ONE

Why Develop with Office and VBA

To some people, the idea of using Microsoft Office as a solutions development tool may sound a bit strange. On the one hand, Office is a set of business productivity applications, each designed to perform a specific task—Word processes text, Excel manipulates numbers, Outlook manages e-mail, and so on. On the other hand, solutions development means the creation of custom programs to fulfill specialized needs. The two seem incompatible—how can you use fixed, defined purpose programs to create custom solutions?

Not too long ago, the answer to this question would have been "You can't." Application programs did their specialized job and that was that. You might at best be able to do some minor customization using the program's built-in macro language, but that was about all. For sophisticated custom solutions you had to turn to a true programming language, such as C++ or Visual Basic.

Over the past few years things have changed—changed for the better, I might add. While Microsoft Office continues to provide a powerful set of dedicated-purpose applications programs, it now also gives you a flexible and robust platform for the development of custom business solutions. And that, in a nutshell, is the subject of this book.

In this chapter we will take a look at some of the advantages of Office development, then we will investigate how Office development works. Finally, we'll examine some of the features that are new with the latest version, Office 2000.

Advantages of Office Development

Any programmer who is faced with a Windows development project has quite a range of development tools available. The first step in the development process, and a very important one, is selecting the tool you will use. Make an inappropriate choice here, and a likely outcome is greater work and expense in developing the project, and a final product that is not as good as it might be. Office is not the best choice in all situations, but by understanding its many advantages you will know when it is the best choice of development tools.

Office Is a Familiar Environment

One of the main advantages of Office is that it is so widely used. Millions of people around the world are already using the Office applications in their day-to-day work. In computer lingo this is known as an *installed base*, and it means that you have millions of potential customers who have already decided they like Office and will therefore be more receptive to custom solutions that have been developed with Office and which therefore do not require the introduction of new technology. Also, since your end users are already familiar with the Office applications, at least to some extent, any custom solutions you create will run in a familiar environment. As a result, the training curve will be more shallow, training time and costs will be minimized, and need for support is also likely to be minimized.

Office Development Decreases Development Time and Expense

In any custom solution, much of the functionality you need is not really new—tasks such as finding files, displaying charts, performing financial calculations, and searching databases are already performed by the Office applications. By using Office as your development platform, all of this functionality is available to you in the form of Office components. The enormous amount of code writing, testing, and debugging that these components represent has already been done for you. Combining these components to create the finished application takes a small fraction of the time that would be required to do it from scratch. Furthermore, VBA code that you write can be saved and reused in other projects, providing additional efficiency. In today's competitive development environment, with ever more demanding clients, these advantages can make a big difference to your success as a developer.

Office Provides an Integrated Solution

All of the components that are available in Office were designed to work together, as they do within the individual Office applications programs. This means that a custom solution created using Office components will also have the resulting advantages of this tight integration. User interfaces will share a familiar look and feel, menu commands will have a similar structure, the familiar common dialog boxes will be used, and so on. Also, error handling—an important part of any Office application—will be consistent. In many cases, a custom Office solution does not appear to the user to be "custom"—it simply appears as a built-in extension of one of the Office applications programs.

Office Development Is Extensible

The components that are exposed by Office are not limited to use within Office, but are available for use with any development tool that supports the Windows COM (Component Object Model) specification. This includes such heavyweight stand-alone development tools as Visual Basic and Visual C++. What this means to you, the developer, is that in the unlikely event that your project's needs cannot be met within the Office development environment, you can go to a more flexible tool, such as Visual C++, to program the functionality you need while still retaining the advantages of using the Office components. This extensibility means that you need not avoid adopting Office as a development tool because of fears that someday it won't quite do what you need.

Not Always Your Best Choice

Despite Office's many advantages as a development tool, it is not always the tool of choice. If your client does not use the Windows platform, then obviously you cannot use Office. Even if your client does run Windows, he or she may have standardized on a different suite of applications programs, and trying to introduce Office is likely to be an unwelcome or at least tiresome approach. Also, remember that Office is designed for the sorts of tasks that are found in a typical business office environment. If your project's needs fall outside of this area—image processing, for example—then Office is not a good choice.

Software Components

Software developers are like anyone else in that they do not like to do unnecessary work. One way to avoid unnecessary work is to try not to do anything twice. For a programmer, this means that once you create the code or visual

interface to perform a specific task, you should not have to program that same task again because you should be able to use the original code or visual interface over and over. With this idea the concept of *software components* was born.

In their earliest form, software components consisted simply of sections of source code that were packaged in a way to make them easily reusable. If, for example, you wrote some code that calculates loan payments, you could put that code in a function library where it would be available to use in any other programs that need to calculate loan payments. This approach was called *structured programming* and for many years it was the most important paradigm used by software developers.

With the introduction of graphical user interfaces such as Windows, the old structured programming paradigm was found wanting. Now it was not enough to simply reuse code, developers also wanted to reuse the visual elements that are such an important part of most Windows applications. Once you had put in the effort to create a visual screen element, such as a box for the display and editing of text, you would of course want to be able to reuse that element in other programs. The first successful attempt at reusable visual elements was Microsoft's Visual Basic, which provided its own set of ready-to-use visual elements, called *controls*, as well as permitting programmers to create their own reusable visual elements. Visual Basic was a true revolution in programming, and the use of prepackaged visual components is a central part of all Windows development.

With the increased complexity of programming for a graphic user interface, however, a new paradigm was needed to extend the old structured programming approach. It is called *object-oriented programming* and it is at the heart of Office development.

Objects and Components

As programs became larger and more complex, the problem of program bugs and errors became increasingly difficult to deal with. Many of these problems are the result of unintended and unexpected interactions between different parts of a program, so by decreasing or eliminating these interactions you would go a long way toward preventing errors and bugs. This was accomplished by encapsulating a program's various functions into a number of independent modules, then combining the modules to create the final program. As you may have guessed, these modules are the objects in object-oriented programming.

One of the basic ideas behind objects is that what goes on inside the object is completely hidden from the rest of the program. This means that unintended interactions, with the resulting errors and bugs, are much less

likely to happen. Of course an object must interact with the rest of the program in some way or it would not be of any use, but with object-oriented programming the ways an object can interact are closely defined and controlled. The means by which the program can interact with an object is called the object's *interface* (not to be confused with visual interface, something different altogether).

We saw earlier how programmers want to package functionality into self-contained, reusable components in order to increase efficiency in the development process. Now we see that programmers also want to divide program functionality into independent objects in order to improve program reliability and to reduce errors and bugs. It turns out that these two goals are mutually supportive, and the object-oriented programming paradigm fulfills both the need for reusable software components and the need for programs constructed of independent modules.

Objects in Office

Given the powerful advantages of the object-oriented paradigm, it is not surprising that Microsoft would use this approach in creating the Office programs. When you are working in Word, Excel, or one of the other Office applications, just about everything you see or do involves objects. On the screen, each toolbar is an object, and each button on the toolbar is also an object. The document you are working on is an object, and each paragraph in that document is an object. The list goes on and on—there are literally hundreds of objects involved in Office—but you get the idea. One important thing to note from this description is that objects can contain other objects, as in a `Document` object containing one or more `Paragraph` objects. This sort of hierarchical arrangement is an important part of the Office object model.

Another important aspect of objects in Office is that as a developer you can create your own. That's right, you are not limited to the objects that are already part of Office but you can design custom objects to meet your development needs, with all the accompanying benefits of reusability and modularity. You may not need to create custom objects all that often, given the wide array of objects provided by Office, but it is a very powerful technique when you do need it.

Now that you understand the fundamentals of objects, we need to look at the three elements that make up an object's interface: properties, methods, and events.

Classes and Instances

You'll sometimes hear the terms *class* and *instance* used in relation to object-oriented programming—what exactly do they mean? A class can be thought of as the blueprint, or plan, for an object, while an instance is a single example of the object created from the class. As an analogy, suppose you have drawn up plans to build a table—those plans are the class. If you build a table from those plans, the table (or object) is an instance of the class. If you then build a second table, that is another instance of the class. There is only one plan (class) but you can create as many objects (tables) as you like.

Office works the same way. There is, for example, a single `Document` class. For each actual document being used, an instance of the class is created—this is a `Document` object. Likewise, there is a `Toolbar` class, with one instance created for each toolbar that is needed. Sometimes the term *object* is used when *class* would be technically correct, but as long as you understand the difference between the plan for an object (the class) and the object itself (an instance) you will not get confused.

Properties

Properties are the way in which objects store information. Sometimes a property represents user data, while other times it determines some characteristic of the object. For example, each cell in an Excel worksheet is represented by a `Range` object. The `Range` object's `Value` property specifies the text or number that is displayed in the cell, while the `FormatNumber` property controls the way the cell displays its data. Some properties are read/write, meaning that you can both determine and change the property. Other properties are read-only, in which case you can determine it but not change it. Some object properties refer to other objects, which is how Office's object hierarchy is constructed.

Methods

A method performs an action on an object. For example, the `Shape` object (which displays a geometric shape on-screen) has a `Flip` method that lets you flip the object either vertically or horizontally. Likewise, the `Document` object, representing a Word document, has the `PrintOut` method that prints the document. If properties are an object's nouns, then methods are its verbs.

Events

Many objects respond to events. Most events correspond to actions performed by the user, such as pressing a key, clicking with the mouse, opening a file, or changing data. Events can also be triggered by VBA code or by the computer operating system. You can write code that responds to events as needed,

making your custom application responsive to the user's input. For example, a `Form` object has an `Open` event that is triggered when the form is first displayed on the screen. You can write code to be executed whenever the `Open` event occurs to ensure that the form always displays centered on the screen regardless of the user's specific screen size.

The Office 2000 Suite

Before starting to develop custom solutions with office, it is a good idea to have some familiarity with the Office applications programs. There are two good reasons for this.

First, working with the Office applications can give you a good idea of what capabilities are available to you, the developer, in the various components that Office makes available. For example, knowing that Excel can calculate the average of a column of numbers, you can assume that this same capability is exposed in an Excel component and will be available to you for use in your custom projects. The same is true for Outlook being able to send an e-mail message, Word being able to underline text, and PowerPoint being able to display a bar chart. These may be trivial examples, but they make the point that you must thoroughly know your tools' capabilities before you can apply them effectively.

Second, a familiarity with the Office applications' built-in capabilities may save you some time and effort. The Office suite is very powerful, and the individual applications have some surprisingly sophisticated facilities. Some tasks may not require a custom project but rather can be accomplished by an application's native functionality. It is somewhat frustrating (not to mention embarrassing) to work at creating a custom solution only to learn that the same thing can be accomplished with a couple of menu commands!

Office 2000 Developer Edition

Office 2000 is sold in several editions, and you can use any edition for development purposes with the limitation that you'll have available only the components exposed by the installed applications. If your edition lacks Access, for example, you will not have its components available.

For serious Office development, however, I recommend Office 2000 Developer. This includes all the applications in Office 2000 Premium (the most complete edition) as well as some special tools for developers. This includes extra add-ins for the VBA editor, Visual Source Safe for source code management, and the Data Environment Designer. You also get the Microsoft Developer Network library, which contains complete reference information for Office development.

Here we'll take a quick look at the Office applications and the components they expose. Much of the remainder of the book will be devoted to exploring these components in detail.

Word

Word is designed for creating documents that will be printed or displayed as a Web page. As such it is organized around the concept of a *document*, which can be anything from a two-line memo to a 500-page multichapter report. Word's basic capabilities include entering and editing text, formatting the appearance of text and its arrangement on the page, and arranging text in columns and tables. Specialized elements, such as headers and footers, page numbers, indexes, and tables of contents are also supported. Word also provides a dictionary for checking spelling, a thesaurus for finding alternate word forms, and a grammar checker. If your custom application needs to work with text, you should look at the Word object framework first. The Word object model is covered in Chapter 5.

Excel

Excel is a spreadsheet program, intended for numerical manipulation and display. It is designed around the *worksheet*, which can be thought of as a sheet of digital paper ruled into rows and columns. At the intersection of each row and column is a *cell* where you can place text, a number, or a formula referring to other cells and/or perform calculations. Worksheets are organized into multipage workbooks. Excel includes a large number of predefined formulas that let you perform a variety of calculations, including many commonly needed statistical, financial, and scientific calculations. You can also write your own formulas as required. Excel also provides sophisticated charting capabilities that permit you to create a variety of chart types that automatically update when the worksheet data changes. For numerical analysis in your custom application, Excel is the place to look. The Excel object model is covered in Chapter 6.

Access

Access is a database management program. Of course, all the Office programs manage data in one form or another, but in the world of computers the term *database* refers specifically to data organized in a record and field format. An address book is a common example of a database, with each person representing a record and each piece of information (first name, last name, address, etc.) representing a field. Access provides tools for creating databases, adding and editing data, searching for specific information, and creating reports based on database data. Look to the Access object model for your

custom application's database manipulation needs. Note, however, that many database capabilities are also provided by the shared data access tools (see "Data Access" on page 12) which are not specifically part of Access but rather are a shared component of Office or Windows. You'll learn about the Access object model in Chapter 7.

PowerPoint

PowerPoint is a presentation program. Its primary purpose is to create "slide shows" for presentation at meetings, on the Web, and so on. Each page or "slide" in a presentation can combine text, charts, and graphical elements, including movies. Sounds can be associated with slides as well. The emphasis in PowerPoint is the creation of presentations that show data in a clear, easy to understand, and pleasant manner. As such, it has excellent tools for text formatting, drawing, page layout, and color control. Look in PowerPoint's object model when your custom application requires these capabilities. I'll cover the PowerPoint object model in Chapter 8.

Outlook

Rather than having a single focus like the other Office applications, Outlook is more of a personal assistant that takes care of a variety of tasks: e-mail, address book, calendar, to-do list, and the like. Its object model reflects this diversity, and you'll be turning to Outlook when your application requires sending or receiving e-mail, calendar functions, and so on. The Outlook object model is covered in Chapter 10.

FrontPage

FrontPage is specialized for creating and publishing Web content—in other words, pages that will be viewed on the World Wide Web. While other Office applications are Web-enabled, their capabilities are rather basic when compared to a specialized tool such as FrontPage. You will use FrontPage objects when your custom application needs to work with Web sites, which can be located either locally on an intranet or remotely on the Web. I'll deal with FrontPage objects in Chapter 9.

Shared Components

Office includes a significant number of components that do not belong to any specific application but are shared by two or more of them. Called shared components, these objects are also available to the developer of custom solutions. They provide, among other things, the ability to search for files, to manipulate Command Bars, to provide animated end-user help, to work with document properties, and to program scripts. Shared components are covered in Chapter 11.

Data Access

Strictly speaking, the data access components are shared because they do not belong to a specific single Office application. They are usually discussed separately from the other shared components, however, because of the great importance that data access has in most custom Office solutions. For the most part, data access in Office uses a strategy called Universal Data Access. There are two parts to this: OLE DB, which provides the low-level data access components, and ActiveX Data Objects, or ADO, which provides a high-level COM-compliant programming interface. These components are covered in Chapter 7.

Web Technologies

Including Internet Explorer version 5, Microsoft's latest Web browser, Office provides the developer with a number of Web-related components. These include:

- Web components, a set of ActiveX controls that let you publish fully interactive documents (spreadsheets, reports, databases, etc.) on the World Wide Web.
- Office Server Extensions can be used to create online threaded discussions.
- Script editor for programmatically working with the scripts and objects that make up an HTML page.
- Data access pages permit users to work interactively with a database on a Web page.

Web-related components are scattered throughout the Office model and are covered as needed throughout the book, in addition to having an entire chapter, Chapter 23, devoted to them.

What's New in Office 2000

Microsoft Office has always offered some degree of programmability, and the number of features available to developers has increased with each subsequent release. The latest incarnation, Office 2000, offers significant enhancements over earlier versions in areas that will be of interest to developers. If you have worked with Office 97, the previous version of Office, this section should be of interest to you. Space limitations preclude covering all of these new features in detail.

Web Integration

In recognition of the increasing importance of the Internet and the World Wide Web, Office now fully integrates its applications programs with the Web. You can publish information to the Web as easily as saving it to disk, which means that you can concentrate on the content of your documents without worrying about their format. HTML, the language of the Web, is now fully supported as an Office file format. These new Web-related technologies are available to Office developers as well as to users of the applications programs.

Virus Protection

Most everyone has heard of the new breed of computer viruses that are spread inside Office documents, commonly known as macro viruses. The recent Melissa virus was only one example. Given the widespread use of Office, and the fact that an Office document can contain executable VBA code, it seems an obvious target for virus creators. In previous versions, macro virus protection was limited to identifying Office documents that contained VBA code and giving the user the option of enabling the code or preventing it from executing. Now, Office documents can be scanned by third-party virus protection software to identify actual viruses. In addition, Microsoft's Authenticode technology permits the digital signing of software components. A user can then identify the source of a component, and open documents only if the contained components have been signed by trusted sources.

Add-In Architecture

An *add-in* is a supplemental program that adds additional capabilities to an Office application. For example, Excel comes with several add-ins that add sophisticated statistical analysis functions to your worksheets. From the user's perspective, the advantage of add-ins is that you can load only the functionality you need, and you can obtain highly specialized capabilities from third-party vendors in addition to those provided as part of Office. For the developer, the add-in architecture is one way to provide a custom solution, one that is tightly integrated with the host application.

Previous versions of Office supported add-ins, but only *application-specific* add-ins that could run in only a particular Office application, such as Word or Excel. The new add-in architecture also supports add-ins based on the Component Object Model, or COM, specification. This specification permits a single add-in to run in any Office application, which provides you with significant flexibility.

More Comprehensive Event Model

Events are a central part of Office development, and the new event model adds more than 20 new events, mostly associated with the Document and Windows objects in Word and PowerPoint. There are also several new events related to Command Bar objects that can be used in all Office applications. And, while previous versions of VBA let you create your own object, you can now define custom events and interfaces associated with those objects.

More Objects

I'll mention only two of the new objects in Office 2000. The `Dictionary` object opens up many possibilities. It is similar to a `Collection` object except that it can hold objects of different data types. File access and management is made easier with the `FileSystemObject`, which provides object-oriented access to the drives, folders, and files on your system.

New VBA Language Elements

VBA includes several new functions that provide additional power for parsing and manipulating strings and for formatting data. For example, the functions `FormatCurrency`, `FormatNumber`, `FormatDateTime`, and `FormatPercent` do just what their names imply. The `Split` function parses a string into substrings based on rules you provide, and the `Filter` and `Join` functions provide other welcome string handling abilities. The new VBA also permits functions to return arrays, and the direct assignment of one array to another.

Improved Data Access

Access to data is an important part of most custom Office solutions. Office supports Universal Data Access (which was described briefly earlier in the chapter) to permit an Office application to access data in essentially any format or location. You are no longer limited to accessing data that is located in recognized relational database file formats. OLE DB, the low-level part of Universal Data Access, is an open specification designed to access data in essentially any format, whether it be a legacy mainframe database, a list of e-mail messages, a graphical format, or what have you. The other part of Universal Data Access is ActiveX Data Objects (ADO), which provide a high-level programmable interface to OLE DB. If you are accustomed to using the older data access technologies, such as Open Database Connectivity (ODBC), Data Access Objects (DAO), or Remote Data Objects (RDO), I think you will be delighted at the power and ease of use of the new tools.

Custom Help Files

A complete Office application includes online help information for the end users. Windows Help has changed recently, and now uses an HTML-based system in place of the older Windows help file format. You can see this in the help files that are provided with the Office applications, but more to the point you can now create your own help files to distribute as part of your custom solution. Custom Help files are covered in Chapter 21.

Summing Up

After reading this chapter I think you'll agree that Office is indeed a powerful tool for developing custom Windows solutions. It is not the best choice in every situation, but I have found that it is a surprisingly versatile development tool that is applicable to a wide range of needs. What's even better, it is relatively easy to learn and to use.

The remaining three chapters in this section cover the fundamentals of Office development. We'll start with an essential topic: how to plan your Office project and how to approach visual interface design. Then we'll get right to work, creating a useful Office application which, while simple, is a great way to get your feet wet. You'll learn how to work with objects and their properties, methods, and events. You'll also get started with VBA and learn how to use the VBA editor, which is an essential part of Office development.