Chapter 1

Where VBA Fits In

In This Chapter

- Describing Access
- Discovering VBA
- Seeing where VBA lurks
- Understanding how VBA works

COPYRIGHTED MATERIAL

This book is about using *Visual Basic for Applications (VBA),* which is a programming language that helps you program, tweak, and squeeze productivity from Access. VBA, which is embedded in Access, is a sophisticated set of programming tools that you can use to harness the power of a packaged application like Access. Just like you need to know how to walk before you can run, you need to know how to use Access before you can start to use Access VBA.

Maybe you want to use Access to manage a large mailing list. Maybe you need Access to manage your whole business, including customers, products, and orders. Perhaps you need to manage enrollments in courses or events. Whatever your reason for using Access, your first step is always to create the tables for storing your data. From there, you can then create queries, forms, reports, and macros to help manage that data. All these steps take place before you even get into VBA, so in this book we have to assume that you're already an experienced Access user who needs more than what queries, forms, reports, and macros can provide. If you're new to Access, this book isn't a good place to start. If you need to brush up on Access, *Access 2007 For Dummies* (by John Kaufeld, Laurie Ulrich Fuller, and Ken Cook; Wiley Publishing) or *Access 2007 All-in-One Desk Reference For Dummies* (Alan Simpson, Margaret Levine Young, and Alison Barrows; Wiley) is a good place to start.

Although Access has progressed through many versions over the years, VBA has remained relatively unchanged. We used Access 2007 to create this book, but the code examples we present should work fine in just about any version of Access. So now, before launching into VBA, take a moment to delve into what tables, queries, forms, and reports are all about, and how VBA fits into the overall scheme of things.

Taking a Look at Access

Access, part of the Microsoft Office suite, is a huge database management system that you work with by using modern object-oriented methods. (The term *object-oriented* stems from the fact that everything you create in Access — a table, form, report, or whatever — is considered an object.

The Access Navigation pane, as shown in Figure 1-1, is the main container in which you store all the main objects that make up a single database. The Navigation pane breaks down the objects into groups — tables, queries, forms, and so on — and each group contains the objects within that group. The following list summarizes the types of objects.

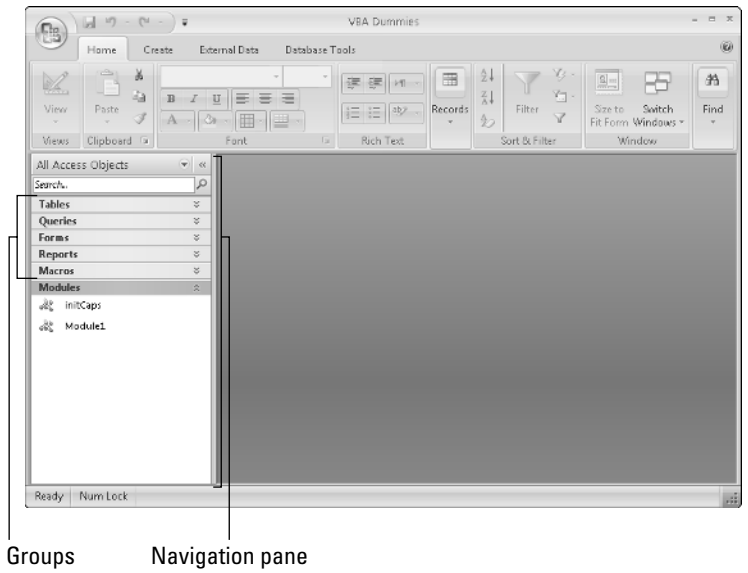

Figure 1-1: The Access Navigation pane.

- **Tables:** *Tables* contain the raw data that all other object types display and manage. Data in tables is stored in *records* (rows) and *fields* (columns).
- **Queries:** Use *queries* to sort and filter data from one or more tables.
- **Forms:** Access *forms* are similar to printed fill-in-the-blank forms, but they allow you to view and change data stored in Access tables.
- **Reports:** *Reports* define how data should be presented on printed pages.
- **Macros:** *Macros* provide a means of automating certain aspects of Access without programming in VBA.
- **Modules:** The *Modules* group, as you soon discover, is one of the places where you store VBA code. If you're not already familiar with modules, that's fine. Modules are what this book is really all about.

One of the most important things to understand is that you don't use VBA "instead of" other objects, like tables and forms. You use VBA to *enhance* the capabilities of other object types. Therefore, it makes no sense to even try VBA until you have a firm grasp of the purpose and capabilities of those other object types in Access.

Understanding VBA

Visual Basic is a programming language — a language for writing instructions that a computer can read and process. VBA is a programming language that's specifically designed to work with the application programs in Microsoft Office including Word, Excel, Outlook, and, of course, Access.

When you write text in a programming language (as opposed to writing in plain English), you're writing *code.* Programmers use the term *code* to refer to anything that's written in a computer programming language. For example, Figure 1-2 shows some sample VBA code. The whole trick to mastering VBA is finding out what all the various words in the language mean so that you can write code that tells Access exactly how to perform a task.

```
Public Function PCase(anyText)
    'Custom Access VBA function to fix all uppercase letters.

    PCase = StrConv(anyText, vbProperCase)

    If Left(PCase, 4) = "P.o." Then
        PCase = "P.O." & Mid(PCase, 5)
    End If

    If Left(PCase, 2) = "Mc" Then
        PCase = "Mc" & UCase(Mid(PCase, 3, 1)) & Mid(PCase, 4)
    End If

    If Left(PCase, 3) = "Mac" Then
        PCase = "Mac" & UCase(Mid(PCase, 4, 1)) & Mid(PCase, 5)
    End If

End Function
```

Figure 1-2: Some sample VBA code.

If the sample code shown in Figure 1-2 looks like meaningless gibberish to you, don't worry about it. People aren't born knowing how to read and write VBA code. Programming (writing code) is a skill you have to learn. For now, it's sufficient just to know what code looks like. Knowing what the code means is one of the skills you master in this book.

Because VBA code looks like a bunch of meaningless gibberish typed on a sheet of paper, it begs the question of why anybody would want to figure out how to read and write a dreadful language like that one. The answer to that question lies in the role that VBA plays in an application like an Access database.

Do, not die

Think of the term *execute* in the sense of "to carry out," as when you execute a U-turn or execute a procedure. Don't think of *execute* in the sense of "terminate the life of."

Access does indeed have a ton of tools that let you create a database without any programming. You could easily spend months or years just finding all the things you can do in Access without writing any VBA code. Yet despite the huge number of things you can do without programming, sometimes you want your database to accomplish a task that's not built into Access. That's where VBA comes in. When you want Access to perform a task that it doesn't already know how to perform, you write the steps to be performed in the VBA programming language.

When you're writing VBA code or just looking at some VBA code written by someone else, Access doesn't do anything. Access doesn't start performing the steps described by that code until Access executes the code. When you write VBA code, you're writing a set of instructions that Access can perform at any time, over and over again.

The ability to use the same code over and over again is the key to automating mundane tasks in Access. For example, if you use Access to print checks, you might have to manually type the part of the check where you spell out the amount, like "Ninety-two and 99/100 Dollars" for $92.99 because Access can't make that translation on its own. But if you could write some code to translate a number like $92.99 into words, you wouldn't need to type all those dollar amounts. Access would just print the correct information as it prints each check.

Seeing Where VBA Lurks

In an Access database, VBA code is stored in modules. Despite its fancy name, a *module* is basically an electronic sheet of paper on which VBA code is typed. A module in Access is either of these two types:

- **Standard:** A page that contains VBA code that's accessible to all objects in the database. A standard module always exists in the Modules group in the Navigation pane.
- **Class:** A page of VBA code that's attached to every form and report you create. You can also create a class module that appears in the Navigation pane.

The main difference between a standard module and a class module is that you can create an instance of your class module in code. A standard module contains procedures you can run from anywhere in your database. A class module contains code that's either attached to an existing form or report or is its own entity in the Navigation pane.

We talk about the types of modules as they become relevant throughout this book. Right now, they're not terribly important. For now, the main thing to keep in mind is that modules contain VBA code. Now take a look at where modules are stored within an Access database.

Finding standard modules

A *standard module* contains VBA code that's accessible to every table, query, form, report, page, and macro within the current database. Like those other objects, a standard module always gets its own group in the Navigation pane (refer to Figure 1-1). When you open the Modules group, the list shows the names of modules (if any) within the current database, as shown in the example in Figure 1-3. This example contains standard modules and class modules.

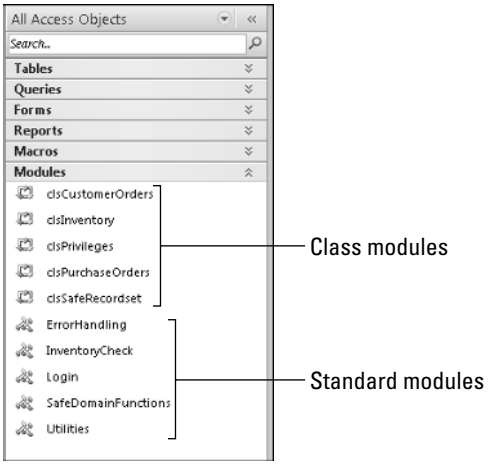

Figure 1-3: Standard and class modules in a database.

Don't be surprised if you open the Modules group in a database and the group is empty. These modules don't just happen: You have to create them.

Finding class modules

Like standard modules, *class modules* contain VBA code that tells Access what to do. Unlike standard modules, however, not all class modules are

found in the Navigation pane. Class modules are often hidden behind forms and reports in your database. You can also create a class module that appears in the Navigation pane, as shown in Figure 1-3.

It might help to define the term *class* as *a class of objects.* In Access, tables are one class of objects, queries are another class, forms are another class, and reports are another, for example. Or, looking at it from the other direction, a single form is an object within your database. That single form is also a member of the class of objects known as *forms.*

We think that it helps to envision a form or report's class module as literally being hidden behind its form or report, as illustrated in Figure 1-4. This type of class module might be hidden from you if you don't know how to find it.

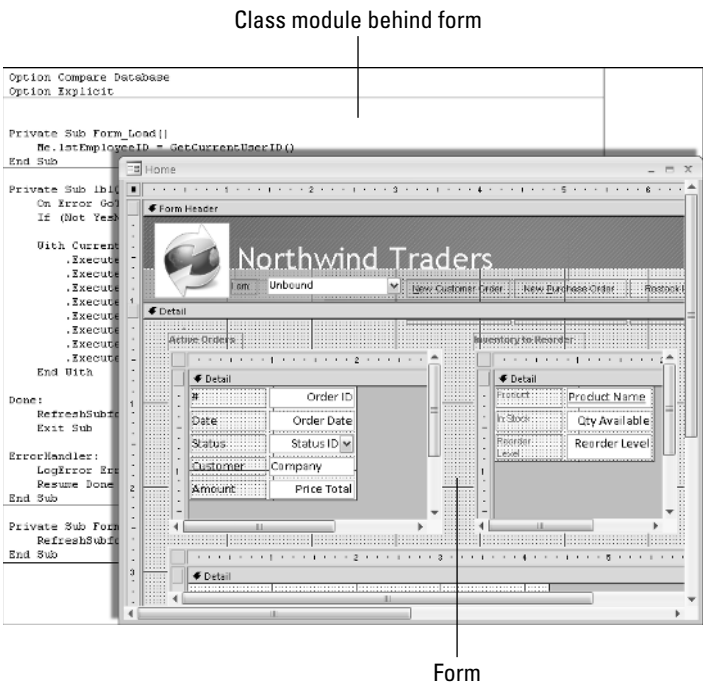

Figure 1-4: Class modules typically hide behind forms and reports.

You have several ways to get to a form's or report's class module, as you discover in upcoming chapters. For now, if you just want to open a class module and have a look, here's one way to do it:

1. **In the Navigation pane, open the Forms group or Reports group, depending on which type of object you want to open.**

2. **Right-click the name of any form or report and choose Design View.**
3. **To see the class module for the open form or report, click the (Form Design Tools) Design tab, and then click the View Code command in the Tools group (see Figure 1-5).**

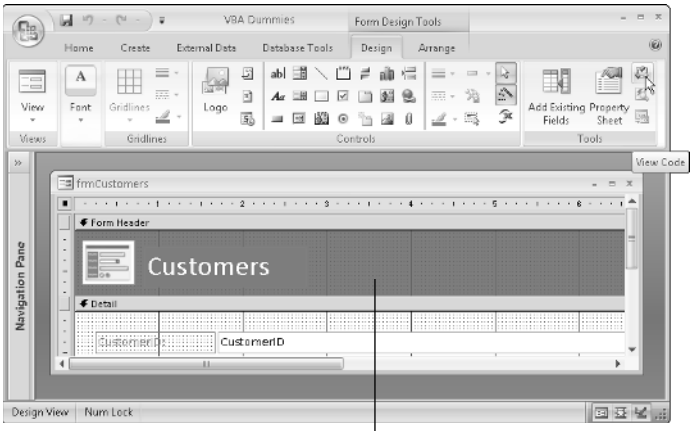

Figure 1-5: Class modules are accessible from form and report Design views.

From VBA to Access

When you open a module, whether it's a standard module or a class module, your screen changes radically. That's because the module opens in the *Visual Basic Editor,* which is a separate program window from Access. In fact, if you look at the taskbar, you still see a taskbar button for Access and another for the Visual Basic Editor. You can switch back and forth between Access and the editor just by clicking their respective taskbar buttons, as shown in Figure 1-6.

Alternatively, you can press Alt+F11 to switch back and forth between Access and the VBA Editor at any time.

If the module you open contains any VBA code, that code is visible in the Code window, also shown in Figure 1-6. If you upgraded a database from a previous version of Access, a class module might contain VBA code, even if you never wrote a line of VBA code in your life, because some of the control wizards in Access 2003 and earlier automatically wrote VBA code for you behind the scenes. In Access 2007, the wizards create embedded macros, which is a new feature that we don't cover in this book.

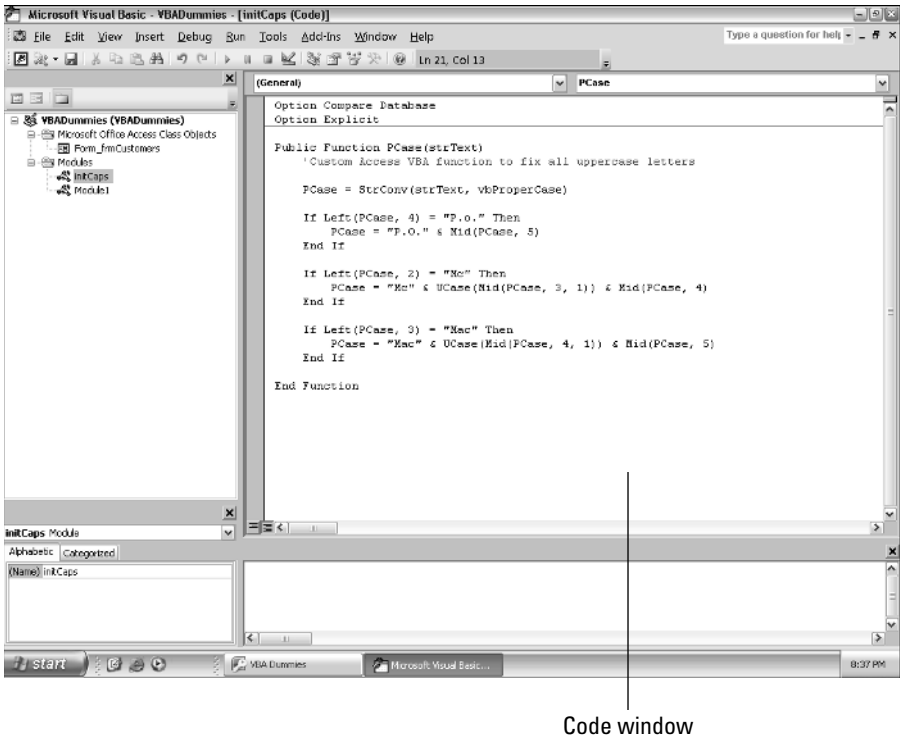

Figure 1-6: Use the taskbar to switch between the Visual Basic Editor window (shown) and the Access window (hidden).

The main thing to keep in mind here is that every time you open a module, you end up in the Visual Basic Editor. You discover how to use that program in upcoming chapters. For now, the most important thing to know is how to close the editor and get back to the more familiar Access program window. Here are two easy ways to close the Visual Basic Editor and get back to the Access program window:

- Choose File⇨Close and Return to Microsoft Office Access (see Figure 1-7).
- Press Alt+Q.

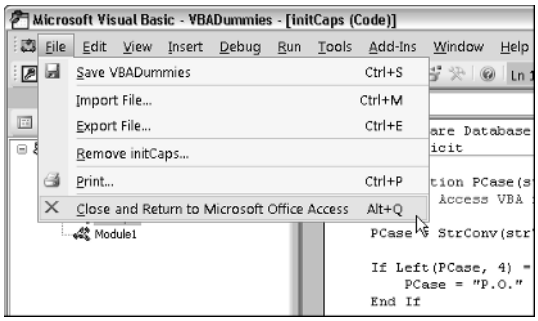

Figure 1-7: The Visual Basic Editor File menu.

The Visual Basic Editor closes, its taskbar button disappears, and you return to the Access program window.

Finding Out How VBA Works

When you open a standard module or class module, there's no telling exactly what you see inside. Some modules are empty; others already contain some VBA code. It all depends on the life history of the module you open. One thing is for sure: If any VBA code is in the module, it's likely organized into one or more procedures.

The term *procedure* in everyday language usually refers to performing a series of steps in order to achieve some goal. For example, the procedure of getting to work every morning requires a certain series of steps. The same definition holds true for VBA code.

Discovering VBA procedures

A VBA *procedure* is a series of instructions written in VBA code that tells an application (like Access) exactly how to perform a specific task. In VBA code, each step in the procedure is a single line of code: a *statement.* When Access executes a VBA procedure, it does so step-by-step, from the top down. Access does whatever the first statement tells it to do. Then it does whatever the second statement tells it to do, and so forth, until it gets to the end of the procedure.

Exactly when Access executes a procedure is entirely up to you. Typically, you want to tie the procedure to some event that happens on-screen. For example, you might want the procedure to perform its task as soon as someone clicks a button. Or perhaps you want your procedure to do its thing whenever someone types an e-mail address into a form. We talk about how that all works in Chapter 6. For now, just realize that you can tie any procedure you create to any event you like.

When the event to which you've tied your procedure occurs, Access *calls* the procedure. What that means is that Access does exactly what the VBA code in the procedure tells it to do. You can envision the process as shown in Figure 1-8 where

1. An event, such as clicking a button, calls a procedure.
2. Access executes the first line in the called procedure; then it executes the second line in the procedure; and so on.

3. When Access encounters the end of the procedure (which is either `End Sub` or `End Function`), it just stops executing code and returns to its normal state.

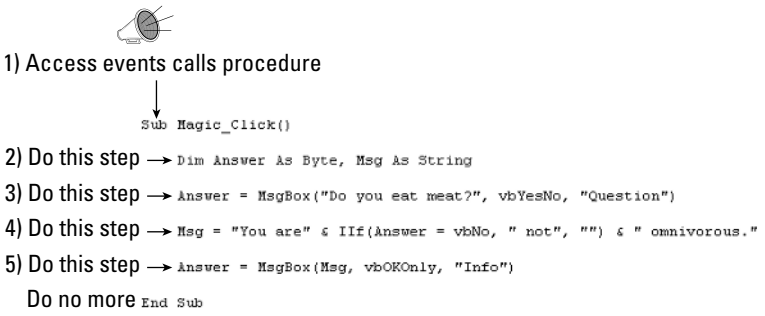

Figure 1-8: Executing a procedure.

If you think of a line of VBA code as a sentence containing words, a procedure is a paragraph containing more than one sentence.

Recognizing VBA procedures

VBA has two types of procedures. One type is a `Sub` procedure. A `Sub` procedure is always contained within a pair of `Sub...End Sub` statements, like this:

```
Sub subName(...)
   'Any VBA code here
End Sub
```

The *subName* part of the example is the name of the procedure. The `(...)` part after the name can be empty parentheses or a list of parameters and data types. The *'Any VBA code here* part stands for one or more lines of VBA code.

When looking at code that has already been written, you see that some `Sub` procedures have the word `Public` or `Private` to the left of the word `Sub`, as in these examples:

```
Private Sub subName(...)
   'Any VBA code here
End Sub

Public Sub subName(...)
   'Any VBA code here
End Sub
```

`Public` or `Private` defines the *scope* of the procedure. Neither type is particularly important right now. All that matters is that you know that a `Sub` procedure is a chunk of VBA code that starts with a `Sub` or `Private Sub` or `Public Sub` statement and ends at the `End Sub` statement.

If you must know right now, a `Public` procedure has global scope (is available to all other objects). A `Private` procedure is visible to only the module in which it's defined. For example, `Private Sub` procedures in a class module are private to the form or report to which the class module is attached.

The second type of procedure that you can create in Access is a `Function` procedure. Unlike a `Sub` procedure, which performs a task, a `Function` procedure generally does some sort of calculation and then returns the result of that calculation. The first line of a `Function` procedure starts with the word `Function` (or perhaps `Private Function` or `Public Function`) followed by a name. The last line of a `Function` procedure reads `End Function`, as illustrated here:

```
Function functionName(...)
    'Any VBA code here
End Function
```

A module can contain any number of procedures. When you open a module, you might at first think you're looking at one huge chunk of VBA code. But in fact you might be looking at several smaller procedures contained within the module, as shown in the example in Figure 1-9. Notice how each procedure within the module is separated by a black line that's the width of the page.

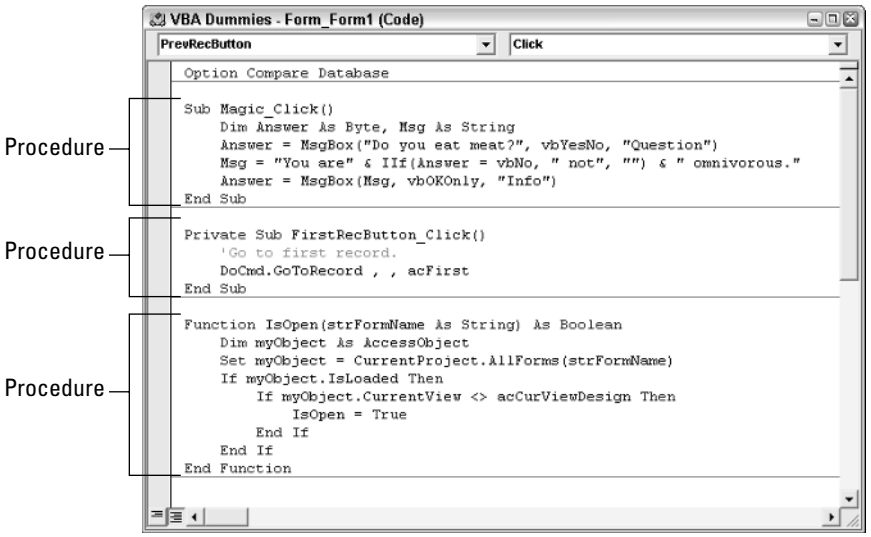

Figure 1-9: A module containing three procedures.

That's the view of Microsoft Access and VBA from 30,000 feet. Just remember that VBA is a programming language that allows you to write instructions that Access can execute at any time. You can write different sets of instructions for different events. Each set of instructions is a procedure, which is a series of steps carried out in a particular sequence to achieve a goal. You write and edit VBA code in the VBA Editor.

The beauty of it all is that you can write lots of little procedures to handle some of your more mundane tasks automatically and effortlessly. You can also extend Access's capabilities by writing procedures that do the tasks Access can't do on its own.