

Real World Microsoft Access Database Protection and Security

GARRY ROBINSON

Apress™

Real World Microsoft Access Database Protection and Security
Copyright © 2004 by Garry Robinson

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN (pbk): 1-59059-126-7

Printed and bound in the United States of America 12345678910

Appendix C reprinted with permission from *From Access to SQL Server*, ISBN: 1-893115-24-0, published by Apress. Copyright © 2000 by Russell Sinclair.

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Technical Reviewer: Frank Rice

Editorial Board: Dan Appleman, Craig Berry, Gary Cornell, Tony Davis, Steven Rycroft, Julian Skinner, Martin Streicher, Jim Sumser, Karen Watterson, Gavin Wray, John Zukowski

Assistant Publisher: Grace Wong

Project Manager: Sofia Marchant

Development Editor: Tracy Brown Collins

Copy Editor: Kristen Imler

Production Manager: Kari Brooks

Production Editor: Janet Vail

Proofreader: Linda Seifert

Compositor: Susan Glinert Stevens

Indexer: Kevin Broccoli, Broccoli Information Services

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, email orders@springer-ny.com, or visit <http://www.springer-ny.com>. Outside the United States: fax +49 6221 345229, email orders@springer.de, or visit <http://www.springer.de>.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit <http://www.apress.com>.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com> in the Downloads section.

.

Backing Up and Recovering Your Databases

IF YOU ARE INVOLVED with a multi-user Access database, you *must* learn how to back up the database properly. This chapter shows you how to check whether everyone has logged off, how to log off all users, how to back up a database properly, how to back up information while people are using the database, and then, when that fateful day comes, how to recover your information. As you progress through these topics, you will learn a good use for XML and even find out which objects in your database take up the most space.

The download material for this chapter includes forms and Visual Basic for Application (VBA) examples, as follows:

- Functions to find out whether anyone is using the database.
- A form to shut down the database at a set time.
- A form that will safely back up a database by using the compact process.
- A form that makes it easy for your users to compact their databases.
- A form to unload all tables to comma-delimited format.
- Two different ways to unload all tables to XML.
- A form to back up all objects to text.
- Special VBA recovery files that the text export procedures write automatically so that you can easily recover tables and objects.



NOTE To find the demonstration material, open the download database for your version of Access—for example, `grMAP2002.mdb`—and select Chapter 5. Before running any samples that involve data, you may need to relink the tables to the Northwind database on your computer.

What You Must Know About Database Backups and Recoveries

Once again, I will provide two overviews about the important issues covered in this chapter. The first provides an overview that's applicable to all IT professionals, because everyone should be concerned with backups. The second summary covers extra topics that are relevant to the developer.

Overview for the DBA, IT Manager, and the Developer

Backing up and recovering an Access database would be easy if you could instantly get everyone to log off the database just before the system backup was due to commence. This process used to be called tape backup, but these days, databases can be backed up to all sorts of media, such as portable hard drives and DVD burners. If people are using the database when your system backup runs, you risk copying a database that is in an unstable state, which could cause you problems when you need to recover the database. This chapter provides solutions to that issue by showing you how to:

- Make sure that your database is ready for a system backup.
- Show you how to back up your database to another location, including the new Access 2003 backup menu command.
- Provide insight into good backup procedures.
- Back up and recover individual tables while the database is in use by using both text and XML formats and objects that use the text format.
- Recover relationships, menus, and import/export specifications.

Of course, there are other reasons why you need to back up your databases, from computer equipment theft to the proverbial bus running over the laptop.

One extremely good reason to use some of the object- and table-exporting software is that someone may unknowingly delete something or lock something while implementing database protection. Because problems like these always lurk around the corner, the more alternatives that you have, the more chances that you have to save lost information.



CAUTION All the backup techniques described in this chapter can leave your security open to exploitation because you are creating yet more copies of the database and its objects for people to view. If you are serious about protecting and securing your backups, read Chapter 12 to find out how the operating system can protect the backup files. As part of this protection, you need to make sure that the Windows account that does the backups has write permissions for the backup directory.

Overview for the Developer

The gist of this chapter is to make it easy for you to incorporate backup routines into your own applications, and most of the material covered in this chapter will require that you integrate the demonstration material into a database. Apart from the benefits of the different backup processes discussed, you'll learn other important things:

- Different ways to identify all the tables and objects in your database.
- Another good use for XML.
- The undocumented and powerful `SaveAsText` and `LoadFromText` methods.
- Find out the very elusive facts about which are the biggest objects in your database, which is a very useful by-product of the object backup software in this chapter.

If that sounds like a lot of work, rest assured that most of the forms and functions from the demonstration database should not require too much customization when you add them to your database.

Now we will have a look in more detail about what you need to do to back up a database that has multiple users.

Backing Up Multi-User Databases

Access 97 or later

To back up an Access database correctly, every user must log off the database. If you back up a database when someone is using it, you risk saving the database in an unstable state. A user may have made changes to data and objects and not saved them, so that when you open the archived database, you may receive a message that states that the database is corrupt. You then will need to use the repair utility and, at best, only a small amount of information will be lost. Unfortunately, you will never be able to determine exactly what that was as any corrupted data is usually unrecoverable.



TIP See the “Further Reading” section at the end of this chapter for a Web reference on database corruption.

To ensure that the database is ready to be backed up, you must have exclusive access to the database. This condition does not apply to some of the exporting backup options discussed in this chapter, but it is a good idea nonetheless. One way to tell whether someone else is in the database is to check for the existence of a file with the same name as the database and an .LDB extension, which indicates an Access locking file. As long as you don't see this file, you should be able to open the database in exclusive mode. There are exceptions, however, which I will explain.



NOTE Chapter 6 discusses, at great length, different ways to find a list of users who are logged onto the database and even how to stop them from logging on to the database. Understanding these processes is important if you want to be the only person to have access to a database at a particular time. The .LDB locking file is discussed further in that chapter.

After you have exclusive access to the database, you can copy the file or export the information from the database. Before I describe some different ways to back up your database and data, I will show you how you can find out whether your database is being used.

Checking Whether Anyone Has the Database Open

The first and simplest way to determine whether someone's in the database is to check for an .LDB file with the same name as the database that you are using. You can check in Windows Explorer as follows:

1. Open Windows Explorer and navigate to the folder that your database is in.
2. Make sure that the display format of the folder is View Details or View List.
3. Sort the files in the display by file name.
4. Find the database and look for a file with the same name and the .LDB file extension.

The .LDB file is a good indicator of other people using the database, but sometimes a user turns off a computer or Windows crashes, and the .LDB file remains open. To cover for these contingencies, you can manually check whether you have exclusive access to a database by doing the following:

1. Open Access.
2. Choose File ► Open and navigate to the folder where your database is.
3. Select the file, click the Open button's drop-down arrow, and choose Open Exclusive (shown in Figure 5-1).

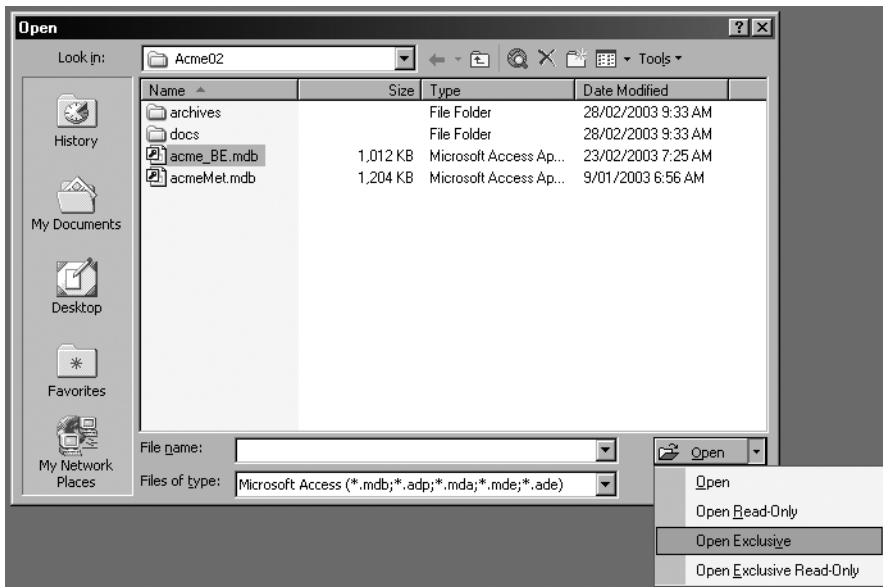


Figure 5-1. Testing whether you have exclusive access to a database.

If your database opens without any problem, then you can copy it to your backup media or compact it. If you are copying the file, you will need to close Access before doing the backup.

Unfortunately, these manual processes are a little tedious, and you may want to automate the process a bit more. To help with that, I have developed two equivalent functions that will tell you whether a database is being used.

Using VBA to Check Whether Anyone Is Using the Database

To find out whether someone is using a database, you will need to test whether you can open that database in exclusive mode. To try the demonstration form, open the sample database for the version of Access that you are interested in and choose Chapter 5 in the Demonstration Database Interface form. The first sample that I want to demonstrate is a form called `frmIsDBopenDAO`. Open this form in design mode because you may need to change the location of the Northwind database. The following code snippet demonstrates how you might use the `IsDatabaseOpen` function. If the (Northwind) database opens in exclusive mode, the function will return a `True` result.


```

' This form will test if it's possible to open a database exclusively.
Const MYDBPATH = "C:\Program Files\Microsoft Office\Office\Samples\northwind.mdb"
Dim myDbIsOpen As Boolean

myDbIsOpen = IsDatabaseOpen(MYDBPATH)
If myDbIsOpen Then
    MsgBox "Database is already open or an error occurred."
Else
    MsgBox "Database is not being used by anyone."
End If

```

The logic used in the `IsDatabaseOpen` function commences by opening a DAO workspace object. By using that workspace object, we then attempt to open a database reference in exclusive mode. If the exclusive reference fails, it returns an error. We can then check the error number to see why we couldn't open the database exclusively.

```

Function IsDatabaseOpen(strDbPath As String) As Boolean
' This function tests whether a database is open.

Const FILENOTFOUND = 3024
Const ALREADYOPEN = 3356
Const ALREADYOPENEXCL = 3045
Const DISKDOESNOTEXIST = 3043

Dim wsp As DAO.Workspace
Dim myDbs As DAO.Database

On Error GoTo IsDatabaseOpen_error
' Returns reference to default workspace.
Set wsp = DBEngine.Workspaces(0)
' Attempts to open an exclusive reference to another database.
Set myDbs = wsp.OpenDatabase(strDbPath, True)
' No one is using the database.
IsDatabaseOpen = False
Set myDbs = Nothing
Set wsp = Nothing

IsDatabaseOpen_Exit:

Exit Function

IsDatabaseOpen_error:

```

```

' Test for errors, which are probably caused by trying to open the
' database in exclusive mode.
IsDatabaseOpen = True
Select Case Err.Number

Case FILENOTFOUND
    MsgBox Err.Description, vbInformation, "File Not Found"
Case DISKDOESNOTEXIST
    MsgBox Err.Description & vbCrLf & vbCrLf & strDbPath, _
        vbInformation, "Disk does not exist"
Case ALREADYOPEN
    ' Opened by one or more people. One name appears in message.
    MsgBox Err.Description, vbInformation, "File Already Open"
Case ALREADYOPENEXCL ' Already opened exclusively by someone.
    MsgBox Err.Description, vbInformation, "File Already Opened Exclusively"
Case Else
    MsgBox "Error number " & Err.Number & " -> " & Err.Description
End Select
GoTo IsDatabaseOpen_Exit

End Function

```

When you try to use the `IsDatabaseOpen` function to open a database that someone is already using in shared mode, error number 3356 (signified by the constant `ALREADYOPEN`) returns an interesting error description. Unfortunately, because someone is using the database in shared mode, this description incorrectly says, “You attempted to open a database that is already opened exclusively by user ‘Admin’ on machine ‘MY COMPUTER.’ Try again when the database is available.” Alternatively, if you have opened the database in exclusive mode and then use the `IsDatabaseOpen` function, it will return a message that says, “The file is already in use” (and doesn’t mention it being exclusive at all).

I find this interesting because if you use the `IsDatabaseOpenADO` function, you will find in the `frmIsDBopenDAO` demonstration form that the equivalent code to check for exclusive access using the ActiveX Data Objects (ADO) library, the error descriptions returned are the same. Remember before running the code in the form that you will need to open this form in design mode because you may need to change the location of the Northwind database. That means that ADO is using exactly the same DAO calls to open the database. In addition, you can see that ADO returns only one error number for all the three errors provided in the DAO example. You then have to parse the error descriptions that accompany the numbers to establish what has gone wrong with the ADO exclusive open.

```
Function IsDBaseOpenADO(MyDBpath As String)
' This function tests whether a database is open
' by using the ADO Library.

Const ALREADYOPEN = -2147467259
Dim cnnDB As ADODB.Connection

On Error GoTo IsDBaseOpenADO_error

Set cnnDB = New ADODB.Connection

' Open database for shared (by default), read/write access, and
' specify database password.
With cnnDB
    .Provider = "Microsoft.Jet.OLEDB.4.0"
    .Mode = adModeShareExclusive
    .Open MyDBpath
End With

IsDBaseOpenADO = False
cnnDB.Close
Set cnnDB = Nothing

IsDBaseOpenADO_Exit:

Exit Function

IsDBaseOpenADO_error:
IsDBaseOpenADO = True
Select Case Err.Number
Case ALREADYOPEN
    ' Opened by one or more people. One name appears in message.
    If InStr(Err.Description, "not a valid path") Then
        MsgBox Err.Description, vbInformation, "File Not Found"
    ElseIf InStr(Err.Description, "file already in use") Then
        ' This database is opened exclusively.
        MsgBox Err.Description, vbInformation, "File Open Exclusively"
    Else
        ' This database is opened by other users.
        MsgBox Err.Description, vbInformation, "File Already Open"
    End If
```

```

Case Else
    MsgBox "Error number " & Err.Number & " -> " & Err.Description
End Select
GoTo IsDBaseOpenADO_Exit
End Function

```

After working through both of these samples in detail, I prefer the DAO function to the ADO function because the DAO function returns better error numbers.

So now that we have examined how to find the “in-use” status of a database, let’s see how to ensure that everyone logs off the database in time for the backup.

Setting Automatic Shutdowns Before Scheduled Backups

Unless you run a business that uses an Access database around the clock, there comes a time in the day when it should be safe to close the database. To make this easy, I have created a form called `frmAutoShutdown` that will shut down your Access database at a set time. To add this form to your database, import it and add a line to your `AutoExec` macro to open the form in hidden mode. From then on, the form will sit quietly in the background and check every few minutes to see if it is time to shut down the database. Just prior to the shutdown time, the form will issue a warning message to users. When the shutdown time arrives, the form will save any open objects or forms and then close the database. Because this process logs everyone off during the night, it is very useful for administering the database in the early morning.

To understand how the `frmAutoShutdown` form works—and possibly to change the setup—let’s review the full form module that follows. The most important thing to be concerned with is the values of the module constants. Generally, you will be setting the `SHUTDOWNHOUR` constant to a time when everyone is tucked into bed and just before you run your scheduled backups. You will need to remember to modify that constant to an appropriate time when you are experimenting with the form. After that, you will see the warning messages and the actual shutdown constants. If you look at the form timer event, you will find that the `MSGMINS` constant decides the interval between triggers. When the shutdown hour arrives, the `Quit` method will shut down the database.

```
Option Explicit
```

```

' User administration constants
' Purpose: Shut down and user messages.
Const MSGMINS = 3           ' Minutes between checking for system shutdowns.
Const SHUTDOWNFLAG = True   ' If True, then the system shuts down once a day.
Const SHUTDOWNHOUR = 0      ' Automatic shutdown hour (24-hour time).

```

```
Const WARNSTARTMINS = 0      ' Time in minutes that warnings start being issued.
Const WARNENDMINS = 10      ' Time in minutes that warnings stop being issued.
Const SHUTDOWNSTARTMINS = 10 ' Starting time in minutes for the shutdown.
Const SHUTDOWNENDMINS = 20  ' Final time in minutes for the shutdown.

Private Sub cmdOk_Click()
    Me.visible = False
End Sub

Private Sub Form_Load()

    ' Always hide this form; the user shouldn't know that it is there.
    Me.visible = False
    Me.TimerInterval = MSGMINS * 1000 * 60#

End Sub

Private Sub Form_Timer()

    ' Shuts down the database (in case anyone has left the database open).
    Dim myDate, myDownTime, myUpTime, myMessage, minsDiff As Integer

    On Error GoTo Quick_Exit

    If SHUTDOWNFLAG Then
        If Hour(Time()) = SHUTDOWNHOUR Then
            ' The time to shut down is nigh.
            If Minute(Time()) > SHUTDOWNSTARTMINS And Minute(Time()) < SHUTDOWNENDMINS Then

                ' Safely huts down the database , saving all open objects.
                Application.Quit acQuitSaveAll

                ElseIf Minute(Time()) > WARNSTARTMINS And Minute(Time()) < WARNENDMINS Then
                    Me.visible = True
                    lblMessage.Caption = "This database will close soon for administration."
                    Me.Caption = "Please Stop What You Are Doing."
                End If
            End If
        End If
    End If

Quick_Exit:

End Sub
```

Now, if you use this form or something similar to shut down the open databases, you will be sure to produce a cleaner backup. Next, let me put my own ideas about conventional backups into the melting pot.

Normal Backups

Access 97 or later

Any database that you are involved with should be backed up to an alternative storage system such as a tape drive or CD-ROM. This backup should be *systematic* (routine—not something you only think about when a crisis strikes—and, ideally, scheduled) and have the following characteristics:

- Performed on a regular basis.
- Kept off-site.
- Stored on a good-quality storage medium, such as a backup tape, memory card, CD, DVD, or portable hard drive.
- Kept in a secure and fireproof location.
- Multiple copies on multiple mediums.

Naturally, your databases need not be the only files kept as part of the backup system. Under no circumstances should the backup be kept only on-site on a sub-standard medium like a floppy disk or saved by using backup system software and hardware that your supplier no longer supports.

Recovering the Backups

More important than the backup itself is regularly testing the recovery process. Make a note in your diary to test recovering your database at least once a month if personnel, software, or hardware changes and once a quarter when the process is stable. If a database becomes corrupt early in the working day, don't try to repair the database. Instead, move that corrupted database to a safe area and restore the backup version from the night before. That way, you are really testing your backup processes.



TIP When it comes to testing backups, it's useful to remember the Boy Scout motto: Be prepared. I also recommend that you plan quarterly "fire drills," where you try to restore your latest backup.

Boutique Hardware Issues

When you set up your backup hardware, you need to be aware of the long-term support for your hardware device in these days when backup mediums come in many configurations. One good example of the sort of backup issue that can arise is the use of these new portable and large hard drives. If you were to keep the hard drive off-site and the power unit on-site, a fire on-site could cause you to lose both your computer and the power for the backup. You would then be forced to purchase an additional power unit from your original supplier. Having tried to do that on a one-year-old model last year, I can tell you that there is no guarantee that they will be in stock.

On-Site Backups

It is a good idea to produce a backup at least once a day of any database that you are working on or using. An Access database can become corrupt occasionally, especially if a large number of users are on it. In this case, a backup from computer hard drive to computer hard drive will suffice. The farther these computers are apart, the better. For both off-site and on-site backups, I use a product called Second Copy (see Figure 5-2), but many backup software packages around will work. I personally prefer to use a backup system that stores data in either its original file formats or in a popular compressed format, such as .ZIP.

Another additional safeguard that developers can use is sending a copy of the development work to your client on a regular basis. This way, your client gets to see your work, and it may cover you if your office or computer suffers a catastrophic event.

Now I will explain why compressed files and file versions are an important part of the developer's backup strategy.

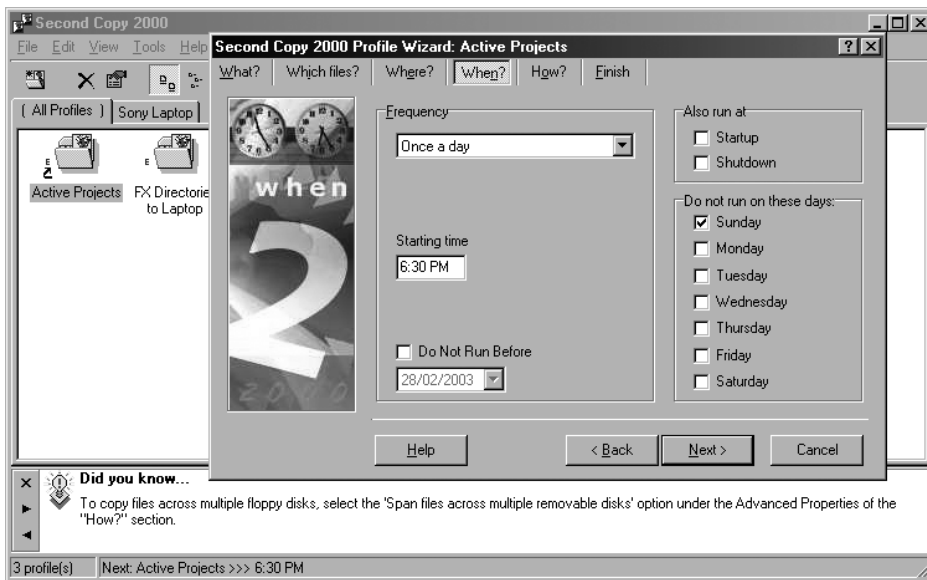


Figure 5-2. Backing up files on a regular basis is essential.

Creating Compressed (.ZIP) Archives

Probably the most vulnerable part of developing Access solutions is the developer. I personally am a little obsessed with cleaning up those objects in my database that just didn't work out. This routine occasionally leads to yours truly deleting the wrong object. Also, sometimes I will misunderstand a client's suggestions and make irretrievable alterations to a form or a module. So how do I get back to a previous point in the development cycle?

The answer is easy, and developers have used it since the beginning of computer time. It's called versions, and it's very simple to do. After you've made a number of alterations and you're happy with those changes, you give the database a version number. This number generally will be sequential and may involve major and minor version numbers or letters. For our business, we use the following procedure for front-end databases:

1. Update the version number on the startup form.
2. Save the database to a .ZIP or other type of compressed file with the same name as the database and a .ZIP file extension.
3. Rename the .ZIP file to include the version number.



TIP Always remember to make a brand new .ZIP file and then rename it. This action will ensure that you won't send the wrong version to a client. I once made the mistake of meaning to add a database to an existing .ZIP file but goofed up. The previous version of a database then went to the client for them to add data. When it came back, I replaced the development version with the client's older version.

To make a version archive, first make a new .ZIP file of your database, then rename the file to the version number shown by the files already in the folder (see Figure 5-3).

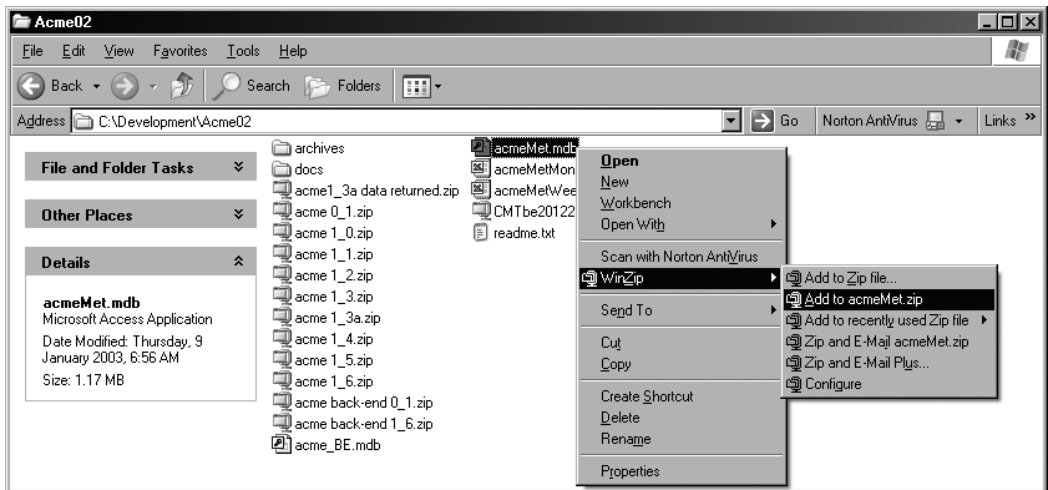


Figure 5-3. Adding the database to a .ZIP file.

With back-end databases, you have to be a lot more careful with managing both the changes and the archives because the client will have the latest data. What you really want to avoid is sending a copy of the back-end database back to the client and inadvertently having that file overwrite the live database. Always keep a copy of back-end databases that the client sends to you because the client could also have problems and might require a backup. Keeping multiple compressed versions of back-end databases whenever you change the data structure is a good idea. One exception to this rule is confidential information. You need to make sure that back-end databases that hold confidential information, such as credit card details, are not being stored on any computer other than those specially configured to protect that information.



TIP Compression systems generally will have the option of a password. You may want to use a password when transferring files by email or when saving confidential or important databases in an archive.

Now I will review how you can use database compacting to back up your databases.

Compacting Databases

Compacting is good to use as a backup mechanism because it ensures that you have full ownership of the file when you do the transfer. Before I explain this backup technique, I will first describe how to compact a database, as using the compact option regularly on your Access database is an important administrative task. To compact a database, first open the database that you want to use and then choose Tools ► Database Utilities ► Compact and Repair. For Access 97, where the compact and repair functions are separated, you should regularly run the Repair option as well because doing so may resolve an as-yet unreported problem in the database.



NOTE If you delete or modify tables or objects, your database will fragment. Fragmenting leads to a drop in performance and can be associated with database issues. Compacting from within a database makes a copy of the database, rearranging how the database file is stored on disk. Once that file is compacted, the new file replaces the existing database.

In installations where the DBA is actually an end user, I find that no matter how many times I ask the reluctant DBA, he or she forgets about the compacting option. To help resolve this issue, I will usually add a form to the database that will make compacting easier. I have called this form `frmCompactEasy` in the demonstration databases. It works by using the `SendKeys` method to send the keystrokes that drive the Compact Database menu option.

```
' Use the SendKeys command to compact the database safely.
SendKeys "%TDC"
```

Compacting makes your database faster and more stable, and the following section explains how to use compacting to back up your database.

Using Compacting to Back Up Your Database

Access 97 or later

If you have used Access 2 or Access 95, you will remember that compacting databases was actually a three-step process. First, you had to use the compact option to make a new compacted database. Then you had to move the existing file to another folder, and finally you had to rename the compacted database back to the same name as the existing database. Naturally, compacting was not so popular in those days.

Even though the old method isn't used for compacting much these days, it can come in handy for doing backups. To use it, we will back up the database by compacting from the current database to a new one in a different location. To find this option, you will need to start Access without selecting any database and then choosing **Tools ► Database Utilities ► Compact and Repair Database**, as shown in Figure 5-4.

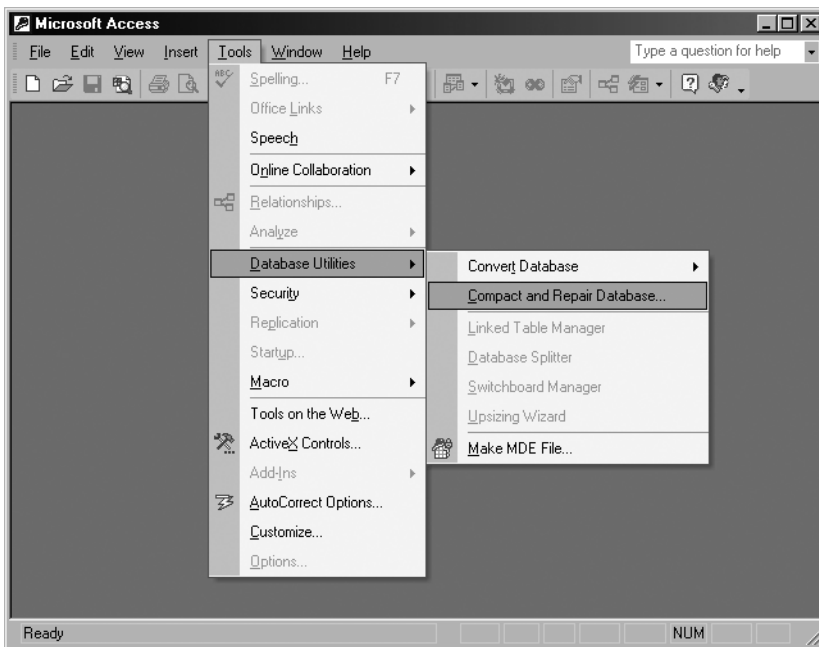


Figure 5-4. Choosing to compact a database for backup purposes.

You will then need to select the database (by using the File dialog) that you want to compact. After that, enter the name and location of the database that you want to backup and compact into (as shown in Figure 5-5). I suggest that you

always give your backups a name that denotes it as a backup, and you should store it in a different folder so that the databases are not mixed up. I also advise that you save your backups in a folder that is difficult to find or protected from users by operating system permissions.

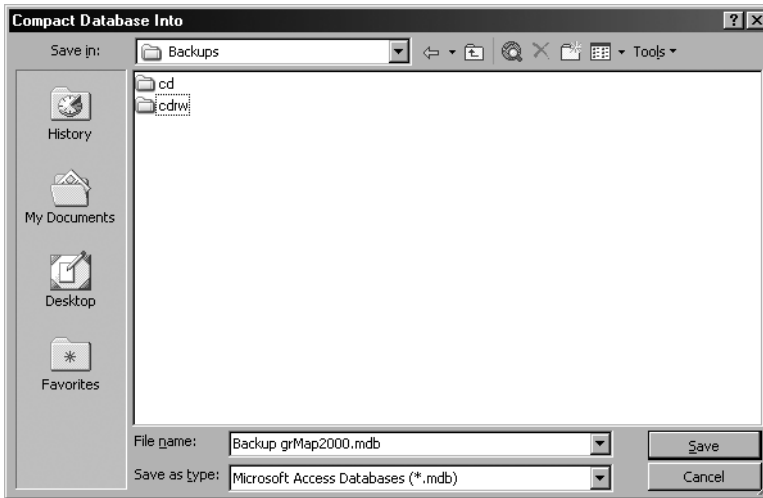


Figure 5-5. Selecting a new folder and entering a name that reflects that the database is a backup.

In Access 2002, the Application object gained a new method that made it easier to back up from one database to another. The next section demonstrates this method.

Compacting and Repairing a Database for Backups by Using VBA Code

Access 2002 or later

Access 2002 introduced a new VBA method—the CompactRepair method—that allows you to compact from one database to another. I’ve adopted this method as part of a function that will run the compacting backup process for you. You will find this code in the sample database in a form called frmCompactDatabase. When using this code, you will need to change the values of the constants to suit your own database and backup naming preferences.

```
Private Sub cmdCompactAnother_Click()
Dim compactOK As Boolean, backupName As String
Const COMPACTFROM = _
    "C:\program files\Microsoft Office\Office 10\Samples\northwind.mdb"
Const COMPACTTO = "C:\temp\northwind.mdb"

compactOK = CompactToNewMDB(COMPACTFROM, COMPACTTO, True)
If compactOK Then
    MsgBox "A newly compacted database called " & COMPACTTO & " has been created."
Else
    MsgBox "Compact was unsuccessful."
End If

End Sub
```



CAUTION The CompactToNewMDB function, described following, deletes an existing file if you set the deleteMdbToFirst argument to True. You should experiment with this function on an unimportant database before implementing it for your backup procedures.

The CompactToNewMDB function deletes the database before compacting is undertaken. If the database already exists in the location to which you want to compact, the Access CompaqRepair method will fail. For this reason, I have incorporated an optional deleteMdbToFirst argument to allow you to delete the destination database first. To run this process to generate multiple backups of your database, I suggest that you incorporate the date and time into the new database file name to make it unique.

```
Function CompactToNewMDB(mdbFrom As String, mdbTo As String, _
Optional deleteMdbToFirst As Boolean) As Boolean
' Input values: the paths and file names of the source and destination files
' plus a Boolean flag that will delete your database if set to True.

On Error GoTo CompactToNewMDB_Error

If IsMissing(deleteMdbToFirst) Then
    deleteMdbToFirst = False
End If
```

```
If deleteMdbToFirst Then

    If Len(Dir(mdbTo)) > 0 Then

        ' Delete the database that you are compacting to.
        Kill mdbTo
    End If
End If

' Compact and repair the database. Use the return value of
' the CompactRepair method to determine if the file was
' successfully compacted.
CompactToNewMDB = Application.CompactRepair(mdbFrom, mdbTo)

CompactToNewMDB_Exit:
Exit Function

CompactToNewMDB_Error:
' You can add your own error handling code here.
Select Case Err.Number
Case Else
    MsgBox "Error number " & Err.Number & " -> " & Err.Description
End Select

Resume CompactToNewMDB_Exit

End Function
```

If you are using Access 97 or 2000, you will find examples of how to compact from one database to another in the VBA help. Search for the term “CompactDatabase” in the Visual Basic Editor.

In the remainder of the chapter, we will concern ourselves with backing up tables and objects from the database into text format.

Saving Tables to Text Files

This section explains the benefits of exporting data into text files and the methods of recovering that data into a (new) database. At anytime of the day, you can perform “live” extracts of all the information from a table to a text file. Exporting to text files is a good idea for the following reasons:

- You can generally do it at any time of the day, whether or not users are in the database. Only the currently saved records will be transferred. Keep in mind, though, that it would be better to rely on information that this software extracts when everyone has logged off, because getting information only from saved records may cause concurrency and contention issues.
- Long-term storage of data in text files will allow a DBA to read the information into multiple products. These text files probably will require accompanying documentation in a simple format so that the person who recovers the data does not have to guess the design of the tables that have been exported.
- Data exports are also a good idea to do prior to adding protection, such as passwords, to your databases.

Right from my early days as an Informix DBA, I learned that it was a good idea to export the tables in databases to text files. I like to do this so that I always know that a DBA looking at the text backups sometime in the future will be able to read the text files by using a text editor and import them into another system. What is more problematic is that you may not have a program that can read the database. As an example, contemplate how you would read information from an Access 97 database that you stopped using in 2002. When you pull the database from the archives and try to open it in 2008 by using the latest version of Access, you may unfortunately find that it doesn't support the Access 97 format. If you think that is unlikely, Microsoft has scheduled to remove Access 97 from its Web site sometime after the start of 2005. Though I don't imagine that Microsoft will drop support for Access 97 files for a long time, you never know. For more details on the support timetable for Microsoft products, see the links in the "Further Reading" section at the end of this chapter.

Saving Tables as Comma-Delimited Text Files

Access 97 or later

To demonstrate how you can export tables to text, use the following example to export all the tables in a database to a comma-delimited text format like that shown in Figure 5-6. You will find this sample under the only button in the details section of the download form called frmGR8_unloadAll. The sample works by first establishing the export folder. In this case, it will create a subfolder called Unload directly under the folder where the database is located. Then a DAO TableDef collection is established, and a loop is used to cycle through all the data tables in the collection. The TransferToText method then exports the table to comma-delimited format.

If you open one of the comma-delimited files, it may display in either a text editor or Excel, depending on your file type associations in Windows Explorer.

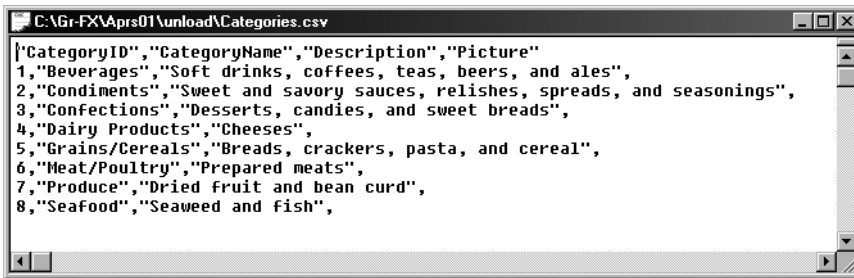


Figure 5-6. The exported comma-delimited file viewed in a text editor.



User Story *Exporting to text files means that, in all likelihood, you still will be able to read the files in 20 years—that's as long as CD-ROMs, tapes, or other such media will still be readable. Moreover, if you think that long-term*

recovery is unlikely, listen to this story. One project that I was involved with had a database of geological data that cost \$50 million to put together. Another company bought the project, did nothing with the data for five years, and in the end couldn't read the backup tapes. When we were asked to help, we managed to recover the text data backups from our tapes, and these were used to build a reasonable copy of the database. We also recovered the database from tapes, but the format was proprietary, and the software that could read the proprietary format was long gone.

The following example exports all the tables in a database to a comma-delimited text format.

```
Private Sub unload_all_Click()

' This form requires a reference to
' Microsoft DAO 3.6 or 3.51 library.

Dim i As Integer, unloadOK As Integer
Dim MyTable As DAO.TableDef
Dim MyDB As DAO.Database, MyRecords As DAO.RecordSet
Dim filen As String, unloadDir As String
```



```
' See Microsoft Knowledge Base Article 306144 if you want to
' change the following file type.

Const UNLFILETYPE = ".csv"
Const UNLSUBFOLDER = "unload\"

On Error GoTo unload_all_Failed

unloadDir = GetDBPath_FX & UNLSUBFOLDER

Set MyDB = CurrentDb
If Len(Dir(unloadDir, vbDirectory)) = 0 Then

    unloadOK = MsgBox("All tables will be unloaded to a new directory called " & _
        unloadDir, vbOKCancel, "Confirm The Unload Directory")
    If unloadOK = vbOK Then
        Mkdir unloadDir
    Else
        GoTo unload_all_Final
    End If
End If

' Loop through all tables, extracting the names.

For i = 0 To MyDB.TableDefs.Count - 1
    Set MyTable = MyDB.TableDefs(i)

    ' Create the file name as a combination of the table name and the file type.

    filen = unloadDir & MyTable.Name & UNLFILETYPE
    If left(MyTable.Name, 4) <> "Msys" And left(MyTable.Name, 1) <> "~" Then

        ' Not an Access system table.
        'Export data in comma-delimited format with column headers.

        DoCmd.Echo True, "Exporting table " & MyTable.Name & " to " & filen
        DoCmd.TransferText A_EXPORTDELIM, , MyTable.Name, filen, True

    End If

Next i
```

```

MsgBox "Unloaded all tables to ... " & unloadDir, 64, "Unloaded Tables"

unload_all_Final:
Exit Sub

unload_all_Failed:
' Problems with unloading.
Select Case Err.Number
Case Else
MsgBox "Error number " & Err.Number & " -> " & Err.Description, _
vbCritical, "Problem unloading tables"

End Select
Resume unload_all_Final:

End Sub

```



CAUTION Not all exported tables will import into Access successfully from the comma-delimited format. This does not mean that there is any problem with the file. Generally, it means that there is a problem with the system that Access uses to predict what the field type of a column is in these external files. For example, when a decimal field is blank for the first 20 lines of the unload file, decimal number columns can be incorrectly classified as integer numbers. Once that incorrect classification has been made, for the rest of the import process, the decimal numbers are rounded to integer values. Also, some text fields can contain unusual characters (like ' or " or foreign languages enunciations), which will unload from the tables with unusual results. Generally, though, this type of data exporting is worth trying first because it is simple and because most systems will read comma-delimited files.

In the preceding code, a function called `GetDBPath_FX` helps to establish a sub-directory below where the current database exists.

I have made previous references to backups only being as good as the recovery process, so I now will show you how to recover a comma-delimited text (.CSV) file.

Recovering Data from a Comma-Delimited Text File

In the following example, I will show you how you can recover data from a comma-delimited text file. The steps to recover the data from the `categories.csv` file (shown previously in Figure 5-6) follow.

1. Create a new blank Access database by choosing File ► New Database.
2. Choose File ► Get External Data ► Import.
3. Choose files of type Text Files (*.TXT, *.CSV, *.TAB, or *.ASC) and click Import.
4. Navigate to the correct folder by using the Look In box and select the file.
5. On the first window of the Import Text wizard, choose Delimited and then click Next.
6. On the second window of the wizard, make sure Comma is selected as the type of delimiter, select the First Row Contains Field Names check box (as shown in Figure 5-7), and then click Next.
7. On the third window of the wizard, choose to load the file into a new table.
8. Accept the Access defaults by using the Next buttons, and on the last window, click Finish.

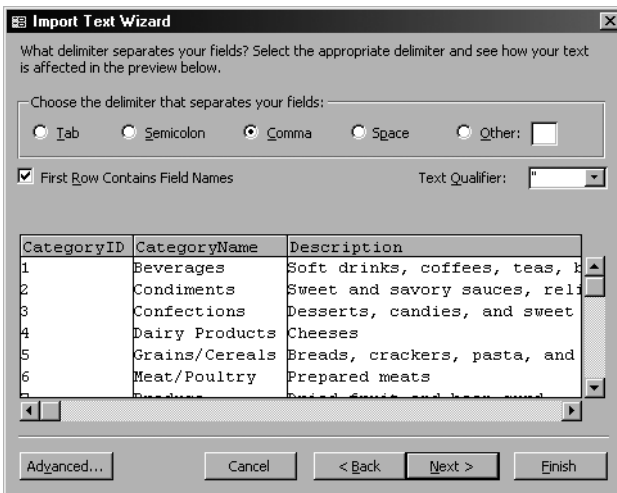


Figure 5-7. The second window of the Import Text wizard.

Now that you have the data loaded into a database, you can append, merge, or replace existing data tables as you see fit. Choosing whether to append, merge, or replace data would be specific to your own data structures and is not within the scope of this book.

Comma-delimited files can prove troublesome if you use them to recover from long-term storage if they are stored without documentation. To alleviate this risk, I recommend that you also store information about the structure of the data in the same location as the text files. Thankfully, there is now a popular new text standard called XML that will provide a more documented format for long-term storage of text files. I will now explain how you can instruct Access to use that format.

Using Access 2002 XML Methods to Export Tables

Access 2002 or later

As I explained in the section on backing up to comma-delimited text files, there are some issues with recovering data from that format. A better way to back up tables to text files is by using the eXtended Markup Language (XML). Access 2002 provides a good platform to get involved in XML because you can easily save or retrieve tables and queries from the XML format by using the ExportXML and ImportXML methods of the application object. In every XML file that Access generates, you will find not only the data but descriptions of the structure of the data as well. This addition is an improvement on such text files as the comma-delimited format because there is no more guessing whether a field is text or integer or decimal. Figure 5-8 shows an XML file created from the Northwind Categories table.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <root xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" xmlns:od="urn:schemas-microsoft-com:officedata">
- <xsd:schema>
+ <xsd:element name="dataroot">
- <xsd:element name="Shippers">
+ <xsd:annotation>
- <xsd:complexType>
- <xsd:sequence>
+ <xsd:element name="ShipperID" od:jetType="autonumber" od:sqlType="int"
+ <xsd:element name="CompanyName" minOccurs="0" od:jetType="text"
+ <xsd:element name="Phone" minOccurs="0" od:jetType="text" od:sqlType="nvarchar">
+ <xsd:sequence>
+ <xsd:complexType>
+ <xsd:element>
+ <xsd:schema>
- <dataroot xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
- <Shippers>
+ <ShipperID>1</ShipperID>
+ <CompanyName>Speedy Express</CompanyName>
+ <Phone>(503) 555-9831</Phone>
+ </Shippers>
- <Shippers>
+ <ShipperID>2</ShipperID>
+ <CompanyName>United Package</CompanyName>
+ <Phone>(503) 555-8199</Phone>
+ </Shippers>
- <Shippers>
+ <ShipperID>3</ShipperID>
+ <CompanyName>Federal Shipping</CompanyName>
+ <Phone>(503) 555-9931</Phone>
+ </Shippers>
+ </dataroot>
</root>
```

Figure 5-8. The Shippers table from the Northwind database in XML format.

In the VBA example provided in the sample database, the procedure shown in the following code will export all the tables in the database to XML files. In this instance, I use the ADO Extensions for DDL and Security library (ADOX) to provide a list of all the tables in the database.

I use ADOX in this example because the database object types have properties that allow you to establish easily whether a table is linked or is normal (nonsystem). As the loop progresses to each table in the tables catalog, the ExportXML method is used. The code under the only button on the form called frmUnload2002xml demonstrates this.

```
Private Sub cmdUnlXML_Click()

' Set a reference for
' Microsoft Active X Data Object 2.5 library.
' Microsoft ADO Extensions 2.5 for DDL and security.

Dim tblsExported As String, cancel As Integer
Dim objT As ADOX.Table, objV As ADOX.View
Dim io As Integer, unloadOK As Integer
Dim adoxCat As ADOX.Catalog
Dim filepath As String
Dim xmlFolder As String, tableName As String
Const REBUILDFILE = "_RebuildMdbTables.txt"
Const INCLUDESCHEMA = 1

xmlFolder = GetDBasePath_FX & "backupXml\"
If Len(Dir(xmlFolder, vbDirectory)) = 0 Then

    unloadOK = MsgBox("All tables will be unloaded to a new directory called " & _
        xmlFolder, vbOKCancel, "Confirm the Unload Directory")
    If unloadOK = vbOK Then
        Mkdir xmlFolder
    Else
        GoTo cmdUnlXML_Exit
    End If
End If

io = FreeFile
Open xmlFolder & REBUILDFILE For Output As io

Print #io, "public sub RebuildTables"
Print #io, ""
Print #io, "' Generated by software written by Garry Robinson"
Print #io, "' Import this into a blank database and type"
```

```

Print #io, "' call RebuildTables "
Print #io, "' into the Immediate Window"
Print #io, ""
Print #io, "msgbox ""This will a load a number of XML files into new tables. "" , _"
Print #io, ", vbInformation"

Print #io, ""

On Error Resume Next
Set adoxCat = New ADOX.Catalog

adoxCat.ActiveConnection = CurrentProject.Connection

tblsExported = "Tables that were exported to " & xmlFolder & vbCrLf & vbCrLf
txtXMLFile.Visible = True
For Each objT In adoxCat.Tables

    If objT.Type = "Table" Or objT.Type = "Link" Then
        ' Queries are ignored in the exporting process.
        tblsExported = tblsExported & objT.Name & vbCrLf
        tableName = objT.Name
        DoCmd.Echo True, xmlFolder & tableName & ".xml"
        filepath = xmlFolder & tableName & ".xml"
        ' Export the table to an XML file.
        txtXMLFile = tableName
        ' Save the table as an XML file.
        ExportXML acExportTable, tableName, xmlFolder & tableName & ".xml", _
            , , , INCLUDESCHEMA

        Print #io, "importXML "" & filepath & """, acStructureAndData"
    End If
Next objT

On Error Resume Next

Print #io, "msgbox ""End of table import from XML""
Print #io, ""
Print #io, "end sub"

```

```

Close io
MsgBox tblsExported, vbInformation, "End Of Exports"
Set adoxCat = Nothing

cmdUnlXML_Exit:
End Sub

```

One good thing about the implementation of XML in Access 2002 is that you can export and import by using XML without truly understanding the format itself. The XML exports and imports work well because they correctly handle issues like unusual characters and bitmaps. In the next section, I show you how you can recover your XML files.

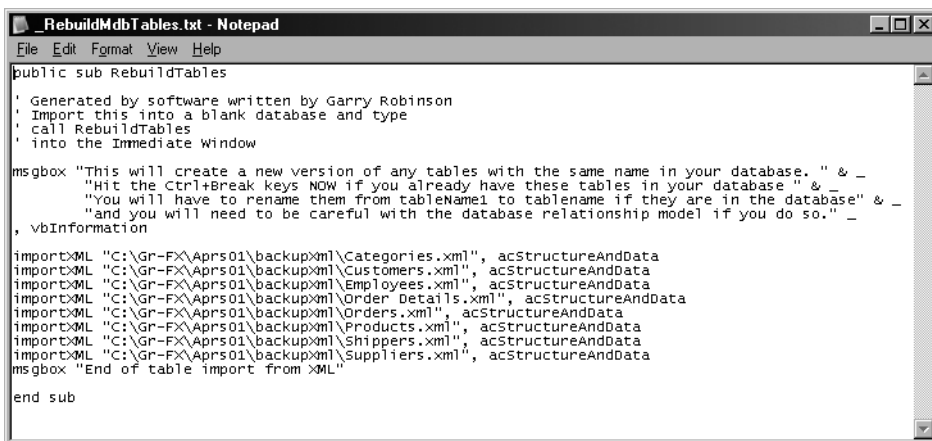
Recovering Data from an Access 2002 XML File

Access 2002 or later

You can use the Import command on the File menu to recover a table that has been exported to XML. Notice that when you import an XML file into a database that already has a table with the same name, you will create a table with the same name plus a numerical suffix, such as TableName1. The process for importing a single XML file in Access 2002 or later follows:

1. Choose File ► Get External Data ► Import.
2. Choose XML documents (*.XML, *.XSD) from the Files of Type drop-down list.
3. Choose the XML file, then click Import, and then click OK.
4. Be wary of choosing the Append to Existing Data option because you will need to check that all the records are new before you load them.

This process will work for a few files, but if you want to recover a number of tables, you may want to automate this process by using the VBA code recovery file generated by the preceding piece of code (found in form `frmUnload2002xml`). As part of the export process, this form generates a table recovery file (shown in Figure 5-9) that you can use to load all the XML files into a blank database.



```

public sub RebuildTables
' Generated by software written by Garry Robinson
' Import this into a blank database and type
' call RebuildTables
' into the Immediate Window

msgbox "This will create a new version of any tables with the same name in your database." & _
      "Hit the Ctrl+Break keys NOW if you already have these tables in your database" & _
      "You will have to rename them from tableName1 to tablename if they are in the database" & _
      "and you will need to be careful with the database relationship model if you do so." _
, vbInformation

importXML "C:\Gr-FX\Aprs01\backupxml\Categories.xml", acStructureAndData
importXML "C:\Gr-FX\Aprs01\backupxml\Customers.xml", acStructureAndData
importXML "C:\Gr-FX\Aprs01\backupxml\Employees.xml", acStructureAndData
importXML "C:\Gr-FX\Aprs01\backupxml\Order Details.xml", acStructureAndData
importXML "C:\Gr-FX\Aprs01\backupxml\Orders.xml", acStructureAndData
importXML "C:\Gr-FX\Aprs01\backupxml\Products.xml", acStructureAndData
importXML "C:\Gr-FX\Aprs01\backupxml\Shippers.xml", acStructureAndData
importXML "C:\Gr-FX\Aprs01\backupxml\Suppliers.xml", acStructureAndData
msgbox "End of table import from XML"

end sub

```

Figure 5-9. The text file that can reload the tables into a blank database.

Here are the steps that will rebuild all the tables from the Access 2002 XML backup process:

1. Create a new blank Access database by choosing File ► Create Database.
2. Open the Visual Basic Editor by pressing ALT+F11.
3. Choose File ► Import File.
4. Choose all file types (*.*).
5. Find the table recovery file called _RebuildMdbTables.txt and click the Open button.

You've now created a module in your blank database called Module1. Review the code in this module. Now you can open the Immediate window by pressing CTRL+G or by choosing View ► Immediate Window. Type the following into the Immediate window to reload the tables:

```
call RebuildTables
```

If you are using Access 2000 or even Access 97, you can take advantage of XML by using the ADO 2.5 library to generate the XML. The next section describes a form that will show you how to do this.

Using ADO to Generate XML Files in Access 2000

Access 2000 or later

Long before I developed the automated XML export code discussed in the section “Using Access 2002 XML Methods to Export Tables,” I used ADO 2.5 to export tables to text files. Using XML in this way isn’t anywhere near as integrated into Access as the ExportXML method in Access 2002, but it does provide a way to generate XML files for backup. If you want to try this particular approach, open the demonstration database for Access 2000 (grMap2000.mdb) and choose the form called frmUnloadADOxml. This form has all the necessary code under the one button and is very similar in structure to that code discussed earlier in the chapter and demonstrated in the section on using Access 2002 XML methods to export tables.

The following code snippet illustrates how the ADO Recordset object’s Save method saves a table to an XML file. Before exporting a database, you must delete the existing XML file by using the Kill function. The code snippet provided requires that you upgrade to at least version 2.5 of the ADO library (Microsoft ActiveX Data Object 2.5 library) and include this reference in your Access application.

If this method of exporting appeals to you, you can also use a proprietary and more compact Microsoft format to store the exported information. Select this format by changing the adPersistXML constant in the second argument of the Recordset object’s Save method to use the adPersistADTG constant. This adPersistXML format may even appeal to you as a security precaution because the file format is binary and cannot be read in a text editor. Another advantage of the adPersistXML constant is that it will allow you to use the ADO 2.1 library that comes with Access 2000.

```
Set rst = New ADODB.RecordSet
rst.ActiveConnection = CurrentProject.Connection
datasource = objT.Name
rst.Open datasource

On Error Resume Next
Kill cachedir & datasource
rst.Save cachedir & datasource & ".xml", adPersistXML
Set rst = Nothing
```

Now that you have seen a number of ways to save a table to a text file, the next section will show you how to save code and objects to text files and then recover the objects at a later date if necessary.

Exporting and Recovering Programming Objects

Access 97 or later

In this section, I will show you how to save queries, forms, reports, macros, and modules (which I'll call programming objects from now on) to text files. I encourage this particular backup approach because it offers additional recovery opportunities and helps you in these specific situations:

- If more than one person is developing software for the database, your systems for cooperating will not always be perfect, and someone's good hard work will be lost.
- If an object in a database becomes irretrievably damaged.
- If you're adding protection to the database, such as passwords and user-level security.

In all these scenarios, backing up the programming objects at regular intervals may help recover the object. Now I will describe how to save VBA code by using a menu.

Exporting a VBA Module to a Text File

Access 97 or later

Your VBA modules and class modules are the only objects in the database for which you can use a menu to save as text and recover from text.

1. Open the Database window (of any database with code).
2. Select Modules.
3. Select an individual module.
4. Choose File ► Export.
5. Save the file as a text (*.txt).
6. Select your folder and file name.
7. Select the Autostart check box and click Export.

Your VBA module will now be saved in a text file. In Access 2000 and later, you can also export modules from within the Visual Basic Editor. First, you need to

select the object that you want to export in the Project Explorer, then you choose File ► Export, and the steps are similar to the preceding procedure. If you use the Visual Basic Editor to save the code, you can use more specific file types like *.BAS and *.CLS rather than *.TXT.

Importing the VBA Text File Back into the Database

In Access 97, the only way to import the VBA text file back into the database is to open the text file in a text editor like Notepad, select all, copy to the paste buffer, open a new module in Access, and paste the code into the new module. You can use this simple method in all versions of Access.

In Access 2000 and later, however, you can also import the VBA text file directly into the database by following these steps:

1. Open a new database.
2. Open the Visual Basic Editor (press ALT+F11).
3. Choose File ► Import File.
4. Find the file and click Open.
5. You will now find the module in the Project Explorer. If it is not visible, choose View ► Project Explorer.

The next section of this chapter will show you how to use the same formats that Microsoft Visual SourceSafe uses to save and recover all the programming objects in your database.

Saving and Retrieving Objects by Using Hidden Methods

Access 97 or later

Unless you suffer from the same compulsive Web-searching disorder that I am afflicted with, you will probably be blissfully unaware that it is possible to save all your queries, forms, reports, macros, and modules to text files. So what, you might ask? Saving objects as text files means that you have a copy of an object that is external to any database that you are developing. If an object is inadvertently changed, you can retrieve that object from the text file. Once you understand the concepts behind saving objects to text, you will find many possible uses for it, such

as using the files to interchange objects between developers and recovering objects from corrupt databases.

To import and export programming objects, we can use two undocumented (hidden) methods of the Application object¹ called `LoadFromText` and `SaveToText`. These methods both require you to specify the object type by using the Access constants, the name of the Access object, and the destination or retrieval location of the file. To illustrate this process, open the Immediate window (press CTRL+G) in the Visual Basic Editor and type `Application.SaveAsText`. From then on, Intellisense will provide you with the list of constants and arguments to complete the statement. In Figure 5-10, I have put together an example to show how you can export and import a form.

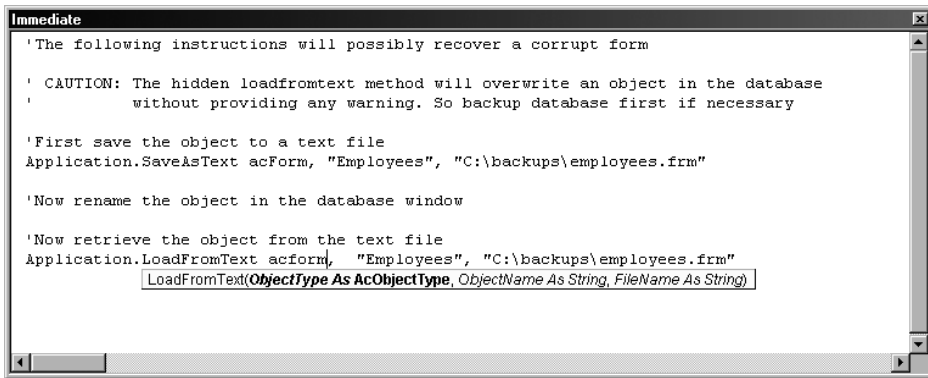


Figure 5-10. The `SaveAsText` and `LoadFromText` methods viewed in the Immediate window.

Exporting All Programmable Objects to Text Files

A good way to describe backing up objects to text is to work through an example from the demonstration database. This demonstration will create a file for each programmable object in the database. You will be able to run this utility even if other people are using the database. This example will coexist well with the exporting of all tables to text example described earlier in this chapter.

1. To show hidden members of objects such as the Application object, open the Object browser from the Visual Basic Editor. Right-click any object and choose Show Hidden Members from the menu.

To experiment with this download, do the following:

1. Make a copy of Northwind.mdb and then open the copy.
2. Import the following objects from the demonstration database:
frmBackupAllObjects, basGR8_exportObjects, and basGR8_Startup.
3. Compile all the code by using the Debug menu in the Visual Basic Editor.
4. Open the form frmBackupAllObjects (shown in Figure 5-11) and click the Back Up All Objects to Text button to start the backups.

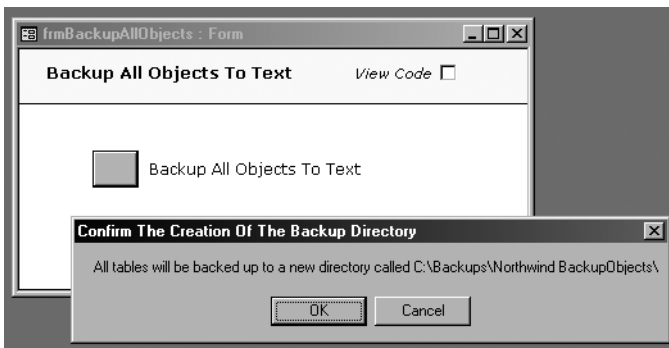


Figure 5-11. The Back Up All Objects to Text form.

5. When the exports are completed, a message box (as shown in Figure 5-12) will tell you where the text copies of the objects are. It will also tell you the name of a text file that you can use to recover all the objects into a blank database by using VBA.



Figure 5-12. The message that appears to tell you where the files went and how to recover them.



TIP Before exporting a database or shipping it to clients, for that matter, it is wise to compile all modules in the database. To do this, open the Visual Basic Editor and choose **Debug ► Compile Project** for Access 2000 and later and **Debug ► Compile All Objects** for Access 97.

Now that the exports are complete, let's have a look at what has happened. All the objects are now stored in text files in a subfolder where the current database is located (shown in Figure 5-13). The file types used for saving the files are *.QRY for queries, *.FRM for forms, *.RPT for reports, *.MCR for macros, *.BAS for modules, and *.CLS for class modules.

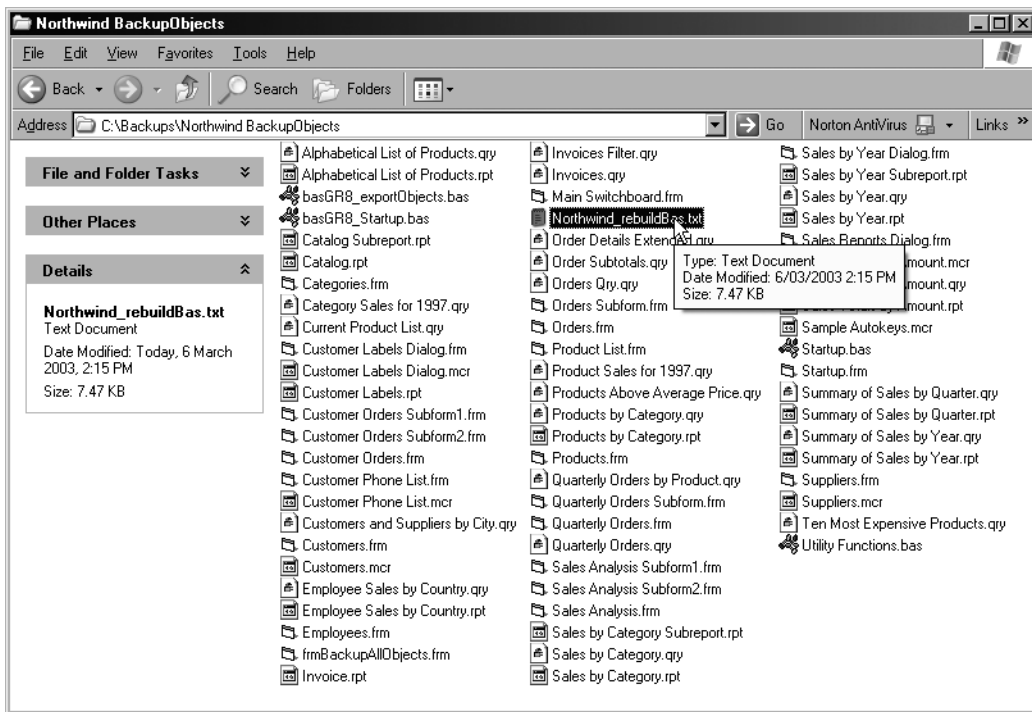


Figure 5-13. Folder showing all the objects exported to individual files.

To gain an understanding of the structure of the exported objects, let us look at the text version of a query that was created by the SaveAsText method (as shown Figure 5-14).



```

Product Sales for 1997.qry - Notepad
File Edit Format View Help
dbMemo "SQL" = "SELECT Categories.CategoryName, Products.ProductName, Sum(CCur([Order Details].U"
"nitPrice*[Quantity]*(1-[Discount])/100)*100) AS ProductsSales, \"Qtr \" & DatePar"
"t(\"q\",[ShippedDate]) AS ShippedQuarter\015\012FROM (Categories INNER JOIN Prod"
"ucts ON Categories.CategoryID=Products.CategoryID) INNER JOIN (Orders INNER JOIN"
"[Order Details] ON Orders.OrderID=[Order Details].OrderID) ON Products.ProductID"
"=[Order Details].ProductID\015\012WHERE ((Orders.ShippedDate) Between #1/1/199"
"7# And #12/31/1997#))\015\012GROUP BY Categories.CategoryName, Products.ProductN"
"ame, \"Qtr \" & DatePart(\"q\",[ShippedDate]);\015\012"
dbMemo "Connect" = ""
dbBoolean "ReturnsRecords" = "-1"
dbInteger "ODBCTimeout" = "60"
dbBoolean "FilterOn" = "0"
dbText "Description" = "Record source for Category Sales for 1995 query. Uses Sum and CCur functions."
dbBoolean "OrderOn" = "0"
dbByte "DatasheetGridlinesBehavior" = "3"
dbBoolean "OrderByOn" = "0"
dbByte "RecordsetType" = "0"
dbByte "Orientation" = "0"
dbByte "DefaultView" = "2"
Begin
Begin
dbText "Name" = "Categories.CategoryName"
dbInteger "ColumnWidth" = "1620"
dbBoolean "ColumnHidden" = "0"
End
Begin
dbText "Name" = "Products.ProductName"
dbInteger "ColumnWidth" = "3210"
dbBoolean "ColumnHidden" = "0"
End
Begin
dbText "Name" = "ProductsSales"
dbInteger "ColumnWidth" = "1410"
dbBoolean "ColumnHidden" = "0"
dbMemo "Caption" = "Product Sales"
dbText "Format" = "$#,##0.00;($#,##0.00)"
End
Begin
dbText "Name" = "ShippedQuarter"
dbInteger "ColumnWidth" = "1665"
dbBoolean "ColumnHidden" = "0"
End
End

```

Figure 5-14. The Product Sales query after being exported to a backup text file.

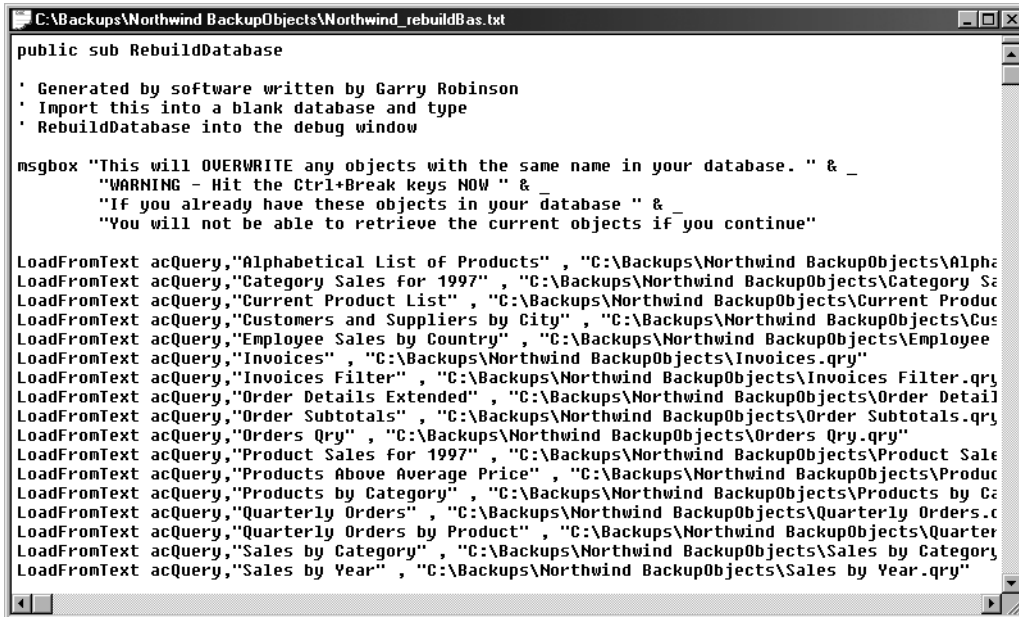
As you can see, not only is the SQL stored, but the column properties, field types, and other details are also stored in the file. It is this complete detail that allows Access to import the objects that are exported, which allows us then to recover a damaged or deleted object.

Okay, so now we have created all the text copies of the database; what use are they if we cannot recover them? To assist in loading all these files back into a database, the export process generated an object recovery file. This file has all the necessary VBA code to import the objects back into an empty database, which I will now describe how to do.

Importing All Programmable Objects into a Blank Database

Retrieving all or some of these objects back from a folder requires you to create VBA code by using the Application object's LoadFromText method once for every object in the database. Writing this sort of code manually for even a small database such as Northwind would be very tedious. To automate this process, the Export All

Objects software automatically generates a text file called Northwind_rebuildBas.txt (shown in Figure 5-15). This file contains VBA code that will load all the objects into a blank database.



```

C:\Backups\Northwind BackupObjects\Northwind_rebuildBas.txt

public sub RebuildDatabase

' Generated by software written by Garry Robinson
' Import this into a blank database and type
' RebuildDatabase into the debug window

msgbox "This will OVERWRITE any objects with the same name in your database." & _
"WARNING - Hit the Ctrl+Break keys NOW " & _
"If you already have these objects in your database " & _
"You will not be able to retrieve the current objects if you continue"

LoadFromText acQuery,"Alphabetical List of Products" , "C:\Backups\Northwind BackupObjects\Alpha
LoadFromText acQuery,"Category Sales for 1997" , "C:\Backups\Northwind BackupObjects\Category Sa
LoadFromText acQuery,"Current Product List" , "C:\Backups\Northwind BackupObjects\Current Produc
LoadFromText acQuery,"Customers and Suppliers by City" , "C:\Backups\Northwind BackupObjects\Cus
LoadFromText acQuery,"Employee Sales by Country" , "C:\Backups\Northwind BackupObjects\Employee
LoadFromText acQuery,"Invoices" , "C:\Backups\Northwind BackupObjects\Invoices.qry"
LoadFromText acQuery,"Invoices Filter" , "C:\Backups\Northwind BackupObjects\Invoices Filter.qry
LoadFromText acQuery,"Order Details Extended" , "C:\Backups\Northwind BackupObjects\Order Detail
LoadFromText acQuery,"Order Subtotals" , "C:\Backups\Northwind BackupObjects\Order Subtotals.qry
LoadFromText acQuery,"Orders Qry" , "C:\Backups\Northwind BackupObjects\Orders Qry.qry"
LoadFromText acQuery,"Product Sales for 1997" , "C:\Backups\Northwind BackupObjects\Product Sale
LoadFromText acQuery,"Products Above Average Price" , "C:\Backups\Northwind BackupObjects\Produc
LoadFromText acQuery,"Products by Category" , "C:\Backups\Northwind BackupObjects\Products by Ca
LoadFromText acQuery,"Quarterly Orders" , "C:\Backups\Northwind BackupObjects\Quarterly Orders.c
LoadFromText acQuery,"Quarterly Orders by Product" , "C:\Backups\Northwind BackupObjects\Quarter
LoadFromText acQuery,"Sales by Category" , "C:\Backups\Northwind BackupObjects\Sales by Category
LoadFromText acQuery,"Sales by Year" , "C:\Backups\Northwind BackupObjects\Sales by Year.qry"

```

Figure 5-15. The VBA recovery file that helps import all the objects into a blank database.

To load all the objects into a new database, follow these steps:

1. Open a new blank database.
2. Open the Visual Basic Editor (press ALT+F11).
3. Choose File ► Import File.
4. Find the file (its name should be Northwind_rebuildBas.txt) and click Open.
5. Find the module in the Project Explorer, which you can view by choosing View ► Project Explorer.
6. Open the Immediate window.

7. Type "call RebuildDatabase" into the Immediate window.
8. Because this database started as a blank project, you need to check your VBA project references by choosing Tools ► References. You will probably be missing references such as DAO and Microsoft Office 10.

As an alternative, you can actually use the `LoadFromText` method to load the individual VBA object recovery files into the database. To do this, open the Immediate window and type

```
LoadFromText acModule, "RebuildDatabase", _
    "c:\Backups\Northwind BackupObjects\Northwind_rebuildBas.txt"
```



CAUTION The `LoadFromText` method will copy over the existing objects without warning. If you are using this method, you probably should open a new blank database and then compare the object with your existing database before importing.

Now I will retrace my steps a little to discuss the VBA code that makes backing up and recovering objects possible.

How Exporting of Objects to Text Works

The following `onClick` procedure for the form `frmBackupAllObjects` shows you how to integrate the exporting software into your database. This procedure establishes both a folder for the backup plus a name for the VBA recovery file. It then calls the `exportObjectsToText_FX` subroutine, which you will find in the `basGR8_exportObjects` module.

```
Private Sub cmdBackupToText_Click()
' Back up all queries, forms, reports, macros, and modules to text files.

Const OBJFOLDER = "BackupObjects\"
Const REBUILD OBJ = "_rebuildBas.txt"

Dim exportAllOK As Boolean, backUpFolder As String
Dim dbNameStr As String, rebuildFile As String
```

```

backUpFolder = GetDBPath_FX(, dbNameStr)
backUpFolder = backUpFolder & dbNameStr & " " & OBJFOLDER

' Back up all objects to text.
rebuildFile = dbNameStr & REBUILDOBJ
exportAllOK = exportObjectsToText_FX(backUpFolder, rebuildFile)

If exportAllOK Then
    MsgBox "Database objects have been exported to text files in " & backUpFolder & _
        ". These files can be recovered into a blank database using VB in the file " & _
        & rebuildFile
Else
    MsgBox "Database export to " & backUpFolder & " was not successful"
End If
End Sub

```

Now we will look at the `exportObjectsToText_FX` subroutine in detail. Initially, the function creates a folder plus the instructions section of the VBA recovery file. You will be able to recognize the VBA code that creates the VBA recovery file by looking for lines that include the output channel variable `io` and the text file creation commands `Open`, `Print`, `Close`, and `FreeFile`. The first half of the subroutine follows:

```

Public Function exportObjectsToText_FX(folderPath As String, _
    rebuildFile As String) As Boolean

' Export all queries, forms, macros, and modules to text.
' Build a file to assist in recovery of the saved objects
' in a clean database.

' This function requires a reference to
' Microsoft DAO 3.6 or 3.51 Llibrary.
'Requires the modules basGR8_exportObjects and basGR8_Startup.

On Error GoTo err_exportObjectsToText

Dim dbs As DAO.Database, Cnt As DAO.Container, doc As DAO.Document
Dim mdl As Module, objName As String
Dim io As Integer, i As Integer, unloadOK As Integer
Dim FilePath As String
Dim fileType As String

If Len(Dir(folderPath, vbDirectory)) = 0 Then

```

```

unloadOK = MsgBox("All tables will be backed up to a new directory called " & _
    folderPath, vbOKCancel, "Confirm the Creation of the Backup Directory")
If unloadOK = vbOK Then
    Mkdir folderPath
Else
    GoTo Exit_exportObjectsToText
End If
End If

' The location of all the text files should be in a folder that is
' backed up and kept off-site.

io = FreeFile
Open folderPath & rebuildFile For Output As io

Print #io, "public sub RebuildDatabase"
Print #io, ""
Print #io, "' Import this into a blank database and type"
Print #io, "' RebuildDatabase into the debug window"
Print #io, ""
Print #io, _
"msgbox ""This will OVERWRITE any objects with the same name. "" & _
Print #io, _
"    ""WARNING: Press CTRL+BREAK NOW "" & _
Print #io, _
"    ""If you already have these objects in your database "" & _
Print #io, _
"    ""You will not be able to retrieve the current objects if you continue"""
Print #io, ""

```

The function must now iterate through the different collections of objects in the database by using DAO. When the loop moves to the next object, the object is saved to text and another line is written to the VBA recovery file. When it comes to exporting modules, I like to differentiate between modules and class modules by saving them to a different file type, which requires that I open the modules in design mode first. If you like, you can remove this additional code and save all class modules as .BAS files. This change will not affect the recovery process at all. Before I start this part of the subroutine, I find it useful to test whether the database has been compiled and then compile it as a final test of the quality of the database. The second half of the exportObjectsToText_FX follows.

```

If Not Application.IsCompiled Then
    ' If the application is not compiled, compile it.
    RunCommand acCmdCompileAllModules
End If

' Now test again whether the database is compiled.
' First, test whether the database is compiled.
If Application.IsCompiled Then
    Set dbs = CurrentDb()

    For i = 0 To dbs.QueryDefs.Count - 1
        objName = dbs.QueryDefs(i).Name
        FilePath = folderPath & objName & ".qry"
        If left(objName, 1) <> "~" Then
            SaveAsText acQuery, objName, FilePath
            Print #io, "LoadFromText acQuery,""" & objName & _
                """, """" & FilePath & """"
        End If
    Next i

    Print #io, ""

    Set Cnt = dbs.Containers("Forms")
    For Each doc In Cnt.Documents
        FilePath = folderPath & doc.Name & ".frm"
        SaveAsText acForm, doc.Name, FilePath
        Print #io, "LoadFromText acForm,""" & doc.Name & _
            """, """" & FilePath & """"
    Next doc

    Print #io, ""

    Set Cnt = dbs.Containers("Reports")
    For Each doc In Cnt.Documents
        FilePath = folderPath & doc.Name & ".rpt"
        SaveAsText acReport, doc.Name, FilePath
        Print #io, "LoadFromText acReport,""" & doc.Name & _
            """, """" & FilePath & """"
    Next doc

    Print #io, ""

```

```

' Scripts are actually macros.
Set Cnt = dbs.Containers("Scripts")
For Each doc In Cnt.Documents
    FilePath = folderPath & doc.Name & ".mcr"
    SaveAsText acMacro, doc.Name, folderPath & doc.Name & ".mcr"
    Print #io, "LoadFromText acMacro,""" & doc.Name & _
        """, """" & FilePath & """"
Next doc

Print #io, ""

Set Cnt = dbs.Containers("Modules")
For Each doc In Cnt.Documents
' Modules need to be opened to find if they are class or function modules.
' You can turn off open module to save all files as .BAS types.
    DoCmd.OpenModule doc.Name
    If Modules(doc.Name).Type = acClassModule Then
        fileType = ".cls"
    Else
        fileType = ".bas"
    End If
    FilePath = folderPath & doc.Name & fileType
    DoCmd.CLOSE acModule, doc.Name
    SaveAsText acModule, doc.Name, FilePath

    Print #io, "LoadFromText acModule ,"""" & doc.Name & _
        """, """" & FilePath & """"

Next doc

exportObjectsToText_FX = True
Print #io, "msgbox ""End of rebuild""""
Print #io, ""
Print #io, "end sub"

Else

MsgBox "Compile the database first to ensure the code is OK .", _
    vbInformation, "Choose Debug, Compile All Modules from the VBA window."
exportObjectsToText_FX = False
End If

```

```

Exit_exportObjectsToText:

On Error Resume Next
    Close io
    Set doc = Nothing
    Set Cnt = Nothing
    Set dbs = Nothing

Exit Function

err_exportObjectsToText:
    Select Case Err.Number
        ' Problems with unload process.

        Case Else
            MsgBox "Error No. " & Err.Number & " -> " & Err.Description
    End Select
    exportObjectsToText_FX = False

Resume Exit_exportObjectsToText

End Function

```

Now that you have seen how to import all the objects, let's have a look at how the file size of the text files will tell you the relative size of the objects in the database.

Size Does Matter: How the Text Backup Files Will Help

This section shows that unloading to text backup systems provides a tool for working out the biggest objects and tables in our databases. If a database is getting large, you will start to wonder about what is taking up all the space. Sure, you can look at the tables, find the one with the most records, and then see if you can reduce its size. But this is a hit-or-miss approach, and you'll miss many good opportunities to decrease your database size. If you think that missing these opportunities is not all that important, ponder this disaster.



User Story *One database I was called in to fix was 100MB large and was performing very poorly. Naturally, I assumed that no one had ever compacted it. Unfortunately, it was in such a mess, it wouldn't compact. After importing all the objects into another database, I managed to compact the database and save 20MB. I then hunted through the system and found a few table changes that saved another 5MB. Next, I tried for another favorite space*

saver of mine: embedded graphs that have too many rows of data stored directly in the graph object. This repair saved a couple more megabytes. Finally, I came across an innocuous small company logo in the corner of every one of the 100 reports. I took a copy of the database and removed this picture from 20 reports. Bingo—10MB saved. I then discovered that the logo was actually a large picture that had been shrunk to a small size. I asked for permission from the manager to remove the logo from the reports, and the database shrunk to only 20MB. I then split the database, and the front-end database reduced to only 4MB. If I had known the relative size of all the objects in the first place, however, this process would have been so much easier.

It's at this point where text backups are so useful because the size of the text files for both the tables and the objects provides a relative indicator of the size of the objects inside the database. If you look at the exports from the Northwind database, you will find that the Orders and OrderDetails XML table export files are the largest. The size reflects the database where these two tables clearly have the most records. Surprisingly, the Categories table is the third biggest text file. It only has five records, but each record includes a bitmap. If this table had many records, the bitmaps would really consume a lot of space unnecessarily and would probably be better stored outside the database. If you sort the programmable object exports folder by file size, the Catalog report and the Customers form are the largest at 150KB each. Though these are not large objects, both these objects have embedded pictures that may not be necessary. If you remove the picture from these objects, they shrink to 50KB when exported.

Though the Northwind object sizes may not be that exciting to you, this file size assessment technique will have a bigger impact on you when you try saving all objects from your databases.

Now we will have a look at backing up and recovering the information not covered by the table and object exports.

Backing Up Other Information

Database properties, relationship diagrams, menus, and import/export specifications can only be backed up to another Access database or documented by printing. To produce a report on the database properties and relationship diagrams as a form of backup:

1. Open a database.
2. Choose Tools ► Analyze ► Documentor.

3. Select the Current Database tab and select both Properties and Relationships. This action will produce a report that shows both database properties (see Figure 5-16) and descriptions of the relationships.

C:\Backups\Northwind.mdb

Friday, 14 March 2003

Database: C:\Backups\Northwind.mdb

Page: 1

Properties

AccessVersion:	08.50	AllowBreakIntoCode:	True
AllowBuiltInToolbars:	True	AllowFullMenus:	True
AllowShortcutMenus:	True	AllowSpecialKeys:	True
AllowToolBarChanges:	True	ANSI Query Mode:	0
Auto Compact:	0	Build:	702
CollatingOrder:	General	Four-Digit Year Formatting:	0
Log Name AutoCorrect Chan:	0	PagesFixed:	true
Perform Name AutoCorrect:	1	ProjVer:	24
QueryTimeout:	60	RecordsAffected:	0
Remove Personal Informatio:	0	RemovePersonalInformation:	0
Row Limit:	10000	Show Values in Indexed:	1
Show Values in Non-Indexed:	1	Show Values in Remote:	0
Show Values in Server:	0	Show Values in Snapshot:	1
Show Values Limit:	1000	StartupForm:	Startup
StartupShowDBWindow:	True	StartupShowStatusBar:	True
Track Name AutoCorrect Inf:	1	Transactions:	True
Updatable:	True	Use Default Connection File:	0
Use Default Page Folder:	0	Use Hijri Calendar:	0
UseAppIconForFormRpt:	False	Version:	4.0

Figure 5-16. The database documentor output shows important database properties.

An additional way to back up the table relationships is to print out the relationship model diagram. To find this option:

1. Open the back-end database where the table relationships are stored.
2. Open the relationships window.
3. Choose File ► Print Relationships.

The output from this menu command is shown in Figure 5-17. This option became available in Access 2000, but a download wizard was available from Microsoft for the same purpose in Access 97.

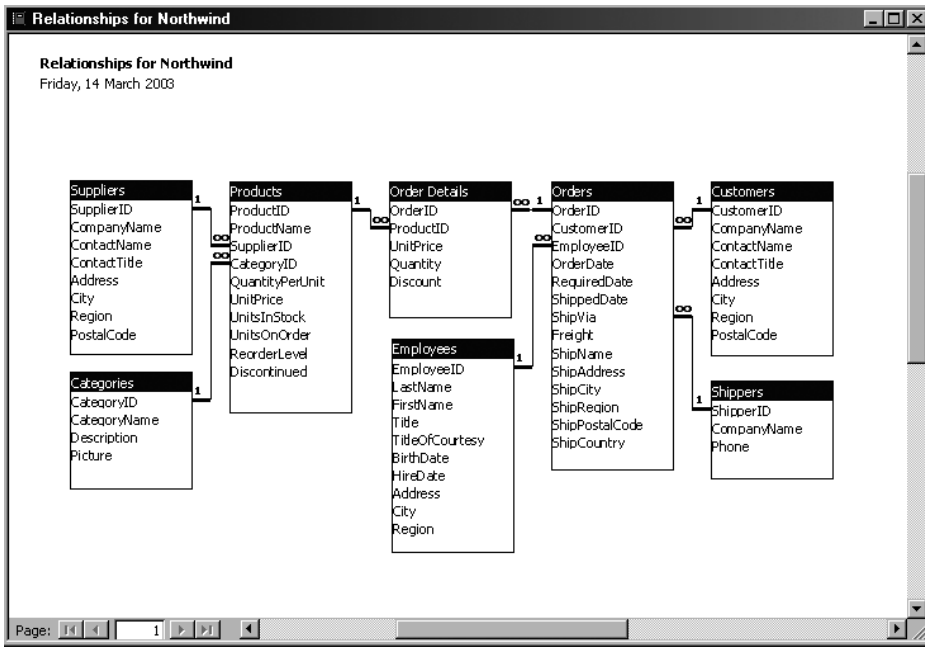


Figure 5-17. The relationship window, which you can print as part of your backup material.

Recovering Your Relationships, Menus, and Import/Export Specifications

If you import or export data to text files regularly, you can save the definition of the responses that you used for that work into an import/export specification. These specifications can take a while to re-create if lost. If you have created menus and toolbars in your application (discussed in detail in Chapter 7) or relationships between tables, then you will wonder how to back up or move that information around. The only practical way to transfer these items is to use the Import Objects wizard.

To open the Import Objects wizard, choose **File ► Get External Data ► Import**. To import the relationships, menus, and import/export specifications, click the **Options** button and then select the three check boxes in the lower left of the wizard (as shown in Figure 5-18).

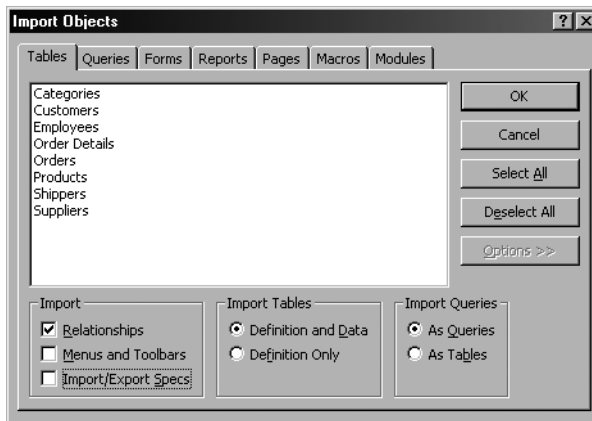


Figure 5-18. The Import Objects wizard after the Options button is clicked.

Select the Relationships check box to include the relationships defined between the tables and queries you import. It is important to note that the only relationships transferred will be those common to the tables that you are importing. Select the Menus and Toolbars check box to import all custom menus and toolbars in the database. If a custom toolbar or menu with the same name exists in the current database, that toolbar won't be imported. Select the Import/Export Specs check box to include all import and export specifications from the database you are importing.



NOTE It is quite acceptable to use the Import command from the Access main menu to import all objects into a blank database. This procedure will form quite an acceptable form of backup. Chapter 9 discusses in detail why the Import command is one of the bigger weaknesses in the Access protection mechanisms.

Backing Up Your Database by Using Access 2003

Access 2003

With the arrival of Access 2003, a new menu command will handle the naming conventions for backing up a database, and it's an option well worth considering for any DBA who is interested in a simple, well-organized backup process for their Access 2000, 2002 format database. The system works by asking you for the folder of your backup database (as shown in Figure 5-19).

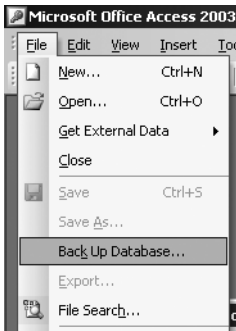


Figure 5-19. The Backup menu command.

It then compacts the current database to that folder. The file name suggested for the backup results from concatenating the name of the current database with a date string. If you back up your database more than once a day, the name of the backup database will also receive a numerical suffix to differentiate between each of the backups for the day (as shown in Figure 5-20).

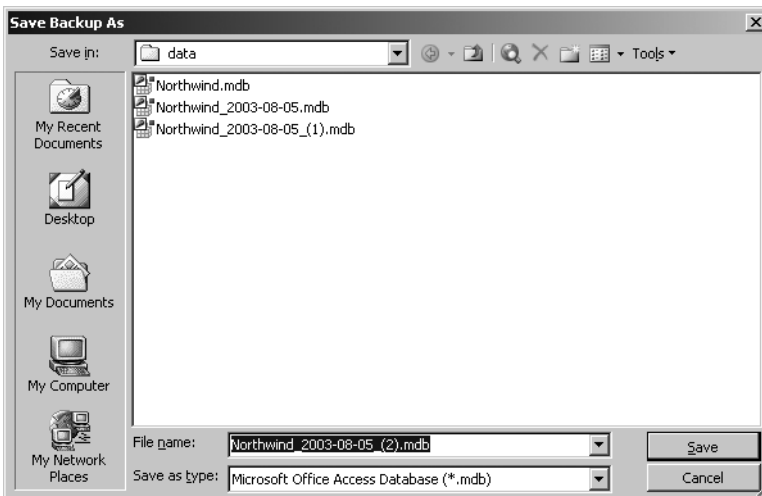


Figure 5-20. Access 2003 backup wizard in action.

Naturally, there are some complications with this process, such as:

- You need to store the backups in a safe place if you are trying to protect the information in the database.
- You will need to make sure that everyone has logged off the database before running the backup.
- You need to make sure that you back up both the front-end and back-end databases.
- You will need to clean up all the old backup databases by using Windows Explorer or some other system that lists files in the directory.

Now that we have reached the end of the descriptions of the backup techniques, I will provide you with some links so that you can continue reading on these topics.

Further Reading

As you might expect, there is never enough information when it comes to powerful products like Access. To assist you with further investigations, I have put together a Web page with links to Web sites and articles on the issues that relate to the material in this chapter. This page includes

- The download location for ADO libraries.
- A page of resources for Access corruption.
- Microsoft product support rules and regulations.
- Microsoft Knowledge Base articles on unusual file extensions.
- How to use XML exports to generate Web reports.
- Information on the `AddFromFile` method.
- How to program your own documentation of tables, indexes, and relationships.
- How to find the Access 97 Relationship Printer wizard.

You can find the further reading Web page for this chapter in the Downloads section of the Apress Web site (<http://www.apress.com>) or at the following address: <http://www.vb123.com/map/bac.htm>.

Reflecting on This Chapter

In this chapter, we have seen that there is a bit more to backing up a database than you would think. Sure, the DBA can put a new tape in the tape drive every night, and your database will be on the tape in the morning. If you ever need to recover that database, that process is probably okay. But if this is all that you do, you are missing an opportunity to introduce some additional safeguards and flexibility into your backup process.

My recommendations for you:

- Add the demonstration form (`frmAutoShutdown`) to your startup sequences in your front-end databases to help to ensure that everyone has logged off by backup time.
- If you need to back up during the day, make sure that everyone has logged off by using the `frmIsDBopenDAO` form demonstrated at the start of the chapter.
- Once a month or more, export all tables and objects to text files to create an additional safeguard.
- If you have a busy database, run the backup to XML or comma-delimited format procedures described in this chapter sometime during the day. This action will protect you from unfortunate incidents such as someone deleting a large number of records.

Finally, remember to test your recovery procedures on a regular basis.

What's Next?

In the next chapter, we will be looking at different ways that you can monitor who is in the database. Knowing who is in the database is very useful information if you have trouble with your backups or software updating because your database is open. You will also learn other useful administration techniques, such as logging when and by whom a database is opened and closed, logging when someone opens an object, and even how to keep people from logging onto the database. Also, I'll introduce an Access form that will show who is connecting to or sneaking into your databases without using the startup form or AutoExec macro.