

Bonus Chapter 1

Creating Excel Applications for Others

In This Chapter

- ▶ Developing spreadsheets for yourself and for other people
 - ▶ Knowing what makes a good spreadsheet application
 - ▶ Using guidelines for developing applications for others
-

Excel programmers develop two basic types of spreadsheets: spreadsheets that only they use and spreadsheets that other people use. This distinction often determines how much effort you need to put into creating a spreadsheet. Usually, developing spreadsheets for your use is much easier than developing spreadsheets that others will use.

In this chapter, I provide general guidelines for developing spreadsheets for someone other than yourself. But even if you're the only person who uses your spreadsheet creations, you might discover some helpful hints.

What's a Spreadsheet Application?

Excel *programming* is essentially the process of building applications that use a spreadsheet rather than a traditional programming language. In many cases, people other than the application developer use these applications.

My working definition of a *spreadsheet application* is this: A spreadsheet file (or group of related files) designed so that someone other than the developer can perform useful work without extensive training. Based on this definition, most of the spreadsheet files you've developed probably don't qualify as spreadsheet applications. You may have hundreds of spreadsheet files on your hard drive, but you probably didn't design most of them so that others can use them.



The qualities of a good application

Like witches, there are good spreadsheet applications and bad spreadsheet applications. How can you tell them apart? A good spreadsheet application does these things:

- ✓ Enables end users to perform a task they probably couldn't otherwise do.
- ✓ Provides an appropriate solution to a problem. The optimal approach for solving a problem doesn't always involve designing an application that works in a spreadsheet environment.
- ✓ Does what it's supposed to do. This might be an obvious prerequisite, but many applications fail to meet this test.
- ✓ Produces accurate results and is bug-free.
- ✓ Performs its job using appropriate, efficient methods and techniques.
- ✓ Traps errors and helps the user correct them.
- ✓ Does not allow the user to accidentally (or intentionally) delete or modify important components.

- ✓ Offers a clear, consistent user interface, so the user always knows how to proceed.
- ✓ Contains formulas, macros, and user interface elements that are well documented.
- ✓ Provides a design that enables developers to make simple modifications without making major structural changes.
- ✓ Presents an easily accessible Help system that offers useful information on at least the major procedures.
- ✓ Is based on a *portable* design — that is, the application runs on any system that has the proper software (in this case, a copy of Excel 2003 and possibly earlier versions).

You can create spreadsheet applications at many different levels, ranging from simple fill-in-the-blanks templates to extremely complex applications that use custom menus and dialog boxes — and may not even look like spreadsheets.



Throughout this chapter, I use the terms *developer* and *end users*. The developer is the person who creates and maintains the application (that's you!), and the end users are the folks who benefit from your efforts (this could include you).

Developing Applications, Step by Step

No simple recipe exists for developing a spreadsheet application. Besides, this isn't a cookbook. Fact is, everyone develops his or her own style for creating spreadsheet applications. In this section I provide you with some general guidelines that I find useful. At the very least, this information can help you improve your own development style.

Spreadsheet developers typically perform some of the following activities. You won't necessarily perform all these steps for every application, and the order in which you perform them may vary from project to project.

- ✓ Determine the user's needs.
- ✓ Plan an application that meets those needs.
- ✓ Determine the most appropriate user interface.
- ✓ Create the spreadsheet, formulas, macros, and user interface.
- ✓ Test and debug the application.
- ✓ Make the application bulletproof (prevent your app from being mangled).
- ✓ Make the application aesthetically appealing and intuitive.
- ✓ Document the development effort.
- ✓ Develop user documentation and online Help.
- ✓ Distribute the application to the user.
- ✓ Update the application when necessary.

I describe these activities in the following sections.

Determining user needs

The first step in developing an application usually involves identifying exactly what the end users require. Skipping this step (or guessing what the users *might* need) often results in additional work later.

In some cases you know the end users very well — you may be one yourself. In other cases (for example, a consultant developing projects for a client), you know little or nothing about the users or their situation.



These tips make this phase easier:

- ✓ Don't assume that you know what the users need. Second-guessing at this stage almost always causes problems later in development.
- ✓ If possible, talk directly to the application end users, not only to their supervisor or manager.
- ✓ Learn what, if anything, the users currently do to solve the problem. You may save some work by adapting an existing application.
- ✓ Identify the resources available at the users' site. For example, try to determine whether you must work around any hardware or software limitations.

- ✓ If possible, find out which systems will run your application. Consider whether your application must run on slower systems or on systems that aren't connected to a network.
- ✓ Understand the end users' skill levels. This information helps you design the application appropriately.
- ✓ Determine the anticipated lifespan of the application. This often influences the amount of effort you put into the project.

One final note: Don't be surprised if the project specifications change before you complete the application. This often happens, and you're better off *expecting* changes rather than being surprised by them.

Planning an application that meets user needs

After you determine the end users' needs, you might be tempted to jump right in and start fiddling around in Excel. Take it from someone who suffers from this problem: Try to restrain yourself. Builders don't construct a house without a set of blueprints, and you shouldn't develop a spreadsheet application without a plan.

How formal you make your plan depends on the project scope and your general working style. You should, however, spend at least some time thinking about what you need to do and come up with a plan of action. Take some time to consider the various ways you can approach the problem. A thorough knowledge of Excel pays off here by helping you avoid blind alleys.



More specifically, you need to consider some general options at this stage, including the following:

- ✓ **File structure:** Should you use one workbook with multiple sheets, several single-sheet workbooks, or a template file?
- ✓ **Data structure:** Should the application use external database files or store everything in worksheets?
- ✓ **Formulas or VBA:** Should formulas perform calculations or should you write VBA procedures? Both have advantages and disadvantages.
- ✓ **Add-in or XLS file:** In most cases, you probably want your final product to be an XLA add-in. Sometimes an XLS file is preferable.
- ✓ **Excel version:** Does your application need to work with older versions of Excel? If so, use such a version for your development work. (And you can't use any native Excel 2003 features.) If the application must work also with Excel for the Macintosh, test it using both products.

- ✔ **Error handling:** Anticipate possible errors and determine how your application will detect and deal with error conditions.
- ✔ **Special features:** Don't reinvent the wheel. For example, if your application needs to summarize lots of data, consider using Excel's built-in pivot table feature.
- ✔ **Performance issues:** Your approach ultimately determines your application's overall performance. Start thinking about the speed and efficiency of your application now. Don't wait until the application is complete and users are complaining about it.
- ✔ **Security level:** Excel provides several protection options for restricting access to particular workbook elements. Make your job easier by determining upfront exactly what you need to protect — and what level of protection is required.

You have to deal with many other project-dependent considerations in this phase. The important point is that you should consider all options and avoid settling on the first solution that comes to mind.



I've learned from experience that you shouldn't let the end user completely guide your approach to solving the problem. For example, suppose that you meet with a manager who tells you the department needs an application that writes text files, which will be imported into another application. Don't confuse the user's perceived need with the solution. In this example, the real need is to share data — using an intermediate text file is just one possible solution. In other words, don't let the users define their problem by stating it in terms of a solution approach. Finding the right approach is your job.

Determining the most appropriate user interface

When you develop spreadsheets that others will use, pay special attention to the user interface. By *user interface*, I mean the method by which the user interacts with the application: clicking buttons, using menus, pressing keys, accessing toolbars, and so on.

Excel provides several features that relate to user-interface design:

- ✔ Dialog box controls placed directly on a worksheet
- ✔ Custom dialog boxes (UserForms)
- ✔ Custom menus
- ✔ Custom toolbars
- ✔ Custom shortcut keys

Consider all your options, as well as the skill level and motivation of the end users. Then decide on the interface elements that make the most sense.

Developing the application

You've identified user needs, determined your approach, and decided on the user interface. Now you can get down to the nitty-gritty and start creating the application — the step that comprises most of the time spent on a project.

The approach you take when developing the application depends on your personal style and the nature of the application. Except for simple template-type applications, your application will probably use VBA macros.

I can't be more specific here, because each application is different. In general, try to keep your VBA procedures short and *modular*. In a modular application, each procedure performs *one* task. Limiting your procedures to a single task makes it much easier to make changes later on. For example, if you write a procedure that collects data from the user, formats the data, and creates a text file from that data, you probably should have created four procedures (three procedures to perform the tasks, and another procedure to call the other procedures).

Testing the application

Every computer user encounters software bugs. In most cases, such problems result from insufficient testing, which fails to catch all the bugs.



After you create your application, you need to test it. This step is crucial and you might spend as much time testing and debugging an application as you do creating the application in the first place. Actually, you should test extensively during the development phase. After all, while you write a VBA routine or create formulas in a worksheet, you want to make sure that the application works as it should.

Try to recruit one or more users to help with the testing. I've found that using a few good beta testers is an excellent way to uncover problems that I have overlooked.

Although you can't test for all possibilities, your macros should handle common types of errors. Some examples:

- ✓ What if the user enters a text string instead of a value?
- ✓ What if the user cancels a dialog box without making any selections?
- ✓ What happens if the user presses Ctrl+F6 and jumps to the next window?

As you gain experience, issues like these become second nature and you account for them with little effort.

Bulletproofing an application

A user can easily destroy a worksheet. Erasing one critical formula or value often causes errors that ripple through the entire worksheet — and perhaps in other dependent worksheets. Even worse, if the user saves the damaged workbook, the corrupt version replaces the good copy on disk. Unless the person using your application has a backup procedure in place, the user could be in trouble — and *you'll* probably be blamed!



Add some protection if other users, especially novices, use your worksheets. Excel provides several techniques for protecting worksheets and parts of worksheets. Table 1 reveals how you can do some of these things.

Table 1	Protecting Users
<i>What to Do</i>	<i>How to Do It</i>
Lock specific cells (using the Protection tab in the Format Cells dialog box) so that they can't be changed.	Doing so takes effect only when you protect the document with the Tools⇨Protection⇨Protect Sheet command.
Protect an entire workbook: the workbook structure, the window position and size, or all three.	This takes effect when you use the Tools⇨Protection⇨Protect Workbook command.
Hide the formulas in specific cells (using the Protection tab in the Format Cells dialog box) so other users can't see them.	This takes effect only when you protect the document with the Tools⇨Protection⇨Protect Sheet command.
Lock objects on the worksheet (using the Protection tab in the Format Object dialog box).	This takes effect only when you protect the document with the Tools⇨Protection⇨Protect Sheet command.
Hide rows.	This helps prevent the worksheet from looking cluttered and provides some protection against prying eyes. Format⇨Row⇨Hide
Hide columns.	This helps prevent the worksheet from looking cluttered and provides some protection against prying eyes. Format⇨Column⇨Hide

(continued)

Table 1 (continued)	
<i>What to Do</i>	<i>How to Do It</i>
Hide sheets.	This helps prevent the worksheet from looking cluttered and provides some protection against prying eyes. Format⇨Sheet⇨Hide
Hide documents.	This helps prevent the worksheet from looking cluttered and provides some protection against prying eyes. Window⇨Hide
Designate workbooks as read-only.	This ensures that they cannot be overwritten with any changes. Choose Save As⇨Options.
Assign a file password.	This prevents unauthorized users from opening your file. Choose Save As⇨Options.

Using an add-in, which doesn't allow the user to change anything on the add-in's worksheets, is another option.

The appropriate level of protection and how you implement it depend on the application and who will use it.



Excel's protection features are not really secure. In other words, if someone really wants to defeat your protection, they can probably find a way to do it.

Looking good

You've undoubtedly seen examples of poorly designed user interfaces, difficult-to-use programs, and just plain ugly screens. If you develop spreadsheets for other people, you should pay particular attention to how the application looks.

The way a computer program looks can make all the difference in the world to users. And so it goes with the applications you develop with Excel. End users appreciate a good-looking user interface. You can give your applications a much more polished and professional look if you devote some time to design and aesthetics.

Evaluating aesthetic qualities is very subjective. When in doubt, keep your worksheets simple and generic with these tips:

- ✔ **Strive for consistency.** This includes fonts, text sizes, and formatting. When designing custom dialog boxes, for example, try to emulate the look and feel of the Excel dialog boxes as much as possible.
- ✔ **Avoid the gaudy.** Just because Excel lets you work with 56 colors doesn't mean that you have to use them all. In general, use only a few colors and no more than two fonts. Better yet, stick to a single font and use bold formatting or larger font sizes for variation.
- ✔ **Keep it simple.** Developers often make the mistake of trying to cram too much information into a single screen or dialog box. Present only one or two chunks of information at a time.
- ✔ **Think modular.** Make it easy for the user to figure out what's what. For example, you can separate different parts of a worksheet by using background colors or borders.
- ✔ **Give users what they expect.** For example, most applications have a File menu. If you name your File menu something else, it can be confusing to users. Think about how programs that you use are consistent, not only internally, but with other programs.

Documenting your efforts

You can easily assemble a spreadsheet application. The hard part is making it understandable to other people. You must thoroughly document your work. Doing so helps you if you need to modify the application (and you will), and it helps anyone else who needs to work on the application (after you get that big promotion).



How do you document a workbook application? Store the information in a worksheet or use another file. Use a paper document if you prefer. Perhaps the easiest way to document a workbook application is to create a separate worksheet in which you store comments and key information about the project.

Use comments liberally throughout your VBA code. An elegant piece of VBA code may seem perfectly understandable to you today — but come back to it in a few months and you may be scratching your head.

Developing user documentation and Help files

In addition to your programming documentation, you need to develop user documentation. You have two basic choices: paper-based documentation or electronic documentation.

Help screens are standard fare in Windows applications. Fortunately, your Excel applications can provide tutelage — even in context. You can develop Help files and display a particular topic on demand.

Although developing Help files requires quite a bit of additional effort, the effort may be worthwhile for a large project. To simplify the process, I suggest that you acquire any of several software products designed for creating Windows Help files.

Distributing the application to the user

You've completed your project and you're ready to release it to the end users. How do you do this?

It depends entirely on the project and who will be using it. You could simply hand over a CD, scribble a few instructions, and be on your way. Or you may install the application yourself. Another option is to develop an official setup program that automatically installs your application. You can write such a program in a traditional programming language, purchase a generic setup program, or write your own setup program in VBA.

You also need to consider the issue of providing support for your application. In other words, who gets the phone call if the user encounters a problem? If you aren't prepared to handle routine questions, identify someone who is. In some cases, you may specify that the developer handles only highly technical problems or bug-related issues.

Updating the application when necessary

You're finished with your application after you distribute it, right? You can sit back, enjoy yourself, and try to forget about the problems you encountered (and solved) during development. In rare cases, yes, you may be finished. More often, however, your application's users will not be completely satisfied.

Sure, your application adheres to all original specifications, but things change. After seeing an application work, users often think of other things the application could be doing. That's right, I'm talking about updates. When you need to update or revise your application, you'll appreciate the fact that you designed it well the first time and fully documented your efforts. If not, well . . . you learn from your experiences.