations, 95
- O**
- object(s)
active, 774
AutoShape, formatting of, 328-330
cropping, 678
defined, 769
methods of, 775, 778
pasting worksheet, 672, 674
properties of, 774-775, 776-778
selecting cells with, 146
VBA, 775-776
VBA manipulation of, 778
- object linking and embedding (OLE),
674-681
new Excel object in Word, 677-678
objects embedded in worksheet,
679-681
range embedded in Word document,
675-677
workbook embedded in Word, 679
- object model, 773, 774
- Object Packager application, 680
- object-oriented language, 775
- objects, drawing
aligning, 332
changing stack order, 330-331
formatting, 328-330
grouping, 331
sizing and rotating, 329
spacing evenly, 332
- OCT2BIN function, 659
- OCT2DEC function, 659
- OCT2HEX function, 659
- ODBC Add-in, 859
- ODBC Manager, 545
- ODBC (Open DataBase Connectivity),
542
- ODDFPRICE function, 661
- ODDFYIELD function, 661
- ODDLPRICE function, 661
- ODDLYIELD function, 661
- Office 2000, 685, 692
- Office 2000 Clipboard, 155-158
- Office Assistant, 7, 8, 19-20
- Office Client Pak, 692
- Office Clipboard, 18-19, 150-151, 497

- clearing contents of, 157
 - sharing data using, 666-668
 - toolbar, 157
 - Office toolbar, 155
 - offset block layout, 351
 - OFFSET function, 227, 403
 - OLAP Cube Wizard, 566
 - OLAP cubes, pivot tables and, 566
 - OLAP (online analytical processing)
 - database, 543, 566
 - calculated fields and, 582
 - calculated items and, 584
 - OLE (object linking and embedding), 674-681
 - new Excel object in Word, 677-678
 - objects embedded in worksheet, 679-681
 - range embedded in Word document, 675-677
 - workbook embedded in Word, 679
 - OLE support, 13
 - On Error statement, 851
 - 1-2-3. *See* Lotus 1-2-3
 - one-dimensional array, 457
 - one-input data tables, 600-603
 - OnKey method, 721
 - online analytical processing (OLAP)
 - databases, 543, 566
 - online help, 7, 784
 - online services, 687
 - Open DataBase Connectivity (ODBC), 542
 - opening workbooks, 75-80
 - automatic, 80
 - file display preferences, 77-78
 - filtering by file type, 77
 - specifying folder for, 77
 - Tools menu for, 78
 - operand errors, 182
 - operator(s), 169, 170-174. *See also specific types of operators*
 - comparison, 526
 - concatenation, 806
 - in formulas, 169
 - intersection, 192-193
 - precedence of, 172-174
 - query, 549
 - range, 201
 - reference, 193
 - option (radio) buttons, 65-66
 - OptionButton control, 834-835
 - OR function, 218
 - OR operator, 525
 - organizational charts, 338
 - embedded, 681
 - orientation, 267
 - cell, 245-246
 - of field button, 576
 - sorting by, 534
 - outlines. *See* worksheet outlines
- P**
- Page Break Preview mode, 265, 277-278
 - page breaks, 276-278
 - inserting, 276
 - previewing, 265, 277-278
 - removing, 276-277
 - page field, 562, 571
 - in pivot table, 561
 - page number, first, 267
 - page settings, 267
 - paired two-sample *t*-test for means, 656
 - panes
 - freezing, 131-132
 - splitting, 130
 - paper size, 267
 - parentheses
 - nested, 173
 - to set operator precedence, 172-174
 - Pareto (sorted histogram) option, 649
 - passwords, 83
 - in Protect Sheet dialog box, 355
 - Paste All button, 157
 - pasting
 - consolidating worksheets by, 451
 - of functions, 204-207
 - example of, 206-207
 - list of names, 713
 - names in formulas, 176
 - pictures of cells, 341-342
 - of VBA functions, 797-799
 - pasting information, 155-161
 - with Office Clipboard, 155-158
 - special ways of, 158-161
 - patterned distribution, 652
 - PCD files, 322
 - PCT files, 322
 - PCX files, 322
 - peer-to-peer networks, 479
 - Percent Style button, 103
 - percentage error bar, 385
 - percentage format, 107
 - Personal directory, 71
 - Personal folder, 82
 - Personal Macro Workbook, 769-770
 - personalized menu, 17-18, 736
 - PI function, 211
 - Pick from List menu, 117
 - pick lists, 514
 - picture charts, 395-399
 - Clipboard to create, 397-399
 - graphic file to create, 397
 - Picture toolbar, 324
 - pictures
 - modifying, 324
 - pasting, 672, 674
 - pictures of cells, 341-342
 - pie chart, 285, 287, 289, 307-308
 - pie-chart maps, 415-416
 - pins on maps, 420-421
 - Pivot Table Wizard, 550
 - pivot tables, 10, 213, 540, 569-593
 - calculated item in, 583-585
 - changing structure of, 574
 - charts from, 587-588
 - completing, 567-573
 - finished product, 571
 - layout, 568-571
 - options, 568, 572-573
 - concept of, 559-561
 - to consolidate worksheets, 585-587
 - creating, 564-567
 - data location, 565-566
 - data specification, 567
 - customer geographic analysis using, 591-592
 - data appropriate for, 562-564
 - displaying on different sheets, 581-582
 - fields in
 - adding, 575
 - calculated, 582-583
 - customizing, 575-576
 - removing, 575
 - formatting, 576-577
 - grouping by dates in, 593
 - grouping items in, 577-581
 - OLAP cubes and, 566
 - produced by Scenario Manager, 613-614
 - refreshing, 575
 - survey data analysis using, 588-590
 - terminology of, 562-563
 - PivotChart, 17
 - PivotChart Wizard, 564, 565
 - pivoting, 574
 - PivotTable toolbar, 436, 569-570
 - PivotTable, 17, 564, 565
 - plot area, 367, 368-369
 - plotting trigonometric functions, 725-726
 - plotting U.S. zip codes, 424-425
 - PMT function, 225
 - PNG files, 322
 - points in data series, 367
 - Poisson distribution, 652
 - portfolio, investment, 636-638
 - position, consolidating worksheets by, 452

- pound signs (#), 135, 181
 - Power Utility Pak, 383
 - games, 717-720
 - Bomb Hunt, 718-719
 - Dice Game, 718, 719
 - Hangman, 719-720
 - Select By Value utility, 783
 - Video Poker, 717-718
 - precedents
 - cell, 701, 702-704
 - selecting cells according to, 146
 - preformatting columns, 513
 - presentations, spreadsheets for, 348
 - PRICE function, 661
 - PRICEDISC function, 661
 - PRICEMAT function, 661
 - print area box, 271-272
 - print preview, 10
 - print quality, 267
 - print settings, custom views of, 278
 - print titles, 272
 - printer, selecting, 264-265
 - printing, 10, 263-282
 - charts, 302-303
 - custom views and, 278-279
 - font problems, 279
 - headers and footers, 269-271
 - hiding cells before, 280-281
 - margin adjustments, 268
 - multiple copies, 266
 - noncontiguous ranges on single page, 279-280
 - one-step, 263-264
 - page breaks, 276-278
 - inserting, 276
 - previewing, 265, 277-278
 - removing, 276-277
 - page settings, 267
 - print preview feature, 273-276
 - accessing, 274-275
 - making changes while previewing, 275-276
 - printer-specific options, 273
 - reports, 38
 - of selected pages, 266
 - settings for
 - adjusting, 264
 - in Page Setup dialog box, 264, 266-273
 - in Print dialog box, 264-266
 - sheet options, 271-273
 - specifying what to print, 266
 - templates to change defaults, 281
 - procedures. *See also* macros
 - defined, 759
 - variable declaration in, 855
 - programming. *See* VBA programming programs, 71
 - project tracking, 481
 - PROPER function, 216
 - property(ies)
 - control
 - CheckBox, 831
 - ComboBox, 832
 - ListBox, 834
 - OptionButton, 835
 - protection options, 13
 - protection, spreadsheet, 355-357
 - Purchase Order template, 89
 - puzzles, word search, 723-724
 - PV function, 225
 - pyramid charts, 285, 314-315
- Q**
- Quattro Pro, 5, 201, 493, 495
 - file formats, 73
 - query(ies)
 - defined, 542
 - on external database files, 540-550
 - changing, 553-554
 - copying or moving, 553
 - creating, 554-555
 - deleting, 553
 - external data ranges, 551-554
 - to get data, 541
 - multiple, 553
 - operators for, 549
 - without Query Wizard, 554-557
 - refreshing, 552-553
 - selecting data source, 542-545
 - starting, 542
 - using multiple database tables, 556
 - using Query Wizard, 545-550
 - refreshing, 549, 542, 552-553, 695
 - saving, 549
 - Web, 694-697
 - query operators, 549
 - quitting Excel, 38
 - QUOTIENT function, 662
- R**
- radar charts, 285, 311
 - RADIANS function, 203
 - radio (option) buttons, 65-66
 - RAND function, 201, 210
 - RANDBETWEEN function, 662
 - Random Number Generation tool, 651-652
 - range(s), 139-168, 839-846. *See also* pivot tables
 - as argument of function, 795-796
 - array, 464
 - array formulas for counting
 - characters in, 468-469
 - array formulas for identifying value
 - in, 467-468
 - array formulas for summing nth value
 - in, 471-472
 - AutoShape around, 849-850
 - bin, 648
 - copying, 149-154, 840-841
 - to adjacent cells, 153
 - to other sheets, 154
 - using drag and drop, 152-153
 - using menu commands, 151, 153
 - using shortcut keys, 152
 - using shortcut menus, 151
 - using toolbar buttons, 149
 - criteria, 523-524
 - data
 - automatically updated, 402
 - modifying, 379-380
 - defined, 139
 - embedded in Word document, 675-677
 - looping through, 843-844
 - moving, 154-155, 843
 - multiple consolidation, 566
 - named, 381-382, 515
 - in VBA code, 840
 - zooming, 128
 - names/naming of, 161-168, 171
 - advantages of, 161-162
 - automatic, 164-166
 - changing, 168
 - deleting, 167
 - manual, 162-164
 - redefining, 168
 - table of names, 166-167
 - valid, 162
 - noncontiguous, on single page, 279-280
 - prompting for cell value, 844-845
 - reference operators for, 193
 - selecting, 140-147
 - all sheets, 145
 - in array, 464
 - for charting, 296
 - complete rows and columns, 141-142
 - determining type of, 845-846
 - to end of row or column, 842
 - multiple, 846
 - multisheet ranges, 143-145
 - noncontiguous ranges, 142-143
 - row or column, 842
 - special selections, 145-147
 - sorting, 536
 - text box to match, 848-849
 - transposing, 160-161

- range intersection, 182, 192–194
 - Range object, 777
 - range operator, 201
 - range references
 - as arguments, 202
 - changeable, 204
 - range selection boxes, 66, 67
 - Rank and Percentile tool, 652–653
 - RANK function, 472–473
 - ranking, array formulas for, 472–473
 - RATE function, 225
 - RC reference style, 140
 - read-only files, 480
 - read-only option, 83
 - read-only properties, 776
 - recalculation, automatic, 3
 - RECEIVED function, 661
 - record(s), 542
 - adding, 556
 - editing, 556
 - recording macros, 760–762, 767–769
 - absolute vs. relative, 768–769
 - rectangle object to hide cells, 281
 - #REF!, 706
 - reference functions, 225–229. *See also specific names of functions*
 - reference operators, 193
 - Reference text box (Consolidate dialog box), 452
 - references
 - cell. *See* cell references
 - range, 202, 204
 - refreshing
 - pivot tables, 562, 575
 - queries, 540, 542, 552–553, 695
 - Registry, 72
 - Regression tool, 653–655
 - relational database, 542, 556
 - relative cell referencing, 178–179, 180, 191
 - relative recording, 768–769
 - relative references, naming formulas and, 191
 - removable medium, backup copies on, 90
 - removing page breaks, 276–277
 - renaming toolbars, 735
 - repeating information during data entry, 116
 - REPLACE function, 215
 - replacing cell contents, 99
 - report(s)
 - PivotChart, 587–588
 - printing, 38
 - scenario, 612–614
 - Solver-generated, 626, 627
 - spreadsheets for, 348
 - summary, 707
 - Report Manager, 859
 - reservation, file, 479–481
 - resizing
 - charts, 299
 - windows, 44–45
 - resource allocation, 635–636
 - restored state, 44
 - result set, 542
 - reversing commands, 53
 - Right arrow, 101
 - RIGHT function, 215
 - ripple effect, 182, 699
 - RLE files, 322
 - ROMAN function, 210–211
 - rotating drawing object, 329
 - ROUND function, 106, 211
 - routing workbooks, 486–488
 - row(s), 133–137
 - array formulas for returning last value in, 476
 - changing heights of, 136
 - copying, 528
 - deleting, 135
 - displaying unique, 528
 - hiding, 136–137
 - inserting, 133–135
 - in lists, 513
 - selecting complete, 141–142
 - row differences, selecting cells with, 146
 - row field, 563, 574
 - ROW function, 470
 - row headings, 28–30, 131
 - printing, 272
 - sorting and, 534
 - using as names, 195
 - on workbook window, 26
 - RTF file, pasting, 672, 674
 - run time, 810
- S**
- Sampling tool, 655
 - saving
 - in HTML format, 690
 - queries, 549
 - to server, 16
 - templates, 752–753
 - workbooks, 37–38, 80–87
 - default location for, 82
 - file naming rules, 82
 - in older formats, 86–87
 - options for, 83–84
 - summary information, 84–85
 - as text file, 265
 - scaling, 267
 - scanner, importing graphics from, 324–325
 - scenario, defined, 597
 - scenario management, 11
 - Scenario Manager, 598, 600, 607–615
 - defining scenarios, 608–610
 - displaying scenarios, 611
 - limitations of, 614–615
 - merging scenarios, 612
 - modifying scenarios, 611
 - report generation, 612–614
 - Scenario tool, 611
 - scientific format, 107
 - scrap, 677
 - Screen Tips, 737
 - screen updates, 852
 - Scroll Lock key, 49
 - ScrollBar, 836
 - control
 - SpinButton, 836
 - TextBox, 837–838
 - of Control Toolbox controls, 826–828
 - defined, 759
 - of files, 71–72
 - of objects, 774–775, 776–778
 - ScrollBar control, 835–836
 - scrollbars, 50–51
 - on workbook window, 27
 - SEARCH function, 216
 - security, file, 83
 - see-through cell selection, 17
 - “See-through View,” 141
 - Select All button on workbook window, 26
 - Select by Value utility, 783
 - Select Case construct, 780
 - selected cells and ranges, custom views of, 278
 - selecting
 - cells
 - during data entry, 116
 - see-through, 17
 - columns
 - complete, 141–142
 - multiple, 135
 - for querying, 545
 - data series, 377
 - during editing formulas, 183
 - with mouse, 102
 - parts of charts, 301, 366
 - printing selection, 266
 - ranges, 140–147
 - all sheets, 145
 - in array, 464
 - complete rows and columns, 141–142
 - multisheet ranges, 143–145
 - noncontiguous ranges, 142–143
 - special selections, 145–147
 - SelectSpecial method, 844
 - self-repair feature, 15

- sequential routing, 487
 - serial number system, date, 98
 - series, AutoFill to create, 187
 - series axis, 367
 - SERIES formula
 - editing, 380–381
 - using names in, 381–382
 - SERIES function, 381
 - SERIESSUM function, 662
 - server, 479, 480
 - backup copies on, 90
 - Microsoft news, 688
 - saving to, 16
 - setup program, 72
 - to install mapping features, 410
 - shading, 246–249
 - shadow, AutoShapes, 333–334, 336–337
 - shared workbooks, 82, 88, 481–484
 - advanced settings for, 483–484
 - conflicting changes between users, 484
 - personal views, 484
 - tracking changes, 484
 - updating changes, 484
 - appropriate sharing, 481
 - designating, 482–483
 - limitations of, 482
 - shared worksheets, with Analysis ToolPak functions, 641
 - sharing, data. *See under* data
 - sheet(s). *See also* worksheet(s)
 - chart, 284, 292, 294–295
 - displaying pivot tables on different, 581–582
 - printing selected, 266
 - sheet tabs on workbook window, 26
 - sheet.xlt file, 750
 - sheet-level names, 188–189
 - Shift+End, 102
 - Shift+Home, 102
 - Shift-left/right arrow, 102
 - shortcut keys
 - to activate sheet, 122
 - associated with menu items, 54
 - for Clipboard operations, 501
 - to copy ranges, 152
 - to create new workbook, 74
 - for executing macros, 766–767
 - formatting values using, 104
 - giving commands using, 64
 - for grouping, 436
 - for inserting worksheet, 123
 - to open existing workbook, 75
 - to recalculate formulas, 184
 - for repeating information, 116
 - to select precedents and dependents, 702–703
 - to show function's argument, 208
 - for ungrouping, 436
 - shortcut menus (context menus), 57–58
 - to copy ranges, 151
 - customizing, 733
 - simultaneous routing, 487
 - SIN function, 212
 - single-cell goal seeking, 10, 618–622
 - example of, 618–619
 - graphical, 620–622
 - single-factor analysis of variance, 642
 - sizing drawing object, 329
 - SLN function, 224
 - SMALL function, 231
 - smoothing constant (damping factor), 646
 - snapshots, 279
 - Solver, 622–638, 859
 - appropriate problems for, 623
 - examples, 623–627
 - allocating resources, 635–636
 - minimizing shipping costs, 629–631
 - optimizing investment portfolio, 636–638
 - scheduling staff, 631–635
 - Options dialog box, 627–628
 - Sort Ascending button, 530, 536
 - Sort Descending button, 536
 - sorted histogram (Pareto) option, 649
 - sorting
 - data, 556
 - lists, 529–536
 - complex, 530–534
 - custom, 534–535
 - file list, 78
 - simple, 530
 - order of, for query, 547–548
 - rules for, 532
 - sound file player, 724
 - source application, 500, 666
 - source data, 563
 - source workbook, 441, 447
 - space-delimited text files, 496
 - special number format, 107
 - Special Cells method, 844
 - spell checking, 11, 708–709
 - SpinButton control, 836–837
 - spitners, 66–67
 - splitting panes, 130
 - spreadsheet(s), 343–360
 - audience for, 343–345
 - budget, 347
 - characteristics of successful, 345–346
 - creating, 349–358
 - considering audience, 350–351
 - designing workbook layout, 351–352
 - developing a plan, 350
 - entering data and formulas, 352–353
 - formatting, 353–354
 - maintenance of, 358
 - protection, 355–357
 - testing, 354
- defined, 3
- errors in, 358–359
- for-your-eyes-only, 344
- maintenance of, 358
- quick-and-dirty, 344
- uses of, 346–349
 - database access, 348
 - financial or data-analysis models, 346–348
 - list management, 348, 349
 - reports and presentations, 348
 - turnkey applications, 349
- Spreadsheet Solutions templates, 89
- SQL (Structured Query Language), 542
- SQRT function, 202, 212, 459
- SQRTPI function, 662
- stacked column chart, 304
- staff scheduling, 631–635
- standard deviation error bar, 385
- standard error bar, 385
- Standard toolbar, 31, 58–59
- starting Excel, 21
- states, window, 43–44
- statistical functions, 229–231. *See also specific names of functions*
- status bar, 22, 24
- status box, Goal Seek, 619
- stock charts, 285, 313–314
- string criteria, 526
- Structured Query Language (SQL), 542
- style(s)
 - applied to worksheet outlines, 437–439
 - built-in, 258
 - named, 257–262, 358
 - applying, 259
 - controlling with templates, 262
 - creating, 260
 - deleting, 261
 - merging, 261–262
 - modifying, 261
 - overriding, 260
- Style tool, 259
- stylistic formatting, 112–114, 235–262
 - alignment, 112
 - attributes, 113
 - AutoFormatting, 253–256
 - controlling, 256
 - using, 254–255
 - background, 252–253
 - borders and lines, 113–114, 249–252
 - 3D effects, 252
 - cell alignment, 242–246
 - horizontal, 242–244

- text control options, 244–245
 - vertical, 244
 - cell orientation, 245–246
 - colors and shading, 114, 246–249, 251
 - conditional, 256–257
 - font and text size, 113, 237–242
 - changing, 239–241
 - default, 238–239
 - multiple, in one cell, 241–242
 - named styles, 257–262
 - applying, 259
 - controlling with templates, 262
 - creating, 260
 - deleting, 261
 - merging, 261–262
 - modifying, 261
 - overriding, 260
 - overview of, 235–237
 - reasons for, 235–236
 - time for, 236
 - Sub keyword, 757
 - Sub statement, 767
 - submenus, 53
 - subroutines, 598, 787–788. *See also* macros
 - defined, 759
 - event-handler, 810, 812–813, 817–819
 - procedure, 772
 - VBA, 757–758
 - subtotals, 563
 - customizing, 576
 - subtraction operator, 170, 172
 - SUM function, 202, 212, 471, 569
 - SUMIF function, 213
 - summary formulas, 434
 - summary information, workbook, 84–85
 - summary report, 707
 - surface charts, 285, 311–312
 - survey data analysis using pivot tables, 588–590
 - switching among windows, 45–47
 - SYD function, 224
 - Symbolic Link files (SYLK), 496
 - file format, 73
 - symbols, Euro currency, 19
 - symmetrical pattern drawing, 721
 - syntax errors, 772
- T**
- Tab, 65
 - tab order, in custom dialog boxes, 821
 - tab scroll buttons, 122
 - on workbook window, 26
 - tab split bar on workbook window, 26
 - tabbed dialog boxes, 68–69
 - tab-delimited text files, 496
 - table(s)
 - data, 367, 390, 391, 599–607
 - limitations of, 607
 - one-input, 600–603
 - two-input, 603–607
 - database (lists), 231
 - defined, 542
 - dynamic crosstab, array formulas in, 475–476
 - formatting, 31–34
 - of names, 166–167
 - pivot. *See* pivot tables
 - TABLE function, 603
 - Tables pane, 555
 - target cell, 623
 - Taskbar, Windows, 18
 - TBILLERQ function, 661
 - TBILLPRICE function, 661
 - TBILLYIELD function, 661
 - template(s), 11, 12, 74, 88–90, 747–754
 - to change default printing, 281
 - controlling named styles with, 262
 - default worksheet, 747, 749–750
 - defined, 747
 - map, 426
 - overview of, 747
 - workbook
 - custom, 747, 750–754
 - default, 747, 748–749
 - included in Excel, 751
 - operation of, 751
 - storing, 753
 - Template Utilities, 859
 - Template Wizard with Data Tracking, 859
 - terminology
 - of databases, 542
 - of Internet, 686
 - of pivot tables, 562–563
 - of VBA, 759
 - testing spreadsheet, 354
 - text, 94
 - centering, 112
 - changing or erasing, 99–102
 - in chart legend, 370
 - in drawing object, 330
 - embedding in worksheet, 679–680
 - entering, 96
 - file formats supported, 493, 495–496
 - in formulas, 169
 - inserting, in AutoShapes, 335
 - in map labels, 420
 - in object to cell, linking, 339–340
 - pasting, 672, 674
 - using outline for, 435
 - text box object to hide cells, 281
 - text boxes, 8
 - to match range, 848–849
 - text criteria, 525, 526
 - text file format, 73
 - text files, 499–500
 - characteristics of, 503–504
 - delimited, 496, 503
 - importing data from, 502–509
 - using Text Import Wizard, 504–509
 - nondelimited, 503
 - saving workbook as, 265
 - types of, 496
 - text format, 107
 - text formatting, 9
 - text functions, 214–216. *See also specific names of functions*
 - text handling, 8
 - Text Import Wizard, 504–509
 - text size, 113
 - text strings, inserting, 756
 - TextBox control, 837–838
 - TGA files, 322
 - 3D charts, 290, 365, 392–394
 - modifying, 392
 - rotating, 392–394
 - from two-input data tables, 606–607
 - 3D effects, 252
 - AutoShapes, 333–334, 336–337
 - tick marks, 373
 - Tick-Tack-Toe, 715–716
 - TIF files, 322
 - time(s), 97, 98
 - custom formatting codes for, 110–111
 - entering current, 117
 - time format, 107
 - TIME function, 223
 - time functions, 221, 223. *See also specific names of functions*
 - title(s)
 - adding, 34–35
 - chart, 365, 369–370
 - print, 272
 - title bar, 22
 - on workbook window, 24
 - TODAY function, 221–222
 - ToggleButton control, 838
 - tool(s). *See also* drawing tools
 - in Analysis ToolPak, 640–657
 - Analysis of Variance, 642–643
 - Correlation, 643–644
 - Covariance, 644
 - Descriptive Statistics, 645–646
 - Exponential Smoothing, 646–647
 - Fourier Analysis, 648
 - F-Test (Two-Sample Test for Variance), 647–648
 - Histogram, 648–649
 - Moving Average, 650–651
 - Random Number Generation, 651–652

continued

- tool(s), (*continued*)
 in Analysis ToolPak, (*continued*)
 Rank and Percentile, 652-653
 Regression, 653-655
 Sampling, 655
 t-Test, 656
 z-Test (Two-Sample Test for Means), 656-657
 analytical, 10
 auditing, 354, 706-707
 for tracing cell relationships, 703-705
 Chart Objects, 365
 copy, 157
 Cropping, 678
 Histogram, 474
 Scenarios, 611
 toolbar(s), 23, 58-63, 731-734
 adding/removing buttons, 738-739
 attaching to workbook, 735-736
 Auditing, 703-704
 Trace Error button of, 705
 autosensing, 737
 changing button functionality, 740-741
 changing button image, 740, 744-746
 Chart, 293, 294, 365
 Circular Reference, 185, 186
 Clipboard, 19, 60
 configuration of, 733
 Control Toolbox, 824
 creating new, 735, 741-744
 customization of, 8, 18
 deleting, 735
 displaying, 735
 Drawing, 325-327
 in Excel 2000, 59
 floating, 59
 Formatting, 58-59, 103-104, 237
 formatting values using, 103-104
 Forms, 824
 giving commands using, 58-63
 hiding/showing, 61-62, 735
 list of, 60-61
 menu bar as, 56
 Microsoft Map, 418
 moving, 62-63, 733-734
 Office, 155
 Office Clipboard, 157
 personalized, 736
 Picture, 324
 PivotTable, 436, 569-570
 renaming, 735
 resetting, 735
 Screen Tips on, 737
 Standard, 31, 58-59
 Web, 692-693
 WordArt, 334
 toolbar buttons. *See* button(s), toolbar
- Tools menu for opening workbooks, 78
 Top 10 filtering, 521
 tracers, cell, 704
 tracing cell relationships, 700-706
 auditing tools for, 703-705
 circular references, 706
 Go To Special dialog box, 701-703
 tracing error values, 705-706
 transition options, 50
 transposing ranges, 160-161
 trendlines, 367, 386-388
 trigonometric functions, 209-213. *See also specific names of functions*
 plotting, 725-726
 troubleshooting, 699-713
 AutoComplete, 710-711
 AutoCorrect, 709-710
 formula, 700
 learning about unfamiliar spreadsheet, 711-713
 pasting list of names, 713
 viewing formulas, 712
 zooming out, 711-712
 other auditing tools, 706-707
 spell checking, 708-709
 tracing cell relationships, 700-706
 auditing tools for, 703-705
 circular references, 706
 Go To Special dialog box, 701-703
 tracing error values, 705-706
 types of worksheet problems, 699-700
 TrueType fonts, 279, 353
 t-Test tool, 656, 657
 turnkey applications, spreadsheets for, 349
 two-dimensional array, 457
 two-factor analysis of variance
 with replication, 642
 without replication, 642
 two-input data tables, 603-607
 Two-Sample Test for Means (z-Test)
 tool, 656-657
 Two-Sample Test for Variance (F-Test), 647-648
 two-sample t-test
 assuming equal variances, 656
 assuming unequal variance, 656
 TXT extension, 504
 TYPE function, 219
- U**
 Undo command, 53
 Undo stack, 84
 unhiding. *See* hiding
 uniform distribution, 652
 Uniform Resource Locator (URL), 686
 United Kingdom, map of, 423
 United Kingdom Standard Regions, map of, 408
 United States, maps of, 408, 423
 up-bars, 367
 Update Add-In Links, 859
 updating changes, 484
 UPPER function, 216
 URL, defined, 686
 Use labels in check boxes (Consolidate dialog box), 453
 UserForms, 807
 defined, 759
 modeless, 20
- V**
 validating data entry, 114-115
 validation criteria, pasting, 159
 validation, selecting cells set up for data, 147
 value(s), 93-94, 95
 #VALUE!, 641
 #VALUE!, 706
 value(s). *See also* numbers, formatting
 changing or erasing, 99-102
 on worksheet, by dragging, 394-395
 converting formulas to, 197-198
 date, 97-98
 entering, 96
 errors related to, 182
 extreme input, 354
 formatting, 102-112
 automatic, 103
 custom, 107, 108-112
 types of formats, 104-108
 using shortcut keys, 104
 using toolbar, 103-104
 in formulas, 169, 196
 hard-coding, 358, 596
 in formulas, 196
 incremental, AutoFill to create, 187
 pasting formulas as, 159
 in range, array formulas for identifying, 467-468
 time, 97, 98
 value axis, 289, 367
 value axis title, 369
 value criteria, 525, 526
 value shading map format, 411-412, 417
 variable(s)
 declaring, 853-855
 in VBA, 778
 variance, Two-Sample Test for, 647-648
 VBA. *See* Visual Basic for Applications (VBA)
 VBA code, preventing access to, 860

- VBA functions, 757, 758–760, 785–799
 analyzing, 788
 arguments of, 792–796
 no argument, 792
 one argument, 793–794
 range argument, 795–796
 two arguments, 794–795
 debugging, 796
 declaring, 789–790
 example of, 786–788
 executing, 790–791
 learning more about, 799
 limits of, 790
 overview of, 785
 pasting, 797–799
 in VBA subroutine, 787–788
 in worksheet, 786–787
- VBA programming, 839–855
 changing settings, 847–848
 Boolean settings, 847
 non-Boolean settings, 847–848
 charts, 850–852
 applying formatting to, 851–852
 modifying properties of, 851
 modifying type of, 850–851
 graphic objects (shapes), 848–850
 AutoShape around range, 849–850
 text box to match range, 848–849
 ranges, 839–846
 copying, 840–841
 determining type of selection, 845–846
 identifying multiple selection, 846
 looping through, 843–844
 moving, 843
 prompting for cell value, 844–845
 selecting row or column, 842
 selecting to end of row or column, 842
 speed tips, 852–855
 alert messages, 852–853
 declaring variable types, 853–855
 screen updates, 852
 simplifying object references, 853
- VDB function, 224
 vector images, 321
 Vendor Independent Messaging (VIM), 485
 vertical cell alignment, 244
 vertical scrollbar on workbook window, 27
 video clips, embedding, 680
 video modes, 351
- Video Poker, 717–718
 viewing formulas, 712
 views of worksheets
 multiple, 128–129
 naming, 133
 Village template, 89
 VIM (Vendor Independent Messaging), 485
 virtual memory, 72
 visible cells, selecting, 147
 Visual Basic Editor (VBE), 759, 768, 807
 Toolbox controls, 808–810
 properties of, 809–810
 Visual Basic for Applications (VBA), 4, 5, 12, 234, 755–784. *See also* macros
 coding, 770–772. *See also* VBA programming
 entering and editing code, 771–772
 tips on, 772
 controlling execution in, 778–780
 For-Next loop, 779
 If-Then construct, 778–779
 Select Case construct, 780
 With-End With construct, 779–780
 functions. *See* VBA functions
 learning more about, 783–784
 objects and collections in, 775–776
 methods, 778
 properties of, 776–778
 subroutines, 757–758
 terminology of, 759
 uses of, 756–757
 variables in, 778
 workings of, 772–775
 VLOOKUP function, 225–226, 793
 volatile functions, 210
- W**
 walls of 3D chart, 367, 392
 WB1 files, 495
 WB2 files, 495
 Web discussions, 16
 Web Form Wizard, 859
 Web pages, 348
 activation from toolbar button, 746
 Web queries, 694–697
 Web site(s)
 defined, 686
 IDG Books, 689
 information from, 687
 Web subscription and notification, 16
 Web toolbar, 692–693
 WEEKDAY function, 222
 WEEKNUM function, 658
- what-if analyses, 11, 348, 595–615
 data tables, 599–607
 limitations of, 607
 one-input, 600–603
 two-input, 603–607
 example of, 595–596
 macro-assisted, 597–599
 manual, 597
 in reverse, 617
 Scenario Manager, 598, 600, 607–615
 defining scenarios, 608–610
 displaying scenarios, 611
 limitations of, 614–615
 merging scenarios, 612
 modifying scenarios, 611
 report generation, 612–614
- What's This? command, 58
 widths, column
 changing, 135–136
 pasting, 159
 window(s), 43–47
 active, 45
 closing, 47
 mouseless manipulation of, 47
 moving and resizing, 44–45
 sizes and positions, custom views of, 278
 states of, 43–44
 switching among, 45–47
 workbook, 24–27
 window control menu button, 23, 24
 Windows Clipboard, 150–151, 158, 497
 sharing data using, 666–668
 Windows Paint program, 323
 Windows Taskbar, 18, 44
 With-End With construct, 779–780
 Wizard, Chart, 287–288
 WK1 files, 494
 WK3 files, 494
 WK4 files, 494
 WKS files, 494
 WMF files, 322
 Word. *See* Microsoft Word
 word search puzzles, 723–724
 WordArt, 334–336
 drawing tips, 335–336
 example of, 335
 WordArt toolbar, 334
 WordPad, 671
 WordPerfect for Windows, 671
 workbook(s), 6, 43, 72
 active, 43
 attaching toolbars to, 735–736
 closing, 87–88
 creating new, 74–75
 dependent, 441
 documenting, 357
 finding lost, 79

continued

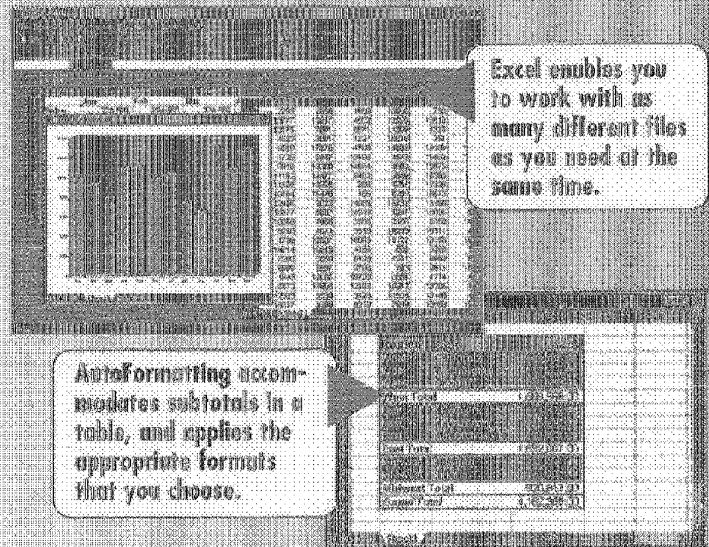
- workbook(s), *(continued)*
- linking, 441-448
 - changing link source, 446
 - examining links, 445
 - external reference formulas for, 442-446
 - reasons for, 441-442
 - to recover data from corrupted files, 449
 - severing links, 446
 - to unsaved workbook, 444
 - updating links, 446
 - mailing as e-mail attachment, 485-486
 - maintenance of, 358
 - merging styles from other, 261-262
 - multisheet, 86
 - opening existing, 75-80
 - automatic, 80
 - file display preferences, 77-78
 - filtering by file type, 77
 - specifying folder for, 77
 - Tools menu for, 78
 - protecting, 356
 - referring to cells in other, 177-178
 - routing, 486-488
 - saving, 37-38, 80-87
 - default location for, 82
 - file naming rules, 82
 - in HTML format, 690
 - in older formats, 86-87
 - options for, 83-84
 - summary information, 84-85
 - as text file, 265
 - shared, 82, 88, 481-484
 - advanced settings for, 483-484
 - appropriate sharing, 481
 - conflicting changes between users, 484
 - designating, 482-483
 - limitations of, 482
 - personal views, 484
 - tracking changes, 484
 - updating changes, 484
 - source, 441, 447
 - templates
 - custom, 747, 750-754
 - default, 747, 748-750
 - included in Excel, 751
 - operation of, 751
 - storing, 753
 - workbook window, 24-27
 - WORKDAY function, 658
 - workgroup application, 479-488
 - file reservations, 479-481
 - mailing workbook as e-mail attachment, 485-486
 - routing workbook, 486-488
 - shared workbooks, 82, 88, 481-484
 - advanced settings for, 483-484
 - appropriate sharing, 481
 - conflicting changes between users, 484
 - designating, 482-483
 - limitations of, 482
 - personal views, 484
 - tracking changes, 484
 - updating changes, 484
 - worksheet(s), 43, 121-137. *See also* cell(s); column(s); row(s)
 - activating, 122
 - adding new, 122-123
 - auditing and annotation of, 10
 - cell referencing outside, 177-178
 - changing name of, 123-124
 - consolidating, 448-455
 - data sources for, 454-455
 - linking worksheets and, 442
 - pivot tables for, 585-587
 - shared workbooks for, 481
 - by using Data ⇨ Consolidate, 452-454
 - by using formulas, 449-451
 - by using Paste Special, 451
 - copying, 125
 - copying ranges to, 154
 - default template, 747, 749-750
 - deleting, 123
 - error-free. *See* troubleshooting
 - formatting. *See* stylistic formatting
 - hiding/unhiding, 125
 - moving, 124-125
 - navigating through, 48-51
 - using keyboard, 49-50
 - using mouse, 50-51
 - objects embedded in, 679-681
 - outlining, 10
 - panes
 - freezing, 131-132
 - splitting, 130
 - pivot tables to consolidate, 585-587
 - protecting, 355-356
 - selecting entire, 145
 - shared, 641
 - size of, 49
 - VBA function in, 786-787
 - views of
 - multiple, 128-129
 - naming, 133
 - zooming, 125-128
 - worksheet controls, 12, 13
 - worksheet map, 707
 - worksheet outlines, 10, 429-440
 - adding data to, 439
 - applying styles to, 437-439
 - charts from, 440
 - creation of, 433-436
 - automatic, 434
 - data preparation, 433-434
 - manual, 434-436
 - displaying levels of, 437
 - example of, 429-432
 - hiding symbols of, 439
 - removing, 439
 - workspace files, 88
 - world countries, map of, 408, 423
 - World Wide Web (WWW), 685, 686. *See also* entries beginning with Web
 - WPG files, 322
 - WQ1 files, 495
 - WQ2 files, 495
 - Wrap Text formatting feature, 118
- X**
- X icon, 100, 101
 - XIRR function, 661
 - XISgalry.xls, 392
 - XLA files, 72, 868
 - XLB files, 72
 - XLC files, 72
 - XLL files, 72
 - XLM files, 72
 - XLM language, 11, 755
 - XLS files, 7, 72, 857
 - XIStart folder, 80
 - XLT files, 72
 - XLW files, 72
 - XNPV function, 661
 - XY (scatter) charts, 285, 308-309
 - XY-sketch, 726-727
- Y**
- Year 2000 issues, 98
 - YEAR function, 222
 - YEARFRAC function, 658
 - years, grouping by, 593
 - YIELD function, 661
 - YIELDDISC function, 661
 - YIELDMAT function, 661
- Z**
- zero, division by, 182
 - zip codes, plotting U.S., 424-425
 - Zoom tool box, 127
 - zooming, 418-419, 711-712
 - worksheet organization sensed from, 702
 - worksheets, 125-128
 - z-Test (Two-Sample Test for Means) tool, 656-657

If Excel can do it, you can do it too...

You, too, can excel -- especially with expert advice from one of the country's leading authorities on spreadsheet software. Whenever you get stuck or need to learn something you've never done before, turn to *Microsoft® Excel 2000 Bible*. With plenty of examples and little-known tips, John Walkenbach guides you step-by-step through the entire program -- from basic cell formatting to the exciting new Web capabilities of Excel 2000.

Inside, you'll find complete coverage of Excel 2000

- Simplify repetitive tasks using formulas and functions
- Illustrate your data with charts, drawings, maps, and other graphics
- Consolidate and juggle multiple worksheets in a single workbook
- Share documents with a workgroup across a corporate intranet or the Internet
- Save worksheets as HTML documents and publish them on your Web page
- Import data from other databases or Office applications using queries
- Analyze data with pivot tables or "what if" scenarios, Goal Seeking and Solver, or the Analysis ToolPak
- Automate Excel using templates, VBA macros, and custom worksheet functions and dialog boxes



Fully functional trial version of Power Utility Pack 97 on CD-ROM, with

- 31 utilities
- 40 custom worksheet functions
- Enhanced shortcut menus
- Games for Excel

Plus

- Adobe Acrobat Reader
- Electronic version of Internet Directory from *Internet Bible*

- PDF Version of the full book
- WinZip 7.0 Evaluation Version



MindSpring Internet Access and Netscape Communicator also included!

Shareware programs are fully functional, free trial versions of copyrighted programs. If you like particular programs, register with the publisher for a standard fee and receive licenses, enhanced versions, and technical support. Freeware programs are free, copyrighted games, applications, and utilities. You can copy them as many times as you like--free!--but they have no technical support.



www.hungryminds.com

System Requirements:
Windows 95/98/NT with
Pentium processor, 16MB RAM, Excel 2000

\$39.99 USA
\$56.99 Canada
£36.99 UK incl. VAT

Reader Level:
Beginning to Advanced

Shipping Category:
Excel 2000/Spreadsheets

1-552-5472-0



X000RR6SCB

Microsoft Excel 2000 Bible
Used, Very Good