Part I: Programming Access Applications

74029c01.qxd:WroxPro 9/27/07 1:43 PM Page 3

1

Overview of Programming for Access

In this chapter, you take a look at the different mechanisms available for programming for Access and what it means to program Access applications. You will see that with the ability to use the Access object model outside of Access, programming *for* Access is not necessarily the same as programming *in* Access. And, while the possibility is there to develop external applications that consume the Access object model, the primary focus of the book is scenarios that use the Access object model and Visual Basic for Applications (VBA) from within Access itself.

In this chapter, you:

- ❑ Learn about using managed code and how it can be used to work with Access applications
- ❑ Review object models that are often used with Access
- ❑ Learn about off-the-shelf applications in the context of Access development and what it means to develop off-the-shelf applications

Writing Code for Access

As a part of the overall Microsoft Office family of products, Access finds itself in an interesting position. It provides many tools that Access developers and programmers at all levels use to create robust database applications for themselves or their users. Access 2007 includes a database engine for storage, forms and reports for presenting data, macro objects for automating simple tasks, and a full-featured programming model using VBA. Collectively, these components make it possible for you, as the programmer or developer, to create rich solutions that can be easily deployed to your users.

As fun as it is to write Access applications, Access doesn't provide everything. For example, it doesn't provide an easy way to create e-mail messages with an arbitrary attachment such as Outlook. It doesn't provide statistical functions, such as Excel, and it doesn't provide word processing functionality, such as Word. It can, however, interact with these applications, as a part of the overall Office family, using each application's respective object model to participate in the larger solution. Chapter 10 provides insight into how Access fits into the larger picture when creating a solution.

In many cases, writing code for Access is different from writing code for other Office applications, such as Word and Excel. Many Access solutions are designed specifically to work with multiple Office applications. In our day-to-day work, we use applications that we have written to enable us to send custom e-mail messages using Outlook or to create a Word document with very specific formatting. In addition, when you develop a database in Access, often you are developing a full-featured application for use by multiple users.

The Access Object Model

The code you write for Access forms, reports, and controls targets the Access object model, but to retrieve data you need to use a data access technology such as Data Access Objects (DAO), or ActiveX Data Objects (ADO). The requirement for using two object models creates an interesting dichotomy when you're writing applications for Access, which also sets Access apart from writing code for other Office applications.

The result of this separation makes it possible to write code for Access applications that can easily be reused between applications. You might be thinking, "Can't I write modular code for Excel-based applications as well?" Of course. But we think you're more likely to do so when writing an Access-based application. From the pure coding perspective, we are not suggesting that you should always separate presentation code from data access code (in different files), but rather that it's possible. However, if you think about it, we do frequently separate the presentation layer from the data access layer when we create an application with linked tables.

The DAO Object Model

Data Access Objects, or DAO, has long been used as the native data access technology for Access. Originally included in the Jet database engine with previous versions of Access, new features appear in DAO for use with Access 2007. You learn more about these new features, and other features of DAO in Chapter 7.

The ADO Object Model

ActiveX Data Objects, or ADO, is another data access technology available to use with Access. Both technologies are acceptable and can be used in conjunction with one another, although it's probably not necessary to do so. DAO, as the native application programming interface (API) for the Access database engine, has performance benefits over ADO. That said, ADO is more generic and thus has its own benefits. For example, ADO provides the ability to create `Recordset` objects that are not bound to a table or query, but rather are created at runtime by appending fields. In addition, you can use an ADO recordset as the data source for a form, combo box, or list box.

Object-Oriented Thinking

Object-oriented programming (OOP) organizes programming tasks into *classes*. A class is a blueprint of the thing being modeled. Often, this is something in the real world such as a customer, or an employee. Other

74029c01.qxd:WroxPro 9/27/07 1:43 PM Page 5

times, it may be something more abstract such as a log file, or a dialog box. Classes are said to be self-describing because they describe the characteristics of an entity, and its behaviors — that is, what it can do. The characteristics of a class are known as properties, and the behaviors of a class are known as methods.

A class is different from an *object*. An object is a unique instance of a class. Over the years, language changes to Visual Basic, and subsequently VBA, have provided some of the OOP features long used by C++ programmers to VB and VBA developers.

Making the decision to use classes often represents a different way of thinking from traditional procedural programming. Classes play an important role in some of the concepts discussed throughout the book. As a result, we discuss programming classes and class design in greater detail in Chapter 3. Classes form the basis of several other pieces of functionality that follow in this book.

Windows API Programming

For those tasks that VBA or the Access object model does not provide, you can use the Windows API. An API, or application programming interface, is a set of functions grouped together by technology. The Windows API contains the core set of functions for Windows itself, but there are other APIs as well. For example, the DAO API consists of the objects, properties, and methods that make up DAO. The API is discussed in depth in Chapter 2.

Working with Managed Code

At the beginning of this chapter, we mentioned *managed code* as an alternative for working with the Access object model. Managed code refers to code whose memory is managed by the Common Language Runtime (CLR) of the .NET Framework. This often refers to code written in Visual C#, or Visual Basic .NET, although this is not a requirement. For example, Visual C++ developers can use C++/CLI available with Visual Studio 2005 to write managed code.

The focus of this book is writing solutions that utilize Access for its strengths and for building high-quality applications that are based on Access. Sometimes, this means writing code using other object models such as Outlook, or Excel, and integrating them as part of an overall solution. Other times, it means writing code in a different language altogether. Several chapters in this book provide examples using C# to either drive the Access object model, or as a library that you call from VBA code inside Access. The managed code samples that are available for download with this book include both C# and VB.NET.

A great amount of material has been written about C# ranging from syntax to constructs to design. Therefore, we won't spend much time explaining the language. We don't expect you to be a C# whiz, but if you are that's great! If not, no worries. We explain the code as we go so the managed code solutions provided are straightforward and understandable. Although the managed code in this book is written using C#, we have provided the VB.NET equivalents on the corresponding Web site for this book.

Because we use managed code in this book, in addition to VBA, let's spend some time discussing managed code in a little more detail.

What Is Managed Code?

As mentioned, managed code refers to any code that is written where memory is managed by the CLR. For the purposes of this book, we write managed code in C# — although you could just as well write in VB.NET. We chose C# because we use it in our daily work in testing Access and find it to be a nice language for many programming tasks.

The portion of the CLR that manages memory is known as the *garbage collector*. The garbage collector is responsible for detecting when objects are no longer needed, and for disposing of them appropriately. If you have written code in other languages such as C or C++, you may quickly recognize this as a powerful feature, although many C and C++ developers may prefer to manage memory themselves. If you fall into this camp, the garbage collector in the CLR provides features for tighter control when objects are cleaned up. Although VBA does not have built-in garbage collection, you are, in effect, managing your own memory in VBA when you set an object to `Nothing` such as:

```
Set objMyObject = Nothing
```

The opposite of managed code of course is *unmanaged code*. Unmanaged code is code whose memory is not managed by a runtime such as the .NET Runtime. This includes code written in C, C++, Visual Basic, or even VBA.

For more information about managed code, please refer to Appendix C of the Access 2007 VBA Programmer's Reference, ISBN 978-047004703.

Versions of Visual Studio

Managed code and the .NET Runtime were first available with Visual Studio 2002, which included version 1.0 of the .NET Framework. Version 1.1 of the .NET Framework was released with Visual Studio 1.1. The most recent release of Visual Studio, Visual Studio 2005 includes version 2.0 of the .NET Framework. For users running Windows XP or Windows 2003 Server, you can download either version of the .NET Framework from the Microsoft Web site. If you are running Windows Vista, version 3.0 of the .NET Framework is now included with the operating system and offers new libraries that further enhance managed code development.

You can even use one of the Visual Studio Express Editions to write managed code using either VB.NET or C#. However, be cautious — the Express Editions of the languages do not include the tools required to create a type library for use with Access and so you have to use a command-line tool that is included with Express Edition. We talk more about this tool in the next section.

Writing Managed Code Libraries to Use with Access

In addition to features such as garbage collection, the .NET Framework provides many libraries that you can use in managed code. Collectively, these libraries are known as the Base Class Library (BCL), and include functionality that is typically found in the Windows API. Because these libraries are available to you when writing managed code, they are nice to use in a type library that you call from VBA code in Access.

A library you create in a managed language such as C# creates a dynamic link library (DLL) file. And, while you can create references to DLL files from Access and VBA, you cannot reference a managed DLL directly. In order to set a reference to a DLL from VBA, it must expose the necessary COM interfaces that

74029c01.qxd:WroxPro 9/27/07 1:43 PM Page 7

are used to provide type information about a class. Managed DLL files do not include these interfaces. Therefore, in order to create a reference to a managed code library, you must first create a type library for the DLL. You can use Visual Studio to create the type library or a command-line tool called `tlbexp.exe`. This tool is available as part of the Visual Studio SDK, which is freely available on the Microsoft Web site.

So, you're probably thinking, why would we include an additional reference in the application when we can just use the Windows API? Good question. Managed code makes it pretty straight forward to write complex libraries that would have required a fair amount of API code to implement in VBA. Examples of this include:

- ❑ Using Windows common dialog boxes
- ❑ Working with the Windows Registry
- ❑ Writing to the Windows event log
- ❑ Retrieving the version number of a file

Writing managed code libraries to use in an Access solution has its benefits — namely, it's easier to write and deploy. However, it is not without its drawbacks. Installing a library written in any language requires an additional file as a part of the installation. However, we feel that the benefits of using managed code (when necessary) as part of a solution outweighs the drawbacks.

Referencing the Access Object Model from Managed Code

In addition to creating managed code libraries that you can call from VBA, you can write managed code that drives the Access object model itself. For example, you might write a report manager solution that bundles the reports in an Access application together to print multiple reports based on a timer. Such an application could use the Access object model to get a list of reports in the database and to print them.

You can add two types of references to a managed application — .NET and COM. Because Access is a COM-based application, the reference to the Access object model is a COM reference. If you add a reference to the Microsoft Access 12.0 object library in Visual Studio, you'll notice that a few additional references are given to you. These additional references are `ADODB`, `DAO`, `Microsoft.Office.Core`, and `VBIDE`. These references are all used somewhere in the Access object model and as a result are included automatically when you add the Access object model from your managed code.

In Chapter 14, you see how to create a managed application using C# to create a build of an Access application including features such as version and build numbers and release dates.

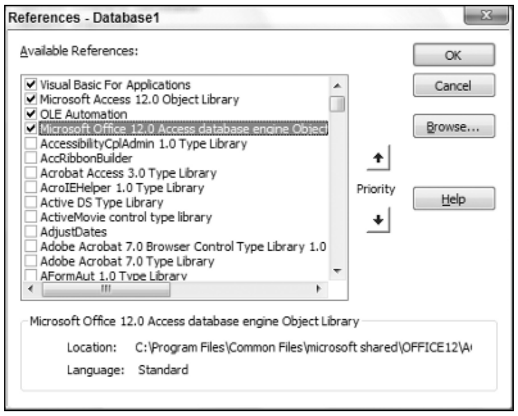
Referencing Other Applications

In order to use object models provided by applications such as Outlook or Project, you typically add a reference. However, adding references creates dependencies in an application that may not be desired. For instance, what happens if you add a reference to the Outlook 12.0 object library but your users are using Outlook 2002 (10.0)?

Issues such as these can be avoided using a technique known as *late-binding*. Late-binding enables you to write code without providing type information about an object. And while it also enables you to trap for error conditions at runtime instead of compile time, it tends to lead to code that is slower to execute. This lag is negligible on today's fast processors with a decent amount of memory. The opposite of late-binding is *early-binding*. Early-binding provides benefits such as compile-time checking and performance improvements. This sounds pretty nice, but it causes problems if your users do not have the same applications installed or the same version. For these times, late-binding becomes very useful.

Discovering References

The easiest way to find a reference to use in your Access application is to view the References dialog box from the Visual Basic Editor, as shown in Figure 1-1.



Available References:

- ✓ Visual Basic For Applications
- ✓ Microsoft Access 12.0 Object Library
- ✓ OLE Automation
- ✓ Microsoft Office 12.0 Access database engine Object
- AccessibilityCplAdmin 1.0 Type Library
- AccRibbonBuilder
- Acrobat Access 3.0 Type Library
- AcroIEHelper 1.0 Type Library
- Active DS Type Library
- ActiveMovie control type library
- AdjustDates
- Adobe Acrobat 7.0 Browser Control Type Library 1.0
- Adobe Acrobat 7.0 Type Library
- AFormAut 1.0 Type Library

Microsoft Office 12.0 Access database engine Object Library

Location: C:\Program Files\Common Files\microsoft shared\OFFICE12\A

Language: Standard

Figure 1-1

You can add references to COM objects that reside in DLL, EXE, or OCX files, or even to other databases that you create in Access. By adding references to other databases you can create reusable libraries of your own written in VBA. We discuss using references to databases in more detail in Chapter 3, and the issue of updating references in Chapter 14.

If you've opened the References dialog box, shown in Figure 1-1, and scrolled through the list, you'll quickly notice that there are a lot of files listed. Because there are probably more files listed on your development machine than those of your users, can you imagine the headache of trying to manage multiple references in an application? With all of these references to choose from, how do you know what you can or should use?

Adding References to Office Applications

The applications in the Office family of products are designed to work well together. As such, it's very common to find Access applications that include references to Office, Outlook, Excel, and the like. You can add a reference to other Office applications in the References dialog box. Adding a reference uses early-binding to an external application.

To use late-binding to another application, you must know its programmatic identifier, or *ProgID*. The ProgID is a string that identifies an application for use by another application. The following table provides the ProgID values for Office applications:

Application	**ProgID**
Access	Access.Application
Excel	Excel.Application
InfoPath	InfoPath.Application
OneNote	OneNote.Application
Outlook	Outlook.Application
PowerPoint	PowerPoint.Application
Project	Project.Application
Publisher	Publisher.Application
Visio	Visio.Application
Word	Word.Application

Summary

In this chapter, we provided some thoughts about what it means to write an Access-based solution. We examined how writing applications for Access is quite different from writing VBA code in other Office applications. Because Access is used to develop applications, sometimes off-the-shelf, it is often viewed as a development tool when compared to other applications in Office such as Word and Excel.

To support this view of Access as a development tool, we discussed the following:

- ❑ Object models commonly used while writing Access applications
- ❑ Programming tools and techniques such as the Windows API and object-oriented programming
- ❑ Using managed code that targets the Access object model, as well as using managed code as a library inside of an Access application

You also saw a glimpse into what lies ahead in the rest of this book. Features that are often found in commercial applications such as configuration, deployment, and help also have their place in Access applications. More important, you can standardize them across many applications. Over the course of the book we introduce code that you can use to integrate features such as these into your own applications.

Coming up next, you learn how to use the Windows API and why it can be useful in your applications.