For years, everyone has been telling you to build your applications with reusable objects. Since Visual Basic 5, you've had the ability to build your own reusable objects. But how many programmers really take advantage of this capability? Yes, building applications with objects requires more thought than building applications without them. However, the results are usually worth the effort.

COM is Microsoft's standard for building reusable objects. These objects are independent of any particular programming language. That means that Visual Basic programs can easily access COM objects written in Visual C++ or Visual J++. It also means that you can write your own objects in Visual Basic, which can be used by Visual C++ programs. Of course, if you're like me, you'll forget about those other languages and simply use Visual Basic to create COM objects that can be used by other Visual Basic programs.

In this chapter, I'm going to review some background material about object-oriented programming in general and how it relates to COM. I'm also going to discuss how COM evolved from its early days into what Microsoft's marketing team refers to as COM and COM+.

## What Is COM?

COM stands for Component Object Model. It is a Microsoft specification that describes how to create reusable objects for programmers working in a Win32 programming environment. COM is a binary standard, which means that any programming language (with the proper facilities, of course) can create COM objects.

Because COM objects are binary, they can be contained in their own executable files. This makes it easy to develop objects that can be distributed independently of an application program file. In order to understand what COM is all about, you need to understand the meanings of the words *component* and *object*.

### What Is a Component?

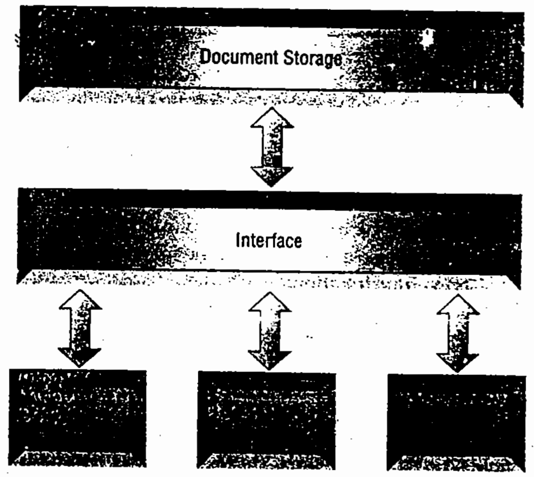
A *component* is an independent piece of code that may be shared with several different programs. When designing an application, you have the option to develop

one large complex program or to break it up into several smaller pieces. If you think this sounds like structured programming, you're right.

By breaking a hard-to-solve problem into multiple smaller problems, you've reduced the complexity of the problem considerably. You've also made it practical for different people to work on the program at the same time, because each person can be responsible for one or more components.

As an example, Figure 1.1 shows the components of a typical word processor. The Document Storage component is separate from the Spelling Checker, Document Display, and Document Printer components.

FIGURE 1.1: Components of a typical word processor



Because each component is isolated from the other components, it is important that you clearly define the boundaries of each component. At a minimum, this means that you describe how other programmers will access the component from their components. This is known as the component's *interface*. In the example shown in Figure 1.1, each of the components interacts with the Document Storage component through a well-defined interface. This interface lets programmers work independently while building their part of the application, and it also tells them how to access the information and perform functions in the other components.

## What Is an Object?

1

An *object* is a set of code that is designed to be reusable with a well-defined interface. Aside from the interface, there is no way to access any of the information in the object. This has the advantage of isolating the details of how the object stores information from the program that uses the object. You can change how the object stores its data without affecting how the calling program works.

**NOTE** A clever enough programmer can usually get around any limitations, even if it means accessing the real memory owned by an object. However, this isn't a good idea, since bypassing the object's interface could possibly corrupt the data managed by the object.

2

An object is actually more than just a set of code. It also represents something. It can represent a piece of information, a file, a set of data from the database, or a paragraph in a Word document—anything you choose. The interface defines a series of subroutines and functions that perform operations against the data contained by the object.

Figure 1.2 illustrates a simplified example of an object used to access a dictionary. This component has one interface—the Spelling Checker. The interface contains four functions:

- `Initialize` is called when the object is created.
- `Terminate` is called when the object is deleted.
- `WordIsSpelledOK` returns true when the word is found in the dictionary.
- `AddWordToDictionary` is used to add a word to the User Dictionary.

Internally, the component maintains three internal resources, which are used only inside the object:

- The Word Cache tracks frequently used words.
- The Master Dictionary contains all of the words supplied with the word processor.
- The User Dictionary is used to hold any words added to the dictionary by the user.

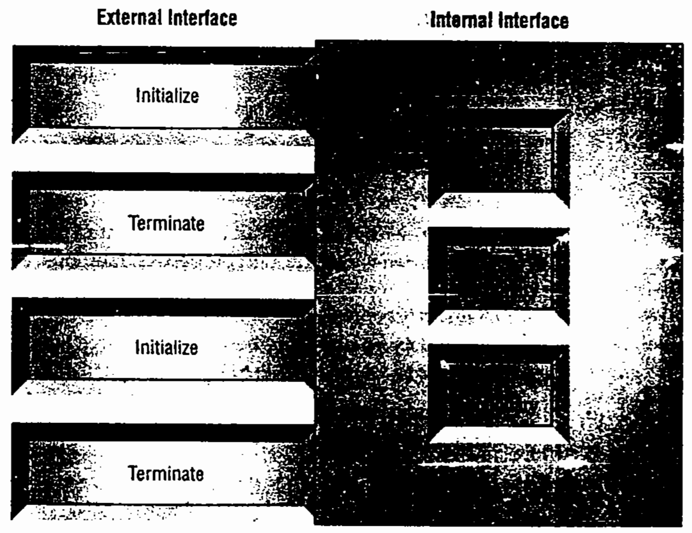


FIGURE 1.2: A simple object

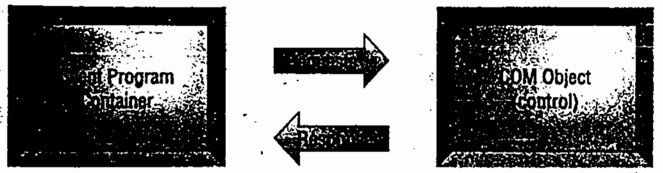
## What Is the Component Object Model?

The Component Object Model, or COM, combines the ability to create individual components with the ability to create reusable objects. This gives you a powerful tool under Microsoft Windows. COM allows you to produce binary object modules that are independent of any programming language and contain well-defined, object-oriented interfaces.

7

COM is based on a client/server model. Each COM object operates as a server that receives and processes requests from a client program, and generates responses. Figure 1.3 illustrates this concept. Notice that I don't describe how the requests and responses are handled. That handling is up to the COM object designer.

FIGURE 1.3:
A COM object and its client



As you know from using Visual Basic, a control can communicate with your program in three ways: through properties, methods, and events.

- A *property* represents a variable in your object, although it's often implemented as a pair of routines: one to retrieve the value from the property and another to assign a value to the property.
- A *method* is merely a function or subroutine that can be called from a Visual Basic program.
- An *event* is a subroutine whose parameters are defined in the control and that exists in your program. It is called by the control to inform your program of various situations. Its response will dictate the action the control will take.

These same three elements form the basis of how your program will interact with a COM object.

**NOTE** A COM client is sometimes called a *container*. A COM object is sometimes called a *control*.

## Why Is COM Important to You?

By now, you have a good idea why you should be using components and objects in your application. But since many languages, including Visual Basic, have the ability to include objects in source code, why should you use COM?

COM is the foundation of most things that Microsoft does. Without COM, Visual Basic wouldn't work, nor would Windows, SQL Server, or any number of

Microsoft products. The true test of Microsoft's confidence in COM is that its applications are based on COM technology. Word 2000 and Excel 2000 are based heavily on COM, as are many other Microsoft applications. So, as the saying goes, if it's good enough for Microsoft, it's good enough for me.

When you build your program using source-code-based objects, the objects are compiled into the object code. This has a couple of disadvantages:

- If you have multiple programs, each executable program has its own copy of the objects. This means that any time you change the objects, you must recompile all of the programs that use them.
- You must use the same programming language that the objects were written in to create your application programs. This restricts your choice of programming languages. Even though you may prefer to program in Visual Basic, sometimes you may find that using another programming language may help you.

Since COM objects are stored in their own independent object file, you can change the object without recompiling the applications that use it. You can also add new interfaces to the COM object to offer new features and capabilities. (Of course, if you modify the existing interfaces to the object, you may need to update the applications that use it.)

## From DDE to COM+

Many of the features you see in today's COM technology go back many years. COM draws its roots from a number of different technologies, including Dynamic Data Exchange (DDE), Visual Basic Extensions (VBX), and Object Linking and Embedding (OLE). An understanding of how COM has evolved over time will give you some insight into the way that COM works and why it works that way. This will prepare you for Chapter 3, where I dissect how COM works.

### DDE

In the early days of Windows, people used a clipboard much like they do today. Users could copy parts of a document in one program onto the clipboard and