SESSION

1

Introduction to Microsoft Access and Programming

Session Checklist

✔ Understanding what programming is

✔ Using the Visual Basic language

✔ Programming for the contemporary developer

✔ Examining database programming

✔ Using the Check Writer example database

✔ Reviewing the basic Access program structure: modules, functions, and subprocedures

30 Min. To Go

Microsoft Access is an outstanding environment for both database users and professional developers. In this session, you learn the difference between programming with the Visual Basic language and using Microsoft Access tables, queries, forms, and reports. You also learn how professional developers use Microsoft Access to create applications, and you are introduced to the Check Writer application example used throughout this book.

In order to get the most from this book, you should be familiar with most of the Microsoft Access objects — including tables, queries, forms, and reports. You should also know the basic concepts of building tables and creating relationships. If you have never created a Microsoft Access form, this book is probably not for you. All the examples start from forms that have been created and explain how to add functions and procedures using Visual Basic, the internal language of Access.

If you have created simple or complex macros, you already understand the basics of events and programming. This book does not teach macros because professional developers use macros only on rare occasions. For example, macros are used for creating certain types of menus or for avoiding .mda library-referencing problems, which cause compile errors when the libraries are not connected. This book does explain how to convert your macros to modules and thoroughly covers event-driven programming, which is used in Microsoft Access applications.

What Is Programming?

Programming is the name given to the process of creating instructions to accomplish a task. This is just one of the many phases of development. Microsoft Access contains a set of tools, one of which is called a *language*. Just as you build words, sentences, and thoughts when you speak or write using human language, you use the programming language to create the program. The language consists of a series of commands that tells the computer how to accomplish the task at hand. Computer programming languages have rules and grammar just like spoken languages.

Tip

Computer grammar is called *syntax*.

Programming can be defined in many ways, but usually words such as *logic*, *structure*, *commands*, *sequence*, and *order* are part of the definition. Professional programmers today generally prefer to be called *developers*. A developer is a person who creates computer applications. Traditional programming is not a necessary element. Creating a form using Microsoft Access can be considered development. When an error message or process is added to the form with a macro or language element, that is also programming. Programming is only one element of development. Analysis, design, testing, and debugging are other key elements of application development.

Microsoft Access contains a variety of tools that enable you to build applications without using the built-in language. These include queries, forms, and reports. Microsoft Access is known as a *database management package* because it gives you the ability to create tables that hold data. If you have created macros using Microsoft Access, you have already programmed, whether you know it or not.

Visual Basic is the language that is used internally with Microsoft Access. It is also called VBA or Visual Basic for Applications, Visual Basic — Applications Edition, and the Visual Basic language. It was called Access Basic in the first versions of Microsoft Access. Whatever you call it, the Visual Basic language is an integral part of Microsoft Access. For the purposes of this book, the Microsoft Access programming language is referred to simply as Visual Basic.

Why Use the Visual Basic Language?

Although Microsoft Access allows you to create process-oriented programs using macros, it is the language that gives you unlimited flexibility. Many Microsoft Access programmers started with macros and eventually realized their limitations. Eventually, you will find that to meet all your needs in Access, you need to program using the Visual Basic language.

There are many things that macros cannot do. Macros cannot

- Create error trapping routines and run a process based on the error
- Use repetitive looping or incrementing of variables

- Perform complex decision making
- Replace runtime parameters to change form display

Tip

Don’t confuse the Visual Basic language within Microsoft Access with the product Microsoft sells called Visual Basic or Visual Basic .NET. Although Visual Basic (the product) and Microsoft Access share the common Visual Basic language, the products themselves are very different. All Microsoft Office products contain roughly the same Visual Basic programming language. These products include Access, Excel, Word, PowerPoint, Outlook, and even MapPoint.

Programming for the Contemporary Developer

If you are fairly new to Microsoft Access and have been a programmer for several decades, you may be wondering, “Why program at all?” You may have already discovered the incredible power of tables, queries, forms, and reports. You may have created fairly complex forms including combo boxes, option groups, ActiveX Objects, and the like, which provide more power than many programs you used just 10 years ago. You also may be very comfortable using macros.

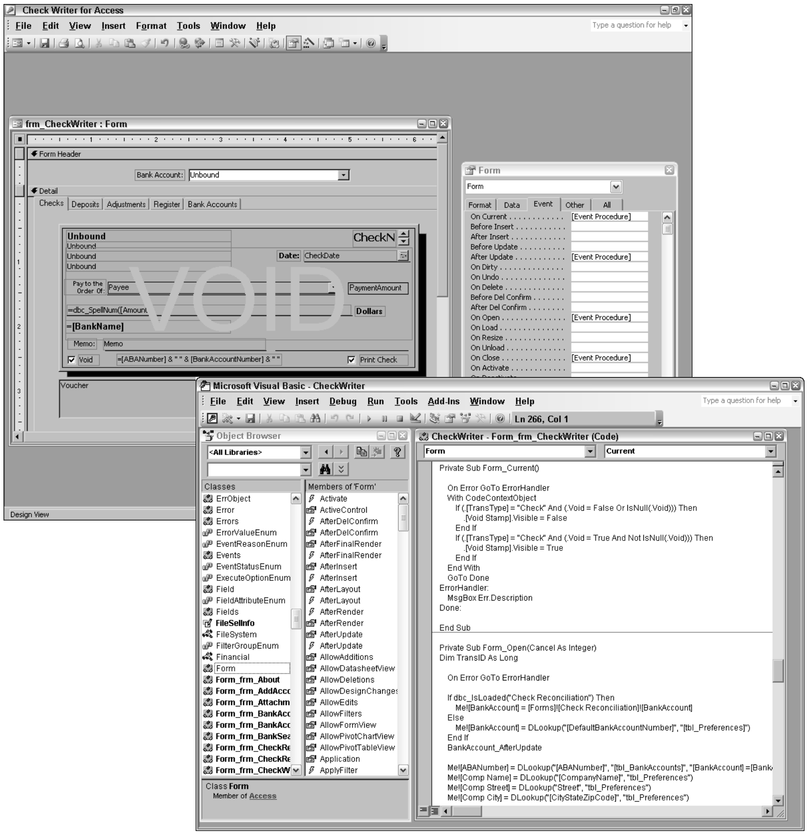
If you have already been programming in languages such as COBOL or dBASE, you may be looking for the window that lets you type in commands so that you can start programming. Although there is such a screen in Microsoft Access, you don’t use the Visual Basic Editor window to write a program (as you might have back in the ’80s). In the early days of personal computers, you started with a blank screen and an editor similar to a simple word processor. You wrote line after line of computer language code to do such mundane things as

Draw a rectangle	`@12,15 To 25,40`
Display a text label	`@13,17 SAY "Customer Name:"`
Allow the user to enter data	`@13,31 GET CUSTNAME`

In those days, you had to worry about where the cursor was and where it was going. Today’s visual tools, such as the Form Design window, make it easy to build forms without ever entering a line of computer code. In fact, using Microsoft Access, you can build fairly sophisticated applications without ever writing a single line of Visual Basic code. You could easily spend a day creating the program for a simple form in dBASE II. In Microsoft Access, you can create the same form with a wizard in a couple of minutes or from the Form Design screen in half an hour.

Microsoft Access uses an event-driven visual programming environment. This means that, generally, you create a form to display something and then, in the Design view of the form, you use the events of the form — the form’s controls, mouse movements, or keyboard keypresses — to add programmed instructions that go beyond just simple form display and data editing. It is a visual environment because you see the user interface at all times. Additionally, as you will learn in the next session, you can view the results of your work nearly instantaneously with only a few mouse clicks. If you are used to the mainframe world, you know the pain of compiling and linking.

An *event* is just that — an event. Examples include a form being opened, a user placing the cursor in a field on a form, a data value being changed, your mouse moving to a specific control, or more than 50 other distinct occurrences for which you can write Visual Basic code. You can also write Visual Basic code for many other things that happen to forms and reports: printing reports, trapping for potential errors, and even checking the passage of time and performing some task after a certain number of seconds. Each of these events serves as a trigger for code to be run (or *executed*, as it is also called in programming vernacular). Figure 1-1 shows some of the events that are behind a form and a Visual Basic window where a simple program has been created to check the value of text box entry and to display an error message if it is null.


Check Writer for Access
File Edit View Insert Format Tools Window Help
frm_CheckWriter : Form
Form Header
Bank Account: Unbound
Detail
Checks Deposits Adjustments Register Bank Accounts
VOID
Design View
Microsoft Visual Basic - CheckWriter
File Edit View Insert Debug Run Tools Add-Ins Window Help
Ln 266, Col 1
Object Browser
CheckWriter - Form_frm_CheckWriter (Code)
Form
Current

Form	
On Current	[Event Procedure]
Before Insert	
After Insert	
Before Update	
After Update	[Event Procedure]
On Dirty	
On Undo	
On Delete	
Before Del Confirm	
After Del Confirm	
On Open	[Event Procedure]
On Load	
On Resize	
On Unload	
On Close	[Event Procedure]
On Activate	

```
Private Sub Form_Current()

    On Error GoTo ErrorHandler
    With CodeContextObject
        If (.[TransType] = "Check" And (.Void = False Or IsNull(.Void))) Then
            .[Void Stamp].Visible = False
        End If
        If (.[TransType] = "Check" And (.Void = True And Not IsNull(.Void))) Then
            .[Void Stamp].Visible = True
        End If
    End With
    GoTo Done
ErrorHandler:
    MsgBox Err.Description
Done:

End Sub

Private Sub Form_Open(Cancel As Integer)
Dim TransID As Long

    On Error GoTo ErrorHandler

    If dbc_IsLoaded("Check Reconciliation") Then
        Me![BankAccount] = [Forms]![Check Reconciliation]![BankAccount]
    Else
        Me![BankAccount] = DLookup("[DefaultBankAccountNumber]", "[tbl_Preferences]")
    End If
    BankAccount_AfterUpdate

    Me![ABANumber] = DLookup("[ABANumber]", "[tbl_BankAccounts]", "[BankAccount] =[Bank
    Me![Comp Name] = DLookup("[CompanyName]", "tbl_Preferences")
    Me![Comp Street] = DLookup("Street", "tbl_Preferences")
    Me![Comp City] = DLookup("[CityStateZipCode]", "tbl_Preferences")
```

Figure 1-1 *Sample events behind a form and the Visual Basic window showing a simple program.*

20 Min. To Go

Database Programming Is Incredibly Flexible

A number of popular software products today include or are built primarily around programming languages. Some of these are .NET, C++, Java, Visual FoxPro, Delphi, Visual Basic, Microsoft Access, and Microsoft Excel. Many more products from small companies with smaller followings are also available, but Microsoft Access is one of the most flexible because it is built around a database management system. Database management systems include the capability to build tables and relationships and to store data.

Microsoft Access is a multidimensional product because it also includes an easy-to-use but powerful set of form and report tools. Its Visual Basic language capability allows the automation or addition of extensions to your simple forms and reports, making them incredibly powerful.

Microsoft Access comes with a *back-end* database management system, which means it has two different data engines to manage data. The first engine is known as *Jet*. You may not be aware of this engine because Jet is built into Microsoft Access. Each time you create a new database, design a table, or write a query, you are using the Jet database engine. The second engine is a smaller version of SQL Server 2000 called the Microsoft Database Engine.

Note

The Microsoft Database Engine is also known as MSDE and has to be installed separately. You can find it in a separate directory on your Office 2003 installation CD.

If you create professional Microsoft Access applications using the Jet engine, you should always create two separate .mdb database files. One database file contains just your tables, whereas the other contains links to those tables, along with your queries, forms, reports, and module code. This technique is known as *file/server computing* because you have a program file and a data file. You may have heard it referred to as *client/server computing,* but the Jet database engine is not a true client/server database engine. Instead, it is a file/server database engine.

Real client-server database engines such as SQL Server or Oracle actually do all their processing on the database server hardware and minimize data sent across the network. Although Microsoft Access uses Jet as a database manager, all processing is done on the client workstation each time the entire data table is sent across the network. The optimum solution for large database applications is to use Microsoft Access as the front-end and to use SQL Server or Oracle as the back end.

Although SQL Server and Oracle are also very powerful database managers, they do not include a set of integrated tools and have no specific programming language. In fact, SQL Server uses Visual Basic to handle its own internal event model known as *triggers and stored procedures*. *Triggers* are events that respond to a change in a data value. This change triggers (or starts) a block of code known as a *stored procedure* (so named because it is also stored in the data table).

Microsoft Access 2003 contains a client-server database model called *projects* and uses the file extension .adp. This built-in client/server model uses the personal desktop version of SQL Server known as the Microsoft Database Engine (MSDE). You can create applications that work with MSDE instead of Jet and then use the more powerful SQL Server when you are ready.

Using the Check Writer Example Database

Included on the companion Web site for this book are sample files that we refer to in various sessions throughout the book. Many more free software samples and demo versions of business systems and third-party tools are available from leading Microsoft Access add-on vendors.

The example used in this book is a fairly simple application that is representative of the types of applications you can develop with Microsoft Access. The example is a working check writer, which is an electronic version of the checkbook you probably use all the time. The example is written in Microsoft Access and programmed with the latest version of Access 2003, Visual Basic, and the internal data access language ADO.

You will use these two files:

CheckWriter.mdb — The program file, including Queries, Forms, Reports, and Modules

CheckWriterData.mdb — The data file, containing only the tables used in the example

If you have never worked with linked databases, this is a great time to start.

Tip

Professional Microsoft Access application developers always keep the program and data in two separate database files. That way, if the programs or design objects (forms, reports, and so on) require changes, the developer can replace the customer's program file without disturbing the data files.

The sessions in this book use various parts of the Check Writer example to show you how to build any professional application. The application is constructed specifically to demonstrate what you must know to be successful in application development, including Visual Basic programming, data design, and forms and report creation.

Using the Check Writer main menu

When you first open the Check Writer application, you see the main menu shown in Figure 1-2. The main menu consists of a few simple buttons and some graphics. When you click a button, that function runs.

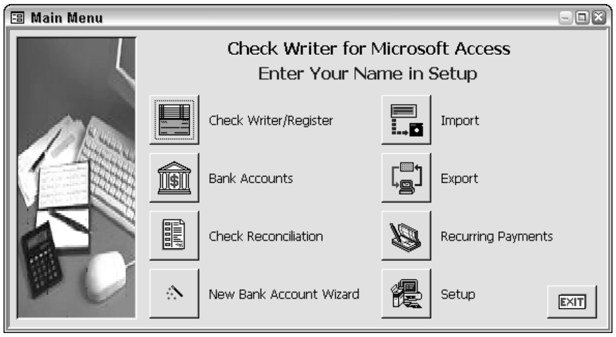


Figure 1-2 *The Check Writer example main menu.*

You can choose from the following functions on the main menu:

Check Writer/Register — A tabbed dialog where you can add, edit, delete, or display checks, deposits, adjustments, and a visual check register.

Bank Accounts A data entry form to enter bank account information for your accounts.

Check Reconciliation A complete electronic check reconciliation screen to help you balance your checkbook.

New Bank Account Wizard A wizard form to create a new bank account and set up the starting balance.

Import A utility form to demonstrate data import and integration.

Export A utility form to demonstrate exporting data.

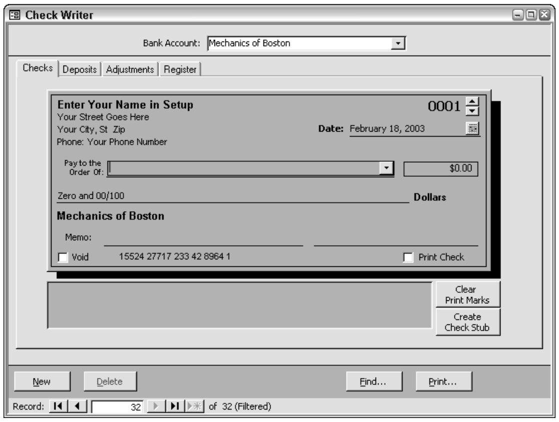
Recurring Payments A form to demonstrate creating a payment that occurs more than once, including saving or retrieving the data.

Setup A simple form to enter your company name and address.

Exit A function that closes the Check Writer application and exits Microsoft Access.

Using the Check Writer tabbed data entry form

The main Check Writer/Register application consists of four tabs. Each tab contains a separate subform that displays one or more records. The tabs filter the data to show only checks, deposits, or adjustments in the first three tabs. The fourth tab shows a check register containing all types of transactions. Figure 1-3 shows the first tab, displaying the check form itself. The check form lets you enter the check number, enter the check date, select or enter a new payee, and enter the amount of the check.



Bank Account: Mechanics of Boston

Enter Your Name in Setup
Your Street Goes Here
Your City, St Zip
Phone: Your Phone Number

0001

Date: February 18, 2003

Pay to the Order Of:

$0.00

Zero and 00/100 Dollars

Mechanics of Boston

Memo:

□ Void 15524 27717 233 42 8964 1 □ Print Check

Record: 32 of 32 (Filtered)

Figure 1-3 *The Check Writer/Register example tabbed data entry form.*

As you learn throughout this book, this form requires a great deal of programming. A function behind the Amount field is used to automatically convert the amount to words. A pair of spin buttons in the upper-right corner of the check let you automatically increase or decrease the check number. When a new payee is added, a program makes sure that you want to add the new value and adds a record to the separate payee database.

Several check boxes appear at the bottom of the form. One check box voids the check and displays a large VOID stamp across it; the other check box prints the check. You can fill in the stub below the check using a button to retrieve values from the check and format them into a single string.

The area in the footer of the form also contains buttons that display great examples of a search-and-print dialog as well as New and Delete buttons that use professional error-trapped subprocedures for handling New and Delete tasks.

Understanding the Check Writer data model

The data model used in the Check Writer example is shown in Figure 1-4. Most of the transaction data is stored in the tbl_CheckWriter table. This table is used to store checks, deposits, and adjustments.

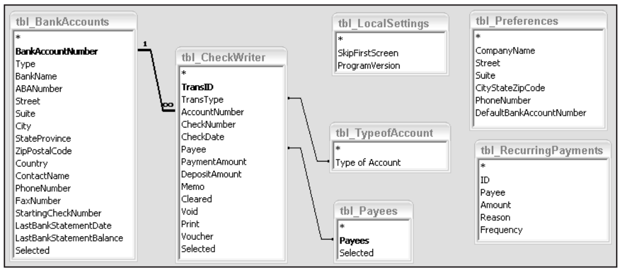


Figure 1-4 *The Check Writer example data model.*

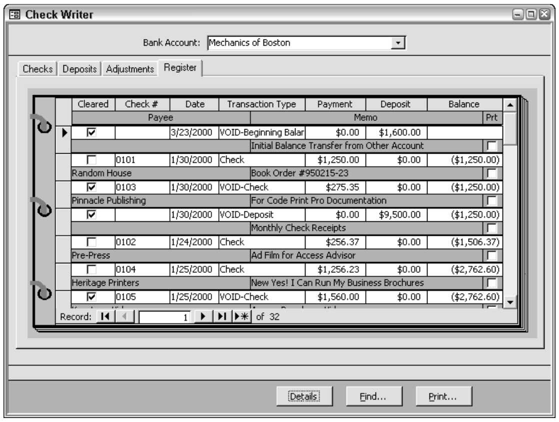
Whereas all the data fields necessary to enter check, deposit, and adjustment data are contained in this table, some fields are also used for reconciliation (balancing your checkbook), printing, and voiding. The TransType field is used to identify the type of transaction and to filter the records for each tab in the frm_CheckWriter form.

The tbl_BankAccounts table contains all necessary information for adding bank accounts and relating this information to the tbl_CheckWriter table. The primary key field for the tbl_BankAccounts table is BankAccountNumber, whereas the primary key in the tbl_CheckWriter table is an AutoNumber field named TransID. The foreign key field to the tbl_BankAccounts table is the AccountNumber field in the tbl_CheckWriter table. An AutoNumber field is used because the other more distinctive keys don’t make sense in a check writer. Although checks usually have sequential numbers, deposits and adjustments do not. Therefore, there is no foreign key combination that works. BankAccountNumber, Date, and TransType cannot work because you can have multiple checks, deposits, or adjustments on the same day.

There are two additional tables used for lookups in the example. The tbl_TypeofAccount table contains a list of transaction types including check, deposit, and adjustments such as Error Correction, Interest, Wire Transfer, ATM, and so on. The tbl_Payees table contains a list of unique payees used by the checks. Each time you enter a new payee into the frm_CheckWriter form payee line, you have the option to add any new payees into the table.

Using the Check Register

The Check Register is found in the last tab of the frm_CheckWriter form, as shown in Figure 1-5. There is not a lot of code behind this form, but the subform is a great example of a continuous form with a complex calculation to display the continuous balance.



Bank Account: Mechanics of Boston

Cleared	Check #	Date	Transaction Type	Payment	Deposit	Balance	Payee	Memo	Prt
☑		3/23/2000	VOID-Beginning Balar	$0.00	$1,600.00			Initial Balance Transfer from Other Account	□
□	0101	1/30/2000	Check	$1,250.00	$0.00	($1,250.00)	Random House	Book Order #950215-23	□
☑	0103	1/30/2000	VOID-Check	$275.35	$0.00	($1,250.00)	Pinnacle Publishing	For Code Print Pro Documentation	□
☑		1/30/2000	VOID-Deposit	$0.00	$9,500.00	($1,250.00)		Monthly Check Receipts	□
□	0102	1/24/2000	Check	$256.37	$0.00	($1,506.37)	Pre-Press	Ad Film for Access Advisor	□
□	0104	1/25/2000	Check	$1,256.23	$0.00	($2,762.60)	Heritage Printers	New Yes! I Can Run My Business Brochures	□
☑	0105	1/25/2000	VOID-Check	$1,560.00	$0.00	($2,762.60)			

Record: 1 of 32

Figure 1-5 *The Check Register form.*

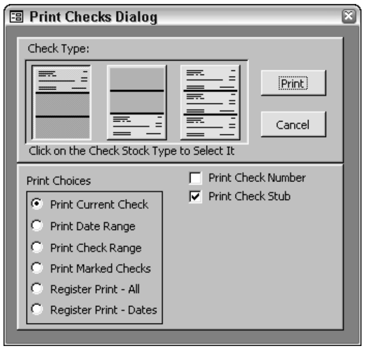
The Check Register subform is made up of multiple lines. In later sessions of this book, you learn how to use multiple-line subforms that include a surrounding OLE object and an editable data source. You also learn how to change the record source of the subform programmatically (using code) when the form is initially opened and how to change a control's calculated control source programmatically.

Viewing the Print Checks dialog

The Print Checks dialog shown in Figure 1-6 is displayed when you click the Print button at the bottom of the Check Writer form. This example is used in several sessions to illustrate the interaction between Visual Basic and dialog boxes. Based on the choices the user selects in the dialog box, various blocks of Visual Basic code are run to select different reports, change query dynasets (the results of a query) that are passed to different reports, and even determine fields that are printed or hidden on reports.

Managing the Check Reconciliation process

The frm_CheckReconciliation form is shown in Figure 1-7. This screen contains a set of tools that allows the user to visually reconcile a checkbook. The user first enters the bank balance to view the difference between the checkbook balance and bank balance, and then the user clicks the Cleared button and watches the difference shrink. Using the combo box on the form, the user employs Visual Basic code to switch between bank accounts. The EDIT button in the continuous subform also allows the user to display the selected transaction by opening the Check Writer form, changing the tab, and finding the record programmatically. This provides a good lesson for navigation within multiple forms.


Print Checks Dialog

Check Type:

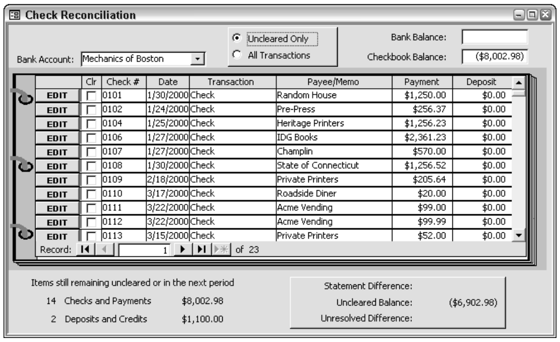
Click on the Check Stock Type to Select It

Print Choices

- ● Print Current Check
- ○ Print Date Range
- ○ Print Check Range
- ○ Print Marked Checks
- ○ Register Print - All
- ○ Register Print - Dates

□ Print Check Number
☑ Print Check Stub

Figure 1-6 *The Print Checks dialog box.*


Check Reconciliation

Bank Account: Mechanics of Boston

● Uncleared Only
○ All Transactions

Bank Balance:
Checkbook Balance: ($8,002.98)

	Clr	Check #	Date	Transaction	Payee/Memo	Payment	Deposit
EDIT	□	0101	1/30/2000	Check	Random House	$1,250.00	$0.00
EDIT	□	0102	1/24/2000	Check	Pre-Press	$256.37	$0.00
EDIT	□	0104	1/25/2000	Check	Heritage Printers	$1,256.23	$0.00
EDIT	□	0106	1/27/2000	Check	IDG Books	$2,361.23	$0.00
EDIT	□	0107	1/27/2000	Check	Champlin	$570.00	$0.00
EDIT	□	0108	1/30/2000	Check	State of Connecticut	$1,256.52	$0.00
EDIT	□	0109	2/18/2000	Check	Private Printers	$205.64	$0.00
EDIT	□	0110	3/17/2000	Check	Roadside Diner	$20.00	$0.00
EDIT	□	0111	3/22/2000	Check	Acme Vending	$99.00	$0.00
EDIT	□	0112	3/22/2000	Check	Acme Vending	$99.99	$0.00
EDIT	□	0113	3/15/2000	Check	Private Printers	$52.00	$0.00

Record: 1 of 23

Items still remaining uncleared or in the next period

14	Checks and Payments	$8,002.98
2	Deposits and Credits	$1,100.00

Statement Difference:
Uncleared Balance: ($6,902.98)
Unresolved Difference:

Figure 1-7 *The Check Reconciliation form.*

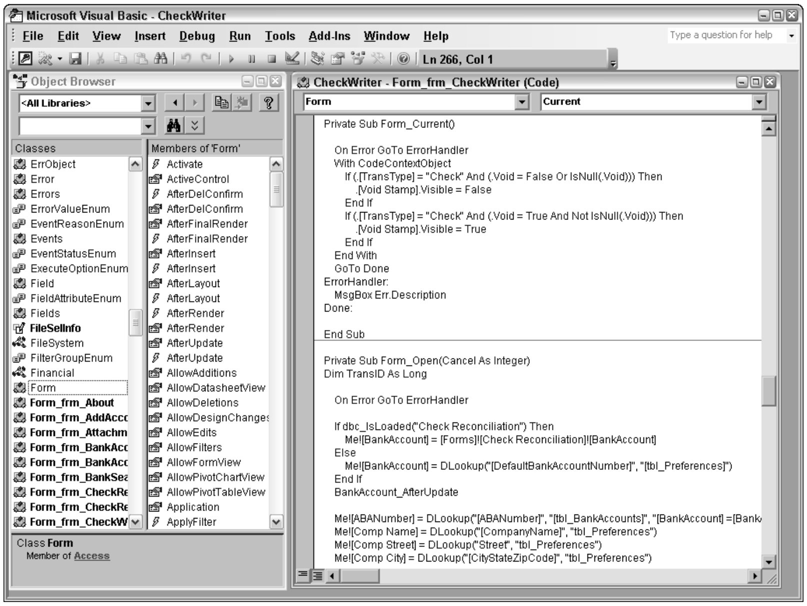
10 Min. To Go

A Typical Visual Basic Screen

In the next session, you learn how to create Visual Basic programs, and you should begin to understand the process of working with Visual Basic. Figure 1-8 shows the Visual Basic Editor window that, in Microsoft Access, is a separate program in a separate window.

Because the Visual Basic Editor is a separate program in Microsoft Access, you can see both Microsoft Access and the Visual Basic Editor in your task bar at the bottom of your Windows screen. Although this is a huge difference internally for Access, it simply means that you can view both windows simultaneously and the editor does not use memory that could otherwise be used by Access. With more memory available to Access, programmer productivity may be increased and Windows itself is more stable, which causes less stress on machines running Access programs.

Tip **The more clean and free memory you give Microsoft Access, the faster your programs will run and the more stable they will be. You should reboot your machine at least once a day, and if you have been moving between design screens and running your Access forms programs frequently, you should close Access once in a while (every hour perhaps) and restart it. Each time you move from a design screen to a datasheet or forms view, Access leaves behind a little dirt known as a *memory leak*. Eventually, your system will report "out of memory," or Access will report errors where there are none. Closing Access and restarting it or rebooting your system will clean up the internal memory. These problems are well known to professional Access developers.**



```
Private Sub Form_Current()

   On Error GoTo ErrorHandler
   With CodeContextObject
      If (.[TransType] = "Check" And (.Void = False Or IsNull(.Void))) Then
         .[Void Stamp].Visible = False
      End If
      If (.[TransType] = "Check" And (.Void = True And Not IsNull(.Void))) Then
         .[Void Stamp].Visible = True
      End If
   End With
   GoTo Done
ErrorHandler:
   MsgBox Err.Description
Done:

End Sub

Private Sub Form_Open(Cancel As Integer)
Dim TransID As Long

   On Error GoTo ErrorHandler

   If dbc_IsLoaded("Check Reconciliation") Then
      Me![BankAccount] = [Forms]![Check Reconciliation]![BankAccount]
   Else
      Me![BankAccount] = DLookup("[DefaultBankAccountNumber]", "[tbl_Preferences]")
   End If
   BankAccount_AfterUpdate

   Me![ABANumber] = DLookup("[ABANumber]", "[tbl_BankAccounts]", "[BankAccount] =[Bank
   Me![Comp Name] = DLookup("[CompanyName]", "tbl_Preferences")
   Me![Comp Street] = DLookup("Street", "tbl_Preferences")
   Me![Comp City] = DLookup("[CityStateZipCode]", "tbl_Preferences")
```

Figure 1-8 *The Visual Basic Editor window.*

Modules, Subprocedures, and Functions

You should understand a few more terms before moving on to the next session. When programmers think of programming in Microsoft Access, they think of Visual Basic modules. You already know that there is a Module tab in the database window. The Modules tab in the database window displays the module libraries created by you. You can have up to 1024 module libraries in a database. Each module library can contain many procedures, just as a bookcase can contain many books.

There are two types of procedures: functions and subprocedures. Functions and subprocedures are the building blocks of modules. Each module contains one or more function or subprocedure, and each function or subprocedure contains one or more Visual Basic statement.

Tip

You will often see the term *procedure* used interchangeably with the Microsoft Access term *subprocedure* and sometimes with *function*.

Procedures hold the Visual Basic program statements that make up modules. Just as a library has both hard-cover and soft-cover books, a module library has two different types of procedures. Both work exactly the same: they allow passed parameters to affect the way they process Visual Basic statements, they can contain error checking, and they can use any Visual Basic statement. The only difference between a function and a subprocedure is whether a value is returned when processing is completed. Functions return a value when processing is complete; subprocedures do not.

The Modules tab is not the only place where Visual Basic modules are found. You can place Visual Basic modules behind any form or report. They behave exactly like functions or subprocedures in a module library; they are just stored in a different place.

Done!

Tip

Generally, if a function or subprocedure is used only by a form or report, you store it behind the object. If it is to be used by more than one form or report, you store it in a module.

REVIEW

In this session, you learned the definition of programming and how Microsoft Access objects interact with the Visual Basic programming environment. You learned about database programming and the types of procedures used in Microsoft Access modules. You also learned about the Check Writer example used throughout this book. The following topics were covered:

- Programming is the name given to the process of creating instructions to accomplish a task.
- Visual Basic, or VBA, is the language used in Microsoft Access modules.
- Microsoft Access uses an event-driven, visual programming environment.
- Microsoft Access uses a file-server@ndbased database management system known as Jet to manage data. You can also create applications that work with the Microsoft Database Engine (MSDE) instead of with Jet, which then enables you to use the more powerful SQL Server when you are ready.
- The example used in this book is a working check writer, which is an electronic version of the checkbook you probably use all the time.
- Each module library can contain many procedures. Functions and subprocedures are the building blocks of modules.
- Whereas module libraries contain only Visual Basic procedures, form and report objects can also contain procedures embedded within the form or report object.

QUIZ YOURSELF

1. Define the term *programming*. (See "What is Programming?")
2. Name three reasons to use Visual Basic instead of macros. (See "Why Use the Visual Basic Language?")
3. What are the two internal database management systems that Microsoft Access contains? (See "Database Programming Is Incredibly Flexible.")
4. What is the difference between file/server and client/server database management systems? (See "Database Programming Is Incredibly Flexible.")
5. What are the main functions of the Check Writer example? (See "Using the Check Writer Example Database.")
6. Name the two types of procedures. (See "Modules, Subprocedures, and Functions.")