Chapter 1

Where VBA Fits In

In This Chapter

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

This is a book 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 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 will always be to create the tables for storing your data. From there, you can then create queries, forms, reports, and macros to help manage those data. All these steps take place before you even get into VBA. So in this book, I 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 is not a good place to start. If you need to brush up further on Access, *Access 2003 For Dummies* (John Kaufeld, Wiley) or *Access 2003 All-in-One Desk Reference For Dummies* (Alan Simpson, Margaret Levine Young, and Alison Barrows; Wiley) would be a good place to start.

Although Access has progressed through many versions over the years, VBA has remained relatively unchanged. I used both Access 2002 and Access 2003 to create this book, but the code examples presented in this book should work fine in just about any version of Access. So now, before launching into VBA, take a moment to discuss 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 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 database window, 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 left column of the database window is the Object list, and each name in the list represents a type of object, as summarized here.

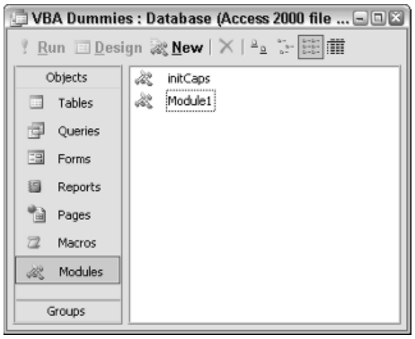


Figure 1-1: The Access database window.

- **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 as well as define relationships among multiple related 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* are objects that define how data should be presented on printed reports.
- **Pages:** *Pages* are similar to forms, but users can access data in tables through a Web browser rather than directly through Access.
- **Macros:** *Macros* provide a means of automating certain aspects of Access without programming.

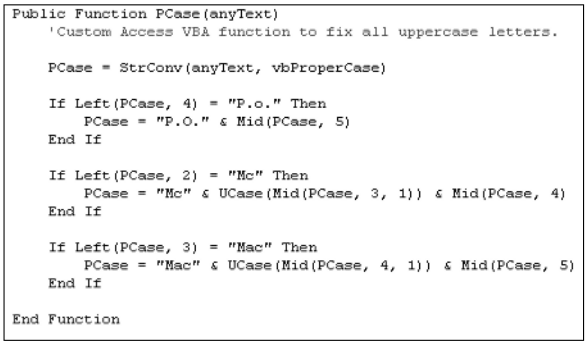
The Modules container, as you'll soon discover, is one of the places where you store VBA code. If you're not already familiar with modules, that's fine. Modules are what this book is really all about. Groups, of course, aren't really separate objects but rather just collections of existing objects. Sort of Access's version of *Favorites.*

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 plan 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 learning VBA is learning what all the various words in the language mean so that you can write code that tells Access exactly how to perform some 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'll master in this book.

Because VBA code looks like a bunch of meaningless gibberish typed onto a sheet of paper, this begs the question of why anybody would want to learn to read and write some dreadful language like that. The answer to that question lies in the role played by VBA in an application like an Access database.

Do, not die

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

The ability to use the same code over and over again is key to automating mundane tasks in Access. For example, if you used Access to print checks, you might have to manually type the part of the check where you type the amount in words, like *Ninety-two and 99/100 Dollars* for $92.99 because Access can't do 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.

Access does indeed have a ton of tools that let you create a database without any programming at all. You could easily spend months or years just learning 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 will want your database to accomplish some 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.

REMEMBER

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 actually perform the steps described by that code until Access executes the code. When you write VBA code, you're actually writing a set of instructions that Access can perform at any time, over and over again.

Seeing Where VBA Lurks

In an Access database, VBA code is stored in *modules.* Despite the fancy name, a module is basically an electronic sheet of paper on which VBA code is typed. The two types of modules in Access are

- **Standard module:** A page that contains VBA code that's accessible to all objects in the database.
- **Class module:** A page of code that's attached to every form and report you create. VBA code in the class module is accessible only to the form or report to which the class module is attached.

The main difference between a standard module and a class module is one of scope. VBA code in a standard module has a *global scope,* which means that the code can be accessed by every object in the database. A class module has a *local scope,* meaning that its code is accessible only to one form or one report in the database.

I talk about the issue of scope as it becomes relevant throughout this book. Right now, it's 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, standard modules get their own button in the Object list at the left side of the database window (refer to Figure 1-1). When you click the Modules button, the main pane shows the names of standard modules (if any) within the current database, as in the example shown in Figure 1-3.

TIP

Don't be surprised if you click the Modules button in a database, and the main pane is empty. Standard modules don't just happen: You have to create them.

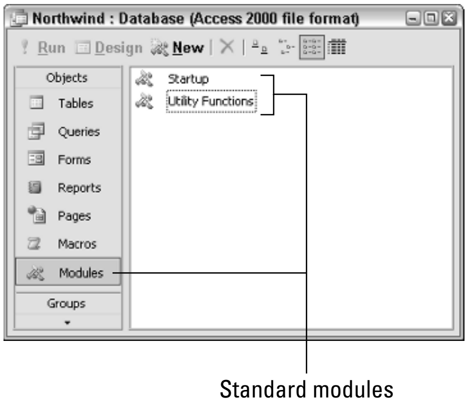


Figure 1-3: Standard modules in a database.

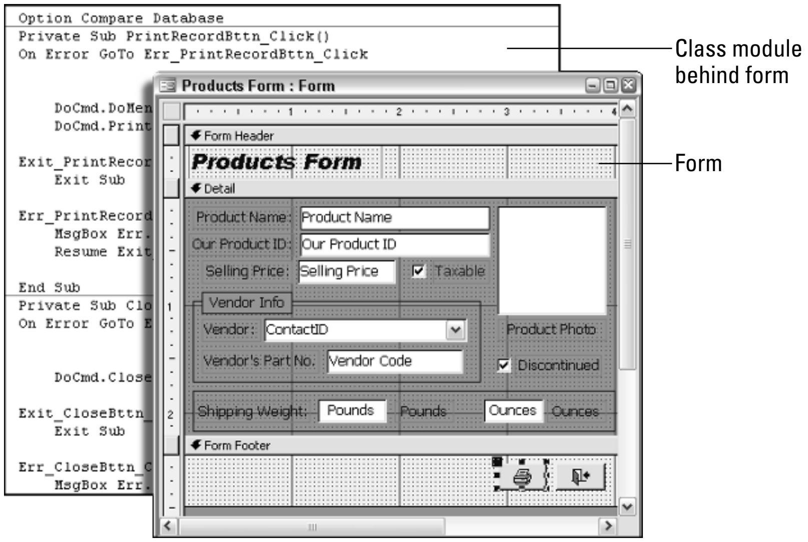
Finding class modules

Like standard modules, *class modules* contain VBA code that tells Access what to do. Unlike standard modules, however, you won't find any class modules in the database window. Class modules are hidden behind forms and reports in your database.

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, reports are another, and so forth. 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.*

Class modules are not global nor public like standard modules. To the contrary, class modules are very private beasts. They bring new meaning to the concept of hermit. Not only are class modules invisible to you most of the time, but they're always invisible to each other. The VBA code in a class module is visible (and usable) only to the form or report to which the class module is attached.

I think that it helps to envision a class module as literally being hidden behind its form, as in Figure 1-4. The VBA code in the class module is always hidden from the other objects in the database. The class module might be hidden from you as well if you don't know how to find it.



```
Option Compare Database
Private Sub PrintRecordBttn_Click()
On Error GoTo Err_PrintRecordBttn_Click

    DoCmd.DoMen
    DoCmd.Print

Exit_PrintRecor
    Exit Sub

Err_PrintRecord
    MsgBox Err.
    Resume Exit

End Sub
Private Sub Clo
On Error GoTo E

    DoCmd.Close

Exit_CloseBttn_
    Exit Sub

Err_CloseBttn_C
    MsgBox Err.
```

Class module behind form
Form
Products Form : Form
Form Header
Products Form
Detail
Product Name: Product Name
Our Product ID: Our Product ID
Selling Price: Selling Price
Taxable
Vendor Info
Vendor: ContactID
Vendor's Part No. Vendor Code
Product Photo
Discontinued
Shipping Weight: Pounds Pounds Ounces Ounces
Form Footer

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

You have several ways to get to a form or report's class module, as you'll 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 database window, click Forms or click Reports, depending on which type of object you want to open.**

2. **Right-click the name of any form or report and choose Design View.**

   To see the class module for the open form or report, click the Code button on the toolbar or choose View➪Code from the Access menu bar (see Figure 1-5).

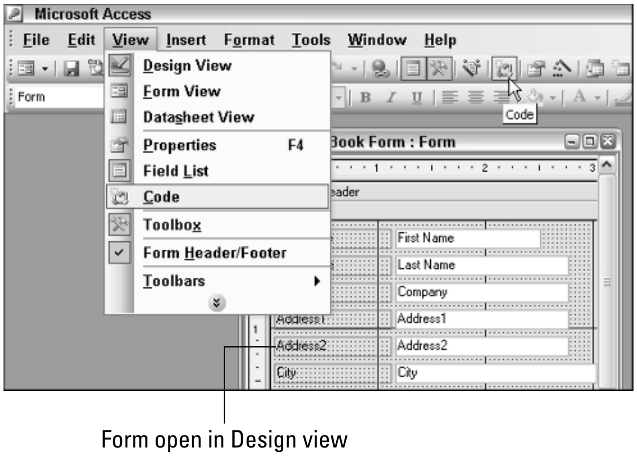


Figure 1-5: Class modules are accessible from form and reports Design views.

From VBA to Access

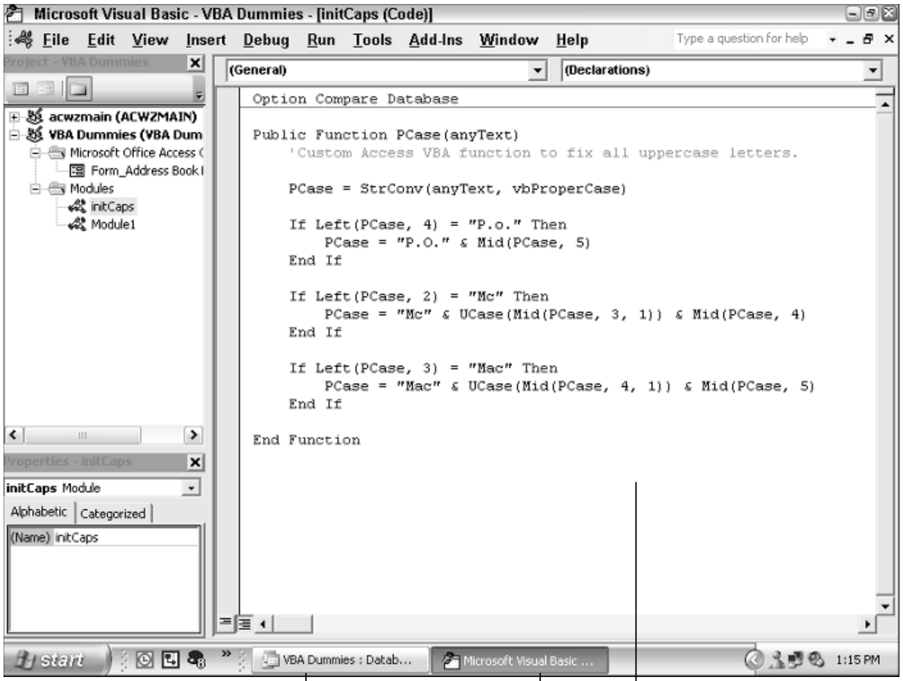
When you open a module, whether it's a standard module or a class module, your screen will change radically. That's because the module opens in the *Visual Basic editor*, which is a separate program window from Access. In fact, if you look on the taskbar, you'll still see a taskbar button for Access.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

TIP

If the module you open contains any VBA code, that code is visible in the editor Code window, also shown in Figure 1-6. 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 the form and report Design views automatically write VBA code for you behind the scenes. But let's not get ahead of ourselves.

The main thing to keep in mind here is that every time you open a module, you will end up in that Visual Basic editor. You'll discover how to use that program in upcoming chapters. For now, the most important thing to know is how to close it 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 more familiar Access program window:

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



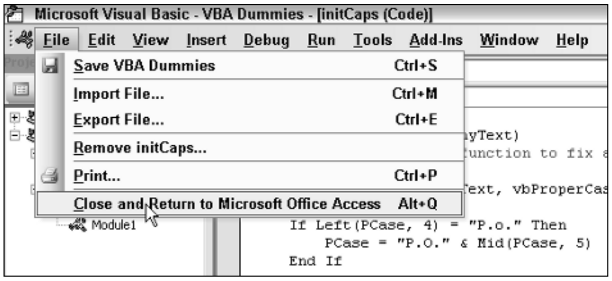
```
Option Compare Database

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 1-6: Move between the Visual Basic editor and Access.

TIP

You can press Alt+F11 to switch back and forth between Access and the VBA editor at any time.

The Visual Basic editor closes, its taskbar button disappears, and you're returned to the Access program window.



Microsoft Visual Basic - VBA Dummies - [initCaps (Code)]
File Edit View Insert Debug Run Tools Add-Ins Window Help
Save VBA Dummies Ctrl+S
Import File... Ctrl+M
Export File... Ctrl+E
Remove initCaps...
Print... Ctrl+P
Close and Return to Microsoft Office Access Alt+Q
Module1

Figure 1-7: The VB editor File menu.

Finding Out How VBA Works

When you open a standard module or class module, there's no telling exactly what you'll see inside. Some modules will be empty; others will already contain some VBA code. It all depends on the life history of the module you open. But one thing is for sure: If any VBA code is in the module, it will likely be 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. A procedure is a series of steps carried out in a specific order to achieve some desired result.

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 onscreen. For example, you might want the procedure to do 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. I talk about how that all works in Chapter 6. For now, just realize you can tie any procedure you create to any event you like.

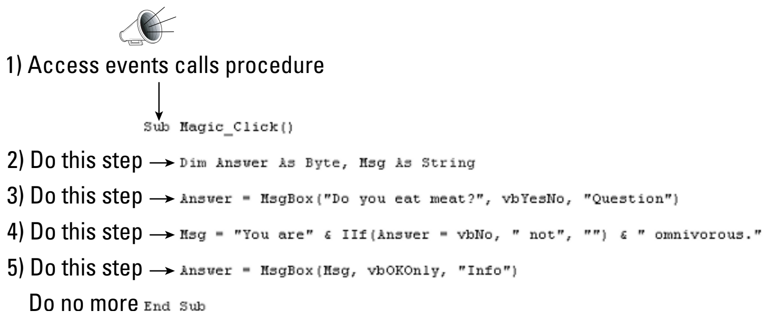
Why would my database contain code?

Those of you who've never written any code might be wondering how any database you've created could possibly contain code. The answer to that riddle lies in the Control Wizards button in the forms and reports Design views.

When you use a Control Wizard to add a button or certain other types of controls to a form, the Control Wizard actually writes VBA code for you. It stores that code in the class module that's hidden behind the form (or report) module.

When the event to which you've tied your procedure occurs, Access *calls* the procedure. What that really means is that Access does exactly what the VBA code in the procedure tells it to do. You can envision the process as 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 will be either `End Sub` or `End Function`), it just stops executing code and returns to its normal state.**



```
Sub Magic_Click()
Dim Answer As Byte, Msg As String
Answer = MsgBox("Do you eat meat?", vbYesNo, "Question")
Msg = "You are" & IIf(Answer = vbNo, " not", "") & " omnivorous."
Answer = MsgBox(Msg, vbOKOnly, "Info")
End Sub
```

Figure 1-8: Executing a procedure.

If you think of a line of VBA code as a sentence containing words, a procedure would be 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, as follows:

```
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 could 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's already been written, you'll 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 is particularly important right now. All that matters right now is that you know that a `Sub` procedure is a chunk of VBA code that starts with `Sub` or `Private Sub` or `Public Sub` statement and ends at the `End Sub` statement.

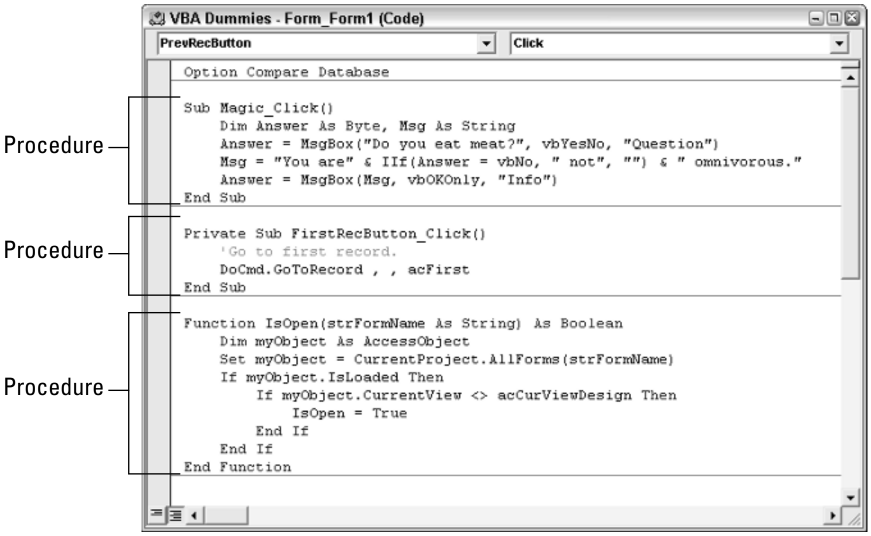
For those of you who must know right now, a `Public` procedure has global scope (is available to all other objects). A `Private` procedure is visible to only the procedure in which it's defined. For example, `Sub` procedures in a class module are private to the form or report to which the class module 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 illustrated in the example shown in Figure 1-9. Notice how each procedure within the module is separated by a black line that's the width of the page.

So that's the bird's-eye 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.



```
Option Compare Database

Sub Magic_Click()
    Dim Answer As Byte, Msg As String
    Answer = MsgBox("Do you eat meat?", vbYesNo, "Question")
    Msg = "You are" & IIf(Answer = vbNo, " not", "") & " omnivorous."
    Answer = MsgBox(Msg, vbOKOnly, "Info")
End Sub

Private Sub FirstRecButton_Click()
    'Go to first record.
    DoCmd.GoToRecord , , acFirst
End Sub

Function IsOpen(strFormName As String) As Boolean
    Dim myObject As AccessObject
    Set myObject = CurrentProject.AllForms(strFormName)
    If myObject.IsLoaded Then
        If myObject.CurrentView <> acCurViewDesign Then
            IsOpen = True
        End If
    End If
End Function
```

Figure 1-9: A module containing three procedures.

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.