

# Real-World SQL-DMO for SQL Server

ALLAN MITCHELL AND MARK ALLISON

**Apress™**

Real-World SQL-DMO for SQL Server

Copyright © 2003 by Allan Mitchell and Mark Allison

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): 1-59059-040-6

Printed and bound in the United States of America 12345678910

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: Ron Talmage

Editorial Directors: Dan Appleman, Gary Cornell, Jason Gilmore, Simon Hayes, Karen Watterson, John Zukowski

Managing Editor: Grace Wong

Project Manager and Development Editor: Tracy Brown Collins

Copy Editor: Nicole LeClerc

Compositor: Susan Glinert

Illustrator: Cara Brunk, Blue Mud Productions

Cover Designer: Kurt Krames

Indexer: Valerie Perry

Production Manager: Kari Brooks

Manufacturing Manager: Tom Debolski

Marketing Manager: Stephanie Rodriguez

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](mailto:orders@springer-ny.com), or visit <http://www.springer-ny.com>. Outside the United States, fax +49 6221 345229, email [orders@springer.de](mailto:orders@springer.de), or visit <http://www.springer.de>.

For information on translations, please contact Apress directly at 2560 9th Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax: 510-549-5939, email [info@apress.com](mailto: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 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.

# Backup and Restore: A DBA's Bread and Butter

ONE OF A DATABASE ADMINISTRATOR'S (DBA's) primary jobs is backing up a database so that should the need arise it can be restored. This chapter deals with how to do that using SQL-DMO by building a sample application.

In our opinion, one of the most important tasks we perform as DBAs is creating a backup plan for our servers and databases. The type of plan we implement depends on the type of environment we're looking at.

When we arrive at a new client's site, the first thing we like to do is find out what we back up, what we don't back up, and why and where. Even if we never use the backups, it's always good to have them, just in case. We've lost count of the number of times we've been asked by developers to restore a database because they forgot to limit a DELETE statement by using a WHERE clause, and instead of deleting 11 rows they ended up removing 11,000.

DBAs aren't exempt from these sorts of issues, either. We recently had a junior administrator ask us if we could restore a database because he managed to drop half the tables in the database before realizing his mistake. Natural disasters are another reason to keep backups. We recently had to restore a few database servers because when we arrived at the client's site, we found the server room doing an impression of Jacques Cousteau—not a pretty sight, we assure you.

In this chapter we'll show you some of the various backup methods available and how you can implement them using SQL-DMO. We provide quite a bit of explanation at first, as we think it's important that you understand a little about how backups work and how to use them. It's also important to look into any options set on the database that may influence your backup strategy. We then move on to cover restoring your backup. We find this an often-overlooked point—after all, what's the point in creating backups if none of them is any good?

To illustrate the concepts we describe, we're going to take you through a couple of our applications. They're very simple, but they provide practical examples of what we're writing about. The first application will do the backups and the second application will do the restore. Finally, you'll look at doing some log shipping. *Log shipping* is a way of maintaining a copy of your database(s) on another server. Having this kind of fallback can reduce the time it takes you to recover from a disaster.



---

**NOTE** If you're using SQL Server 2000 Enterprise Edition, you can configure log shipping in your Database Maintenance Plan Wizard. If you don't use this version of SQL Server, there is a log shipping tool in the SQL Server Resource Kit in ToolsAndSamples\SimpleLogShipper.

---

We begin with a discussion of the available backup methods, and then we discuss

- Applying sample backup methods
- Choosing your backup method
- Using backup devices
- Performing the backup
- Using the 15-minute database checker application

## Types of Backup Available

There are four types of backups available to you: full, differential, file group, and transaction log. We describe each backup type in more detail in the following sections.

### *Full Backup*

In a *full backup*, SQL Server does exactly as the backup's name suggests: It backs up the entire database. Every single database page is taken to the media of your choice, be it file or tape. Users can still be logged in at the time of the backup, but they can't be performing any of the following:

- DBCC SHRINKDATABASE
- DBCC CHECKALLOC
- SELECT INTO
- BCP

If users are active in the database during the backup, it makes sense that there will be transactions in the database that the backup has missed. It is for this reason that at the end of the backup, SQL Server backs up the transaction log to capture these new transactions.

## Differential Backup

When a full backup occurs, SQL Server marks all the data pages as having been backed up. When you use your database again, and maybe insert some new data or update a row in a table, SQL Server marks that extent as changed. *Differential backups* hunt through your database and only backup those extents (8 pages of data) that have been tagged as modified.

Differential backups, however, won't reset the marker to indicate that it has been backed up. A differential backup is cumulative, meaning that it will include all database changes made since the last full backup. If you perform a differential backup today and one tomorrow, then the backup tomorrow will contain all of the changes in today's backup as well. This method is faster than the others because it only backs up new data in the database.




---

**NOTE** Your mileage may vary with this method, especially in SQL Server 7.0. The number of changes isn't necessarily indicative of the time that a differential takes to happen. Backup file size is also something that varies greatly. We've spoken to people who have a database of 10GB with 32GB differential backup files.

---

## File Group Backup

This is possibly the most difficult type of backup from a management perspective. The way a file group backup works is this: In a database you can have many file groups, which can contain many physical files. You can place objects in your database into certain file groups. You choose this method if, for example, you have a table that has 40 million rows and it is constantly being queried. To get the most from the table, you decide to place it in a file group on the fastest hard disk drive you have or a fast redundant array of independent disks (RAID) 10 array. When this table is queried, you can get to your data more quickly. Another use is when you have a very large database (VLDB) and the amount of time it takes to back up the whole thing is greater than the window of opportunity you have. If you split up

the database into different file groups, you can back them up one at a time, and when they're all backed up, you have a full database backup equivalent.

## *Transaction Log Backup*

The transaction log maintains a history of the activity in your database. When you insert a row into a table in your database, you aren't actually writing it to disk but to the log. At specified periods, called *checkpoints*, SQL Server will write out your INSERT statement to the disk. As we hope you can see, the log is a very important file. A *transaction log backup* will back up all transactions that have happened in your database since the last full, differential, or transaction log backup. The way in which the log behaves is dependent on some of the options set in your database, which we explain later in the section "Considerations for Choosing Your Backup Method."

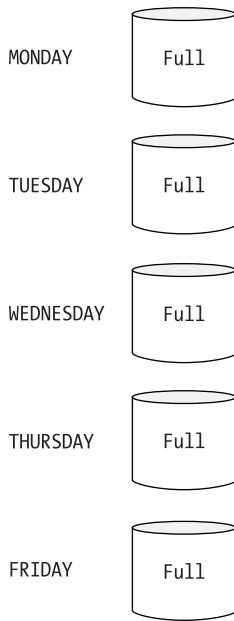
## Sample Backup Scenarios and Methods

As we mentioned earlier in the chapter, how you back up your databases is dependent on many factors. In a test environment, you may only need to do a full backup once a week and no more. In a production environment, however, you'll most probably need to be able to recover your database to the time as near as possible to a disaster. You can use a combination of the types of backups we described in the previous section to achieve your aims. In this section, we describe two methods of backing up your database.

### *Method 1*

This first method is the simplest backup strategy. Every day at a predetermined hour you do a full backup of your database. You don't do anything else to it at all. This unfortunately gives you 24 hours of exposure (*exposure* being the amount of data you could lose). If you lost the server or database in the middle of the day, then you could only recover to the backup taken the night before. (Yes, if you got to the log file you would be able to recover more, but we're presuming here that you only have your backups to go on.)

This type of plan may be suitable in a test environment where being up-to-date is not essential, or where you can recover any lost data through scripts. In order to recover this database, you would just need to restore it from the previous night's backup and carry on. This method is very quick to set up, but it's only suitable in a few cases. Figure 2-1 shows a possible backup scenario using only full backups.



*Figure 2-1. Scenario using only full backups*

## Method 2

The second method is a more detailed plan that should enable you to recover the database to the second of a disaster. For example, say on a Sunday night you do a full backup of your database. Then Monday through Friday you do hourly transaction log backups during business hours. Each weekday night you do differential backups of the database as well. Then you have a database failure on Thursday at 3:30 P.M. and you manage to back up the current transaction log of the database. To restore this database to its previous state, you would need to do the following:

1. Restore the full backup from Sunday.
2. Restore the differential backup from Wednesday night.
3. Restore the transaction log backups that occurred between start of business on Thursday and 3:00 P.M.
4. Restore the final transaction log backup that you managed to rescue from the database.

As you can see, this method is certainly involved, but it does allow you to recover more data than the first method. In fact, if you're able to get to the current transaction log, then you may be able to recover the database without losing any data at all. Figure 2-2 shows a possible backup scenario using a combination of full, differential, and transaction log backups.

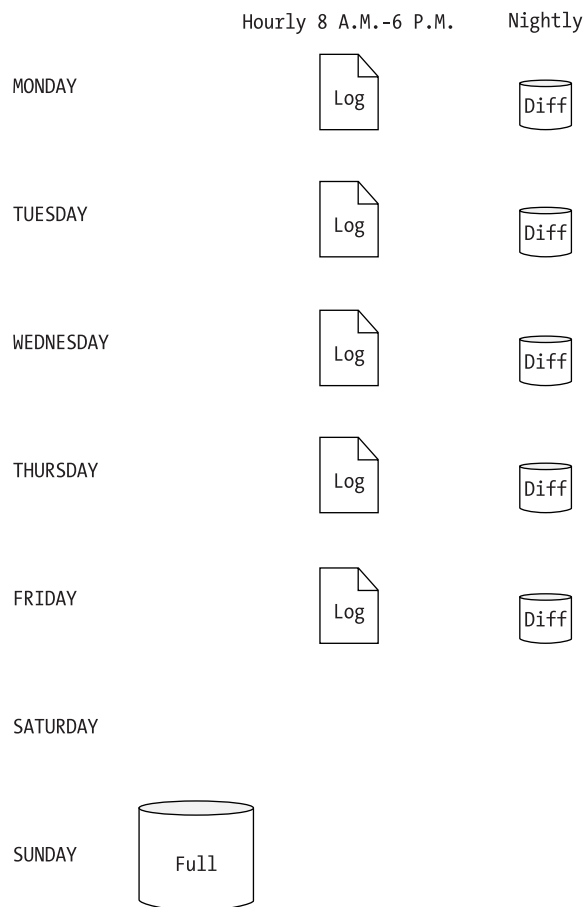


Figure 2-2. Scenario using full, differential, and transaction log backups

## Considerations for Choosing Your Backup Method

Remember earlier when we said there were certain database settings that would affect your ability to do some types of backup? Well, here they are. In the sections that follow, the database settings are broken down into the different options for both SQL Server 7.0 and SQL Server 2000.





---

**NOTE** We explain how to set these options using T-SQL, SQL-DMO, and Enterprise Manager in Chapter 1.

---

## SQL Server 7.0

In SQL Server 7.0 you have to do a bit more work than you do in SQL Server 2000, as version 2000 sets the version 7.0 options under the covers when you choose your recovery model.

### *Truncate Log on Checkpoint*

This option is very easy to remember. It wrecks your chances of doing transaction log backups. This means that the only backups you can perform with this option set are full and differential. This might not be a problem for you, but we only recommend using this option in development and test environments.

The way the option works is this: When SQL Server has written out your data to disk from the log at preset intervals, it will come along and clean up after itself. It doesn't back up the log—it just gets rid of any transactions that have been marked as completed. If you try to back up the log when this option is set, you'll get an error. It effectively creates holes in the life of a database. You have nothing to put back into your database because SQL Server has removed the entries from the log.



---

**NOTE** Lots of people ask if they can just get rid of the transaction log because they don't need it. The answer to this is no. SQL Server needs the transaction log to ensure database consistency. If you get rid of the transaction log and your database has the option Truncate Log on Checkpoint set, then you can't do file group backups, because these backups rely on the transaction log to make them consistent.

---

### *Select Into/Bulk Copy*

The Select Into/Bulk Copy option enables you to perform actions such as SELECT INTO, BCP, or WRITETEXT in your database. These types of actions are often referred to as *nonlogged*, but this is actually a bit of a misnomer as they're logged

but only minimally. Once you perform one of these actions, you invalidate point-in-time recovery of your database. *Point-in-time recovery* is where you can stop a restore of a database at a given time, which is preferably just before the action occurred that caused you to do the restore.

An example of this would be when you have a table with records in it that you want to replicate, perhaps just in case the following operation goes wrong. You could CREATE the table and then do an INSERT INTO to get the records in, but instead you decide to use SELECT INTO, as that will create the table for you as well. SQL Server will then create your new table and pump the records in. It will log the fact that it allocated extents to the table, but it won't log every insertion. This is where gaps in recovery may appear.



---

**CAUTION** Using SELECT INTO on a large table isn't a good idea. It will lock up tempdb for the duration of the statement; therefore, it has the potential to stop others from working. If you can create the table first and then do an INSERT statement, you'll stop this from happening.

---

## SQL Server 2000

SQL Server 2000 does things a little differently from SQL Server 7.0. You can still use the options described previously, but they're provided for backward compatibility and should only be used as such. In version 2000, you have recovery options, of which there are three.

### *The Simple Recovery Option*

You would use this option when you have no use for the transaction log whatsoever and you want to make sure that it doesn't get too large. As with the Truncate Log on Checkpoint option in SQL Server 7.0, the Simple Recovery option allows you to only use full and differential backups. If your database is set to Simple, then you can't do file group backups, as these backups rely on the transaction log to make them consistent.

### *The Bulk-Logged Recovery Option*

Here is where the differences between the SQL Server 7.0 and SQL Server 2000 options really start to become evident. With this model you can perform your minimally logged operation as you can with the Select Into/Bulk Copy option of SQL Server 7.0. The transaction log will log the fact that a nonlogged operation has occurred, and pages within the database will be marked as having been subjected to minimally logged operations. When you come to back up your database, the log file itself will still be small, but in addition to backing itself up it will also back up those extents that have been modified by the minimally logged operation.

### *The Full Recovery Option*

The Full Recovery option allows you to have the least data loss of the three options. If you've been making transaction log backups, then you have the ability to restore to a point in time. Although this may sound like the best option, there's a penalty to pay. Because everything but the kitchen sink is logged, this option can result in the size of the transaction log being huge. This means, of course, that it will take a longer time to back up as well. You can still perform your minimally logged operations when the database is using the Full Recovery mode.

## Using Backup Devices

There are two ways of telling SQL Server where to send your backups. The first is by typing in the file path to the location where you want SQL Server to send your backups every time you issue the BACKUP DATABASE command. The second is by creating a backup device that points to the location where you want SQL Server to send your backups and then using the name of the backup device each time you do a backup. The only difference between the two methods that you'll probably notice is that the second method involves less typing.

Another neat thing about using a backup device is that in Enterprise Manager you can easily see, using the GUI, the contents of the backup device. We'll show you shortly how you can create your own view of the contents of your backup device, and we'll also show you the code that Enterprise Manager is issuing under the covers. Let's see, then, how to add a backup device to your server using three methods: T-SQL, SQL-DMO, and Enterprise Manager.

*T-SQL*

```

sp_addumpdevice [ @devtype = ] 'device_type' ,
    [ @logicalname = ] 'logical_name' ,
    [ @physicalname = ] 'physical_name'
    [ , { [ @cntrltype = ] controller_type
        | [ @devstatus = ] 'device_status'
    }
    ]

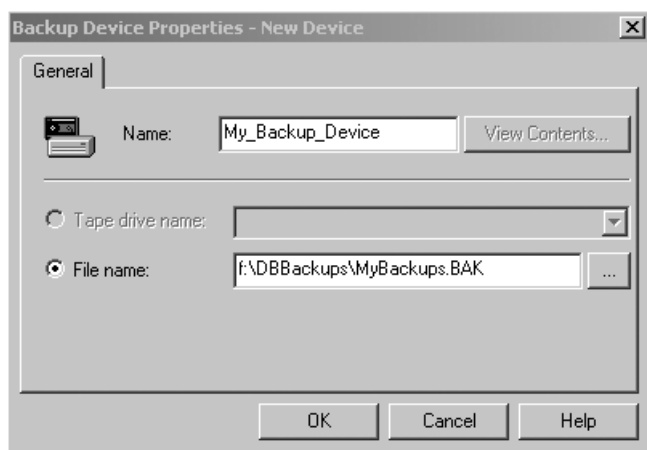
```

Example

```
EXEC sp_addumpdevice 'disk','My_Backup_Device','f:\DBBackups\MyBackups.bak'
```

*SQL-DMO*

In Enterprise Manager, you'll find the option to add a device under the Management folder. Then click the Backup icon. Figure 2-3 shows the prompt in SQL Server when you create a new backup device.



*Figure 2-3. Adding a backup device in Enterprise Manager*

The following code creates a SQLServer object and a BackupDevice object:

```
Private Sub cmdCreate_Click()
```

```
Dim oServer As SQLDMO.SQLServer
Dim oDevice As SQLDMO.BackupDevice
```

```
Set oServer = New SQLDMO.SQLServer
```

```
Connect to the SQL Server
oServer.LoginSecure = True
oServer.Connect "AM2"
```

```
Set oDevice = New SQLDMO.BackupDevice
```

The following code sets the properties of the backup device. First, you set the Name property of the device:

```
oDevice.Name = "My_Backup_Device"
```

You want to create a device that points to location on the physical hard disk, so you specify the options here to do that:

```
oDevice.Type = SQLDMODevice_DiskDump
oDevice.PhysicalLocation = "f:\DBBackups\MyBackups.bak"
```

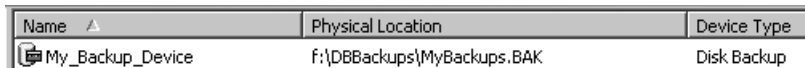
Once you've set all the properties of the device, the last thing you need to do is add the Device object to the Devices collection of the server:

```
oServer.BackupDevices.Add oDevice
```

```
oServer.DisConnect
Set oServer = Nothing
```

```
End Sub
```

The result of these three methods is shown in Figure 2-4.



Name ▲	Physical Location	Device Type
My_Backup_Device	f:\DBBackups\MyBackups.BAK	Disk Backup

*Figure 2-4. Creation of the device is confirmed.*




---

**TIP** SQL Server Books Online provides a list of the other options for the device type and controller type.

---

## Performing the Backup

Now you come to the meat of the chapter, where you'll actually perform your backup. In this section, we show you how to do a full, a differential, and a transaction log backup. We also show you some of the options available to you when you do a backup. The following code shows how to do the backup for the database. The two code listings that follow are taken directly from SQL Server Books Online.

### T-SQL

```
BACKUP DATABASE
{ database_name | @database_name_var }
TO < backup_device > [ ,...n ]
[ WITH
    [ BLOCKSIZE = { blocksize | @blocksize_variable } ]
    [ [ , ] DESCRIPTION = { 'text' | @text_variable } ]
    [ [ , ] DIFFERENTIAL ]
    [ [ , ] EXPIREDATE = { date | @date_var }
      | RETAINDAYS = { days | @days_var } ]
    [ [ , ] PASSWORD =
      { password | @password_variable } ]
    [ [ , ] FORMAT | NOFORMAT ]
    [ [ , ] { INIT | NOINIT } ]
    [ [ , ] MEDIADESCRIPTION = { 'text' | @text_variable } ]
    [ [ , ] MEDIANAME =
      { media_name | @media_name_variable } ]
    [ [ , ] MEDIAPASSWORD =
      { mediapassword | @mediapassword_variable } ]
    [ [ , ] NAME =
      { backup_set_name | @backup_set_name_var } ]
    [ [ , ] { NOSKIP | SKIP } ]
    [ [ , ] { NOREWIND | REWIND } ]
    [ [ , ] { NOUNLOAD | UNLOAD } ]
    [ [ , ] RESTART ]
    [ [ , ] STATS [ = percentage ] ]
]
```

Here's the code for backing up the log:

```
BACKUP LOG { database_name | @database_name_var }
{
    TO < backup_device > [ ,...n ]
    [ WITH
        [ BLOCKSIZE =
            { blocksize | @blocksize_variable } ]
        [ [ , ] DESCRIPTION = { 'text' | @text_variable } ]
        [ [ , ] EXPIREDATE = { date | @date_var }
            | RETAIN_DAYS = { days | @days_var } ]
        [ [ , ] PASSWORD =
            { password | @password_variable } ]
        [ [ , ] FORMAT | NOFORMAT ]
        [ [ , ] { INIT | NOINIT } ]
        [ [ , ] MEDIADESCRIPTION =
            { 'text' | @text_variable } ]
        [ [ , ] MEDIANAME =
            { media_name | @media_name_variable } ]
        [ [ , ] MEDIAPASSWORD =
            { mediapassword | @mediapassword_variable } ]
        [ [ , ] NAME =
            { backup_set_name | @backup_set_name_var } ]
        [ [ , ] NO_TRUNCATE ]
        [ [ , ] { NORECOVERY | STANDBY = undo_file_name } ]
        [ [ , ] { NOREWIND | REWIND } ]
        [ [ , ] { NOSKIP | SKIP } ]
        [ [ , ] { NOUNLOAD | UNLOAD } ]
        [ [ , ] RESTART ]
        [ [ , ] STATS [ = percentage ] ]
    ]
}
```

As you can see, you have a lot of options available to you when you back up a database or log, and you may never use most of them. In our examples, we're concerned with the ones you'll use most, but for completeness, we think it's worthwhile to provide a short description of each option:

- **DATABASE\_NAME:** This is simply the name of the database you want to back up.
- **BACKUP\_DEVICE:** This is where you want to put the backup. It can be either a physical path or one of the backup devices you created earlier.

- *BLOCKSIZE*: This is the physical size of blocks in bytes. According to SQL Server Books Online, this isn't necessary, as SQL Server will choose the most appropriate size for you.
- *DESCRIPTION*: This is a label for your backup. You may want to include a short description of why you made the backup.
- *DIFFERENTIAL*: This option indicates if this backup is differential.
- *EXPIREDATE*: This option gives a date for when the backup can be overwritten.
- *RETAIN\_DAYS*: This option is like *EXPIREDATE*, except it indicates how many days the backup is kept before you're able to overwrite it.
- *PASSWORD*: You can password-protect your backup, and you must supply the password in order to restore from it.
- *FORMAT/NO FORMAT*: This option indicates whether or not the media header should be written on all backup devices.
- *INIT/NOINIT*: This option indicates whether or not to clear down the contents of your backup device.
- *MEDIADESCRIPTION*: This option gives you a chance to describe the media as something useful.
- *MEDIANAME*: This option gives you a chance to give the media a name.
- *MEDIAPASSWORD*: If you specify a password for the media, then before you're able to create a backup on it, you must supply that password.
- *NAME*: This is the name of the backup set.
- *NOSKIP/SKIP*: If you've set your backups to have expiration dates, then using *NOSKIP* will force SQL Server to check that before trying to overwrite it. The *SKIP* option will bypass checking and overwrite as necessary.
- *NOREWIND/REWIND*: This option indicates to SQL Server whether or not to rewind and release the tape.



- *NOUNLOAD/UNLOAD*: This option indicates to SQL Server whether or not to rewind and unload the tape. SQL Server Books Online says this is only for tape devices, although later you'll see Enterprise Manager using it against disk devices once you get some backups onto your device.
- *RESTART*: This option does exactly what its name indicates: It restarts an interrupted backup at the point of interruption.
- *STATS = [percentage]*: This option indicates to SQL Server to let you know when each percentage of the backup has completed. If you don't specify this option, the default is 10 percent. You only use this on large backups because it means you have something to look at as the backup is happening.

For the following examples you'll back up the same database to the same backup device that you created earlier, but you'll use different options in each so you can see the results of the options.

## *Backing Up MyDMODatabase*

Here you'll back up the database MyDMODatabase and label the backup as having been done using T-SQL. The backup here is a full backup.

### *T-SQL*

```
BACKUP DATABASE MyDMODatabase
TO My_Backup_Device
WITH
    NAME = 'Done using T-SQL'
```

After you do the backup, check its integrity with the following code:

```
RESTORE VERIFYONLY FROM My_Backup_Device
```

If you had more than one backup on the same device, you would need to specify a file number, as in the following code:

```
RESTORE VERIFYONLY FROM My_Backup_Device WITH FILE = 1
```

In Enterprise Manager, there are a number of ways to get your database backed up. We've chosen to show three here for brevity:

- Right-click the database, and then choose All Tasks and Backup Database.
- Choose the Tools menu and then select Backup Database.
- Select the wand from the toolbar, choose Maintenance, and then select Backup Wizard.

Which method you choose to use is immaterial—you end up looking at the same screen eventually—but we rarely use wizards. Figure 2-5 shows the screen that greets you if you choose the Tools menu and then select Backup Database. As you can see from the options selected in the figure, we're doing a differential backup of the MyDMODatabase. We're calling it "Done Using EM" and appending it to the backup device we created earlier.

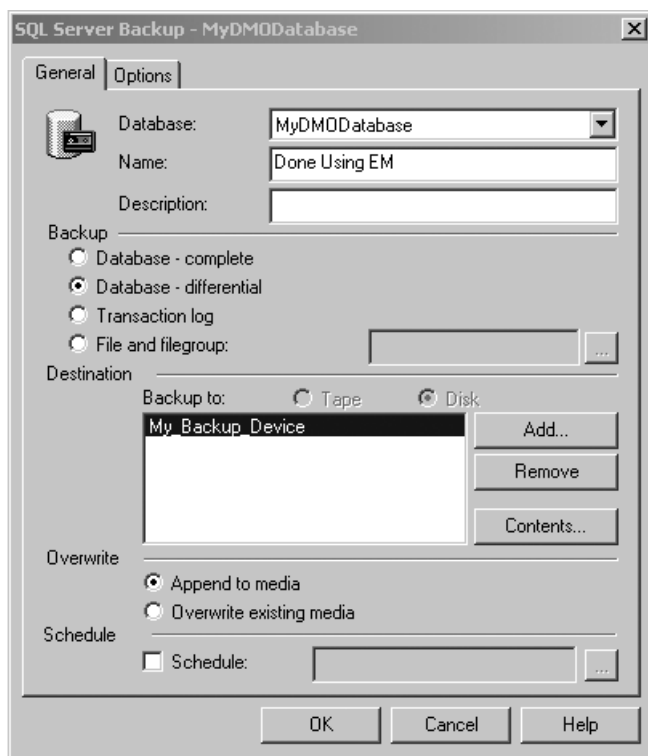


Figure 2-5. Enterprise Manager backup database

## Backing Up MyDMODatabase Using the Application

You'll be backing up MyDMODatabase to your backup device using a differential backup and setting the time that SQL Server keeps this backup without allowing it to be overwritten for 2 days. We've provided a GUI along with it to show the ease with which you can do it (see Figure 2-6).

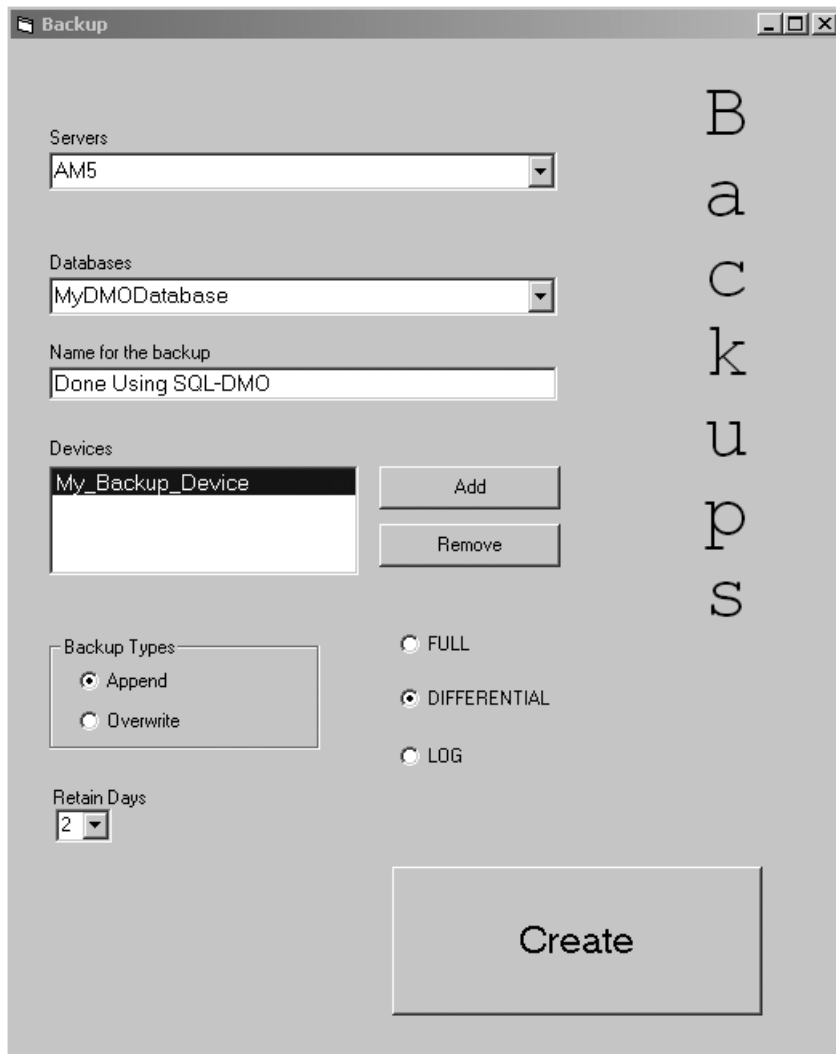


Figure 2-6. The backup application

This simple application takes very little time to create. It does more or less exactly what the GUI in Enterprise Manager does. Now we'll go through the mechanics of what's happening behind the scenes and show you a few things about what Microsoft is doing in Enterprise Manager under the covers.

```
Public oServer As SQLDMO.SQLServer
Public oDevice As SQLDMO.BackupDevice
Public oDatabase As SQLDMO.Database
Public oBackup As SQLDMO.Backup
Public oRegisteredServer As SQLDMO.RegisteredServer
Public oApp As SQLDMO.Application
Public oRestore as SQLDMO.Restore
Public oGroup as SQLDMO.ServerGroup
```

The following procedure will populate the Servers combo box with a list of registered servers:

```
Private Sub ShowServers()

Set oApp = New sqldmo.Application
Set oRegisteredServer = New sqldmo.RegisteredServer

cboServers.Clear
```

### *Grouping Your Servers*

You may group your servers in Enterprise Manager to define their roles in your business or indicate where they're located in the country. First, you need to loop through the ServerGroups collection, looking for the names of your server groups:

```
For Each oGroup In oApp.ServerGroups
```

Groups contain your registered servers, so you need to loop through your groups looking for any to add. To do this, you'll need to loop through the RegisteredServers collection.

```
For Each oRegisteredServer In oApp.ServerGroups(oGroup.Name).RegisteredServers
    cboServers.AddItem oRegisteredServer.Name
Next oRegisteredServer

Next oGroup
```

```
Set oGroup = Nothing
Set oRegisteredServer = Nothing
Set oApp = Nothing
```

```
End Sub
```

```
Private Sub ShowDatabases(servername As String)
```




---

**NOTE** There's another way to look for all your SQL Servers: You can use the ListAvailableSQLServers method of the Application object. The reason we don't use this method here is because although it will find your SQL Servers, it will only do so if you're on the same network segment as the SQL Servers. This is because the ListAvailableSQLServers method uses a broadcast to locate the SQL Servers and this broadcast won't travel through routers. We've met some people on newsgroups that have also had this problem.

---

### *Outputting Database Names to the Combo Box*

When you select the server that contains the database you want to back up, you need to populate the Databases combo box with a list of available databases on that server. Here you just loop through the Databases collection of the connected SQLServer object and output the database names to the combo box.

```
Set oServer = New SQLDMO.SQLServer

oServer.LoginSecure = True
oServer.Connect servername

cboDatabases.Clear

For Each oDatabase In oServer.Databases
    cboDatabases.AddItem oDatabase.Name
Next oDatabase
```

```
oServer.DisConnect  
Set oServer = Nothing  
Set oDatabase = Nothing
```

```
End Sub
```

```
Private Sub ShowBackupDevices(servername As String)
```

For this procedure, you want to list the available predefined backup devices on your server. All you do is loop through the BackupDevices collection of the server.

```
Set oServer = New SQLDMO.SQLServer  
  
oServer.LoginSecure = True  
oServer.Connect servername  
  
lstdevices.Clear  
  
For Each oDevice In oServer.BackupDevices  
    lstdevices.AddItem oDevice.Name  
Next oDevice  
  
oServer.DisConnect  
Set oServer = Nothing  
Set oDevice = Nothing  
  
End Sub
```

### *Prompting the Actual Backup*

The following procedure will actually do the backup and will be called by the Create button on the form. The procedure takes a number of parameters. The first two, servername and DatabaseName, are obvious. The third, Location, is where you want to send the backup. The fourth, DeviceYN, indicates whether the location you've chosen is a predefined backup device or one that you've added because maybe this is an ad hoc backup. The fifth parameter, BackupType, will tell you whether you're doing a full, differential, or transaction log backup. Next, RetainDays, is how long you want to keep the backup before you're able to overwrite it. Then you tell the backup if you should initialize the device first with InitFirst. Finally, BackupName is the name of the backup.

```
Private Sub BackupTheDatabase(servername As String,
    DatabaseName As String, Location As String,
    DeviceYN As Integer, BackupType
    As SQLDMO_BACKUP_TYPE, RetainDays As Integer,
    InitFirst As Boolean, BackupName As String)
```

```
On Error GoTo Err_handler
```

```
Set oServer = New SQLDMO.SQLServer
```

During testing we came across an interesting anomaly. If you try to back up a database in Enterprise Manager and you don't supply a backup name, SQL Server will tell you that you need one, as shown in Figure 2-7.



Figure 2-7. Message box requesting a backup name




---

**NOTE** The error also happens if you try to back up the database using SQL-DMO using your application. This seems fair, as all Enterprise Manager does is issue SQL-DMO anyway. The problem is when you try to back up the database using T-SQL. You don't need to supply a name, and SQL Server has no problem with that. If you read the header from your backup, you'll see that the name of the backup set is "NULL," which indicates that no value is passed.

---

## Initializing the Backup

You initialize your backup and restore objects with the following code:

```
Set oBackup = New SQLDMO.Backup
Set oRestore = New SQLDMO.Restore
```

Then you log on to the server on which you want to do a backup:

```
oServer.LoginSecure = True  
oServer.Connect servername
```

You now pass your values to the backup object. What type of backup is it?

```
oBackup.Action = BackupType
```

What's the name of the backup?

```
oBackup.BackupSetName = BackupName
```

Which database are you backing up?

```
oBackup.Database = DatabaseName
```

How long are you keeping the backup?

```
oBackup.RetainDays = RetainDays
```

Are you initializing the media first?

```
oBackup.Initialize = InitFirst
```

Here's where you indicate whether or not the location you're sending the backup to is a predefined device or a location you added. Note the different properties of the Backup object you use. You also set the Restore object's place to look for files.

```
If DeviceYN = 1 Then  
    oBackup.Devices = Location  
    oRestore.Devices = Location  
Else  
    oBackup.Files = Location  
    oRestore.Files = Location  
End If
```

Here's the actual backup:

```
oBackup.SQLBackup oServer
```

This is the check of the backup to ensure it's structurally sound. After all, what's the point in a backup that's rubbish?



```
oRestore.SQLVerify oServer
```

```
MsgBox "The database Backup Succeeded", vbOKOnly, "Backup Completed"
```

```
oServer.DisConnect
```

```
Set oServer = Nothing
```

```
Exit Sub
```

```
Err_handler:
```

The following error number indicates a corrupted backup:

```
If Err.Number = -2147218262 Then
```

```
MsgBox "The backup you have just " & _  
"performed would appear to be " & _  
corrupted", vbCritical, "Backup problem"
```

```
Else
```

```
MsgBox "The database Backup " & _  
Failed" & vbCrLf & Err.Description, vbOKOnly, _  
"Backup Completed"
```

```
End If
```

```
oServer.DisConnect
```

```
Set oServer = Nothing
```

```
Exit Sub
```

```
End Sub
```

```
Private Sub cboServers_Click()
```

```
ShowDatabases cboServers.Text
```

```
ShowBackupDevices cboServers.Text
```

```
End Sub
```

Here's where you add new locations for your backups:

```
Private Sub cmdAdd_Click()
```

```
Dim strNewLocation As String
```

```
strNewLocation = InputBox("Enter A New Location", "New Backup Location")
```

```
If strNewLocation <> "" Then  
    lstdevices.AddItem strNewLocation  
End If
```

```
End Sub
```

```
Private Sub cmdCreate_Click()
```

```
Dim servername As String  
Dim BackupType As SQLDMO_BACKUP_TYPE  
Dim Init As Boolean  
Dim DeviceCounter As Integer  
Dim BackupName As String
```

```
Init = False
```

You need to set the backup type that you're going to be using:

```
If optFull.Value = True Then  
    BackupType = SQLDMOBackup_Database  
ElseIf OptDiff.Value = True Then  
    BackupType = SQLDMOBackup_Differential  
ElseIf OptLog.Value = True Then  
    BackupType = SQLDMOBackup_Log  
End If
```



---

**NOTE** As with Enterprise Manager, when you back up the log you get no options for it. The default behavior of SQL Server when backing up the log is to back it up and remove completed transactions from it.

---

Here you indicate if you want to initialize the device:

```
If OptOverwrite.Value = True Then  
    Init = True  
End If
```

If you've chosen a server, a database, and somewhere to send it to, you can proceed.

```
If (cboServers.Text <> "" Or cboDatabases.Text <> ""
Or lstdevices.ListIndex <> -1) Then
```

```
    Set oServer = New SQLDMO.SQLServer
    Set oBackup = New SQLDMO.Backup
```

Finally, log on to the server:

```
oServer.LoginSecure = True
oServer.Connect cboServers.Text
```

### *Locating the Backup to Restore*

Here's where you find out if the location for the backup is in the BackupDevices collection. You do this by looking for a match between the names in the list box on the form and the names of your BackupDevices. If you get a hit, you set DeviceCounter = 1, which indicates "Yes."

```
For Each oDevice In oServer.BackupDevices
    If oDevice.Name = lstdevices.Text Then
        DeviceCounter = 1
    End If
Next oDevice
```

Because you need to supply the backup with a name, and you may have forgotten to put anything in the Name box, you supply a default in the form of *<database name>\_yyyymmddhhmmss*.

```
If txtName = "" Then
    BackupName = _
cboDatabases.Text & "_" & Format(Now(), _
"yyyymmdd") & Format(Now(), "hhmmss")
Else
    BackupName = txtName.Text
End If
```

Here you call the backup procedure with the relevant parameters:

```
BackupTheDatabase cboServers.Text, cboDatabases.Text,
lstdevices.Text, DeviceCounter, BackupType, cboRetain.Text,
Init, BackupName

End If

End Sub
```

Remove devices from the Devices list box (don't remove them from the server, though):

```
Private Sub cmdremove_Click()
If lstdevices.ListIndex <> -1 Then
    lstdevices.RemoveItem lstdevices.ListIndex
End If
End Sub

Private Sub Form_Load()
cboRetain.ListIndex = 0
ShowServers
optAppend.Value = True
optFull.Value = True
End Sub
```




---

**NOTE** When you open up Enterprise Manager and select your database, SQL Server remembers where you last sent it to for backing up. We've chosen not to do this here, but Enterprise Manager uses the following code to do it:

```
use msdb
select
distinct f.device_type, f.physical_device_name, f.logical_device_name,
    b.database_name
from
backupmediafamily f, backupset b where
    b.database_name = N'MyDMODatabase'
and
b.backup_finish_date = (select MAX(backup_finish_date)
    from backupmediafamily
INNER JOIN backupset ON
backupmediafamily.media_set_id=backupset.media_set_id
```

```

where backupset.database_name = N'MyDMODatabase'
and (backupmediafamily.device_type=2 or
backupmediafamily.device_type=102))
and b.media_set_id = f.media_set_id

```

---

## Backing Up the Log

Periodically, you may need to back up the log of your database. Using Enterprise Manager and your SQL-DMO application, you can find everything in the same place and you just specify a log backup. In T-SQL, however, you need to use slightly different syntax. Most of the options are the same, but a few need explanation. Here's the code, which is taken from SQL Server Books Online.

### T-SQL

```

BACKUP LOG { database_name | @database_name_var }
{
    TO < backup_device > [ ,...n ]
    [ WITH
        [ BLOCKSIZE =
        { blocksize | @blocksize_variable } ]
        [ [ , ] DESCRIPTION =
        { 'text' | @text_variable } ]
        [ [ , ] EXPIREDATE =
        { date | @date_var }
        | RETAINDAYS = { days | @days_var } ]
        [ [ , ] PASSWORD =
        { password | @password_variable } ]
        [ [ , ] FORMAT | NOFORMAT ]
        [ [ , ] { INIT | NOINIT } ]
        [ [ , ] MEDIADESCRIPTION = { 'text' | @text_variable } ]
        [ [ , ] MEDIANAME =
        { media_name | @media_name_variable } ]
        [ [ , ] MEDIAPASSWORD =
        { mediapassword | @mediapassword_variable } ]
        [ [ , ] NAME =
        { backup_set_name | @backup_set_name_var } ]
        [ [ , ] NO_TRUNCATE ]
        [ [ , ] { NORECOVERY | STANDBY = undo_file_name } ]
        [ [ , ] { NOREWIND | REWIND } ]
        [ [ , ] { NOSKIP | SKIP } ]

```

```

        [ [ , ] { NOUNLOAD | UNLOAD } ]
        [ [ , ] RESTART ]
        [ [ , ] STATS [ = percentage ] ]
    ]
}

```

We would like to explain the TRUNCATE options:

- *NO\_TRUNCATE*: This option indicates that the log should be backed up, but committed transactions shouldn't be removed from it. The SQL-DMO constant for this is

```
SQLDMOBACKUP_Log_NoTruncate
```

- *NO\_LOG*: This option tells SQL Server to get rid of the committed transactions in the log and to not bother about backing them up to a physical file or tape. You don't need to specify a backup device or location for the option, as the log doesn't get backed up. This creates a hole in your recovery options because you're missing some transactions. It's advisable to follow this type of log backup with a full database backup to restore your ability to recover from a disaster. The SQL-DMO constant for this is

```
SQLDMOBACKUP_Log_NoLog
```

- *TRUNCATE\_ONLY*: This option is the same as NO\_LOG. The SQL-DMO constant for this is

```
SQLDMOBACKUP_Log_TruncateOnly
```

## Restoring the Database

The primary reason you create a backup is that in the event of a disaster you can have a good go at re-creating your environments with as little loss of data as possible. The following code listings show how to restore a database and log. The code is taken from SQL Server Books Online.

### T-SQL

```

RESTORE DATABASE { database_name | @database_name_var }
[ FROM < backup_device > [ ,...n ] ]
[ WITH
    [ RESTRICTED_USER ]
    [ [ , ] FILE = { file_number | @file_number } ]

```

```

[ [ , ] PASSWORD =
{ password | @password_variable } ]
[ [ , ] MEDIANAME = { media_name | @media_name_variable } ]
[ [ , ] MEDIAPASSWORD =
{ mediapassword | @mediapassword_variable } ]
[ [ , ] MOVE 'logical_file_name'
TO 'operating_system_file_name' ]
[ ,...n ]
[ [ , ] KEEP_REPLICATION ]
[ [ , ] { NORECOVERY | RECOVERY | STANDBY = undo_file_name } ]
[ [ , ] { NOREWIND | REWIND } ]
[ [ , ] { NOUNLOAD | UNLOAD } ]
[ [ , ] REPLACE ]
[ [ , ] RESTART ]
[ [ , ] STATS [ = percentage ] ]
]

```

The following code shows how to restore a log:

```

RESTORE LOG { database_name | @database_name_var }
[ FROM < backup_device > [ ,...n ] ]
[ WITH
[ RESTRICTED_USER ]
[ [ , ] FILE = { file_number | @file_number } ]
[ [ , ] PASSWORD = { password | @password_variable } ]
[ [ , ] MOVE 'logical_file_name' TO 'operating_system_file_name' ]
[ ,...n ]
[ [ , ] MEDIANAME = { media_name | @media_name_variable } ]
[ [ , ] MEDIAPASSWORD = { mediapassword | @mediapassword_variable } ]
[ [ , ] KEEP_REPLICATION ]
[ [ , ] { NORECOVERY | RECOVERY | STANDBY = undo_file_name } ]
[ [ , ] { NOREWIND | REWIND } ]
[ [ , ] { NOUNLOAD | UNLOAD } ]
[ [ , ] RESTART ]
[ [ , ] STATS [= percentage ] ]
[ [ , ] STOPAT = { date_time | @date_time_var }
[ [ , ] STOPATMARK = 'mark_name' [ AFTER datetime ]
[ [ , ] STOPBEFOREMARK = 'mark_name' [ AFTER datetime ]
]
]
]

```

The syntax for restoring a database and restoring a log is rather similar, but once again we'll explain those options that are perhaps not obvious.

- *FILE*: If you back up to a device on a number of occasions, then you'll need to specify this option. File numbers increment in multiples of one and are allocated on writing to the device. This will enable you to specify the particular backup on the device you require.
- *MOVE*: This option is used when you want or need to place the physical files in a different place than where they were backed up from. For example, say you have a database that has both its data file and log file on the F drive. You want to restore that database to another server, but that server only has a C drive and a D drive. Because the database knows where it previously lived on the disks, it defaults to that location. In this example, you don't have an F drive, so you need to tell SQL Server to move the files.
- *NORECOVERY/RECOVERY/STANDBY*: If you specify *RECOVERY*, then you'll bring the database up and make it available to users. You won't be able to apply further backups. If you use *NORECOVERY*, then you can apply more backups to the database, but the database is unavailable. Finally, if you choose *STANDBY*, then you're able to apply more backups to the database and you also make the database *READ-ONLY*. With this option, you also need to specify an undo file so you can roll back any transactions.
- *REPLACE*: You use this option if you have an existing database and you want to apply the backup of a different database to it. You force the restore over the top.
- *STOPAT*: This option allows you to specify exactly when you want to restore, down to the thousandth of a second. For example, a user may have dropped all the tables in the database at 2:30 P.M., so you would need to restore to just before that point in the log.
- *STOPATMARK/STOPBEFOREMARK*: SQL Server 2000 allows you to mark the log, and you can then use that mark as a reference point for your restores. For example, say you successfully do a load of a database and you mark the log. You carry on and manage to mess things up later on. You can choose to restore to your previous mark.



## Implementing the Restore

In the next example you'll restore a database from a full database backup and a transaction log backup. The database backup will be on a backup device, and the transaction log backup will be on a physical file.

```
RESTORE DATABASE MyDatabase FROM MyBackupDevice
WITH FILE = 1, NORECOVERY
```

```
RESTORE LOG MyDatabase FROM
disk = 'C:\Logfilebackups\LogFordatabase.bak'
WITH RECOVERY
```

If you want to look at the details of a backup before you go and restore it, the following three commands won't actually restore the database but will just tell you about the backup:

```
RESTORE FILELISTONLY FROM ...
RESTORE LABELONLY FROM ...
RESTORE HEADERONLY FROM ...
```

Figure 2-8 shows the Restore database screen in Enterprise Manager. Here you decide which database to restore and also where you restore the database from.

You can access the screen in Figure 2-8 in a number of ways. We include only two here for brevity.

- Right-click the database itself or the Databases folder and choose All Tasks followed by Restore Database.
- From the Tools menu, choose Restore Database.

On this screen, you specify the database you want to restore, which can be either an existing database or a new one. You also indicate what type of backup it is and where you want to get the backup from. Figure 2-9 shows the options available for placing the restored files and what state to leave the database in.

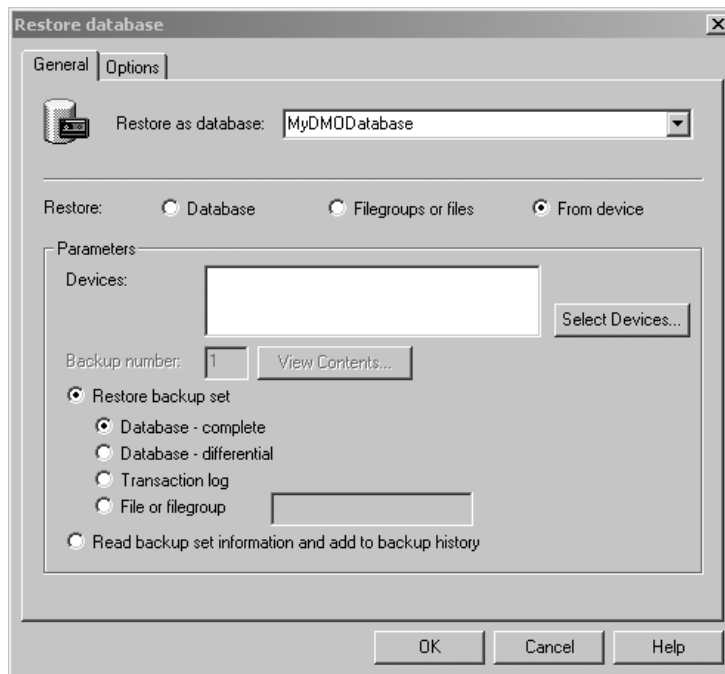


Figure 2-8. The Restore database screen in Enterprise Manager

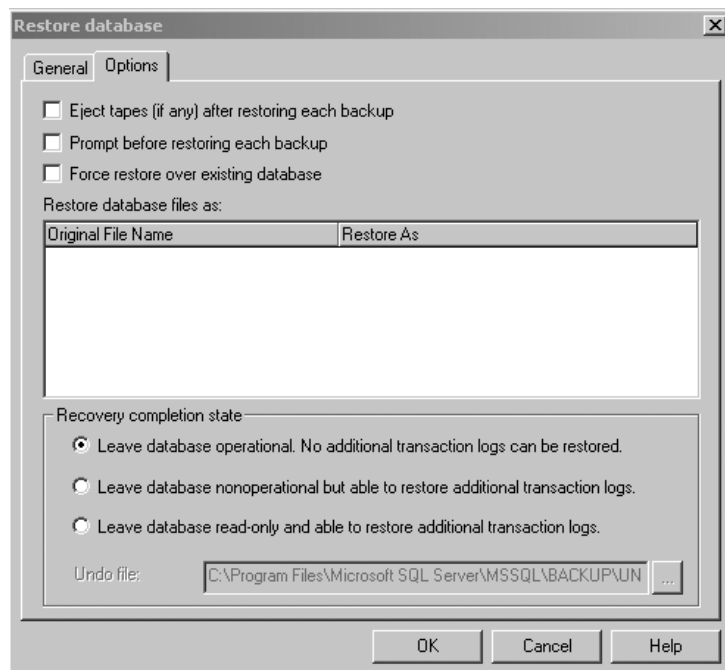


Figure 2-9. The Options tab in the Restore database screen in Enterprise Manager

From the Restore database Options tab, you can choose to move the physical files to a different location (remember the MOVE option in T-SQL). You can also specify the recovery state of your database (RECOVERY, NORECOVERY, or STANDBY). Enterprise Manager is quite easy to get along with.




---

**NOTE** Enterprise Manager isn't particularly good at refreshing, and here's an example of how you can get caught out with it. Say you choose to restore a particular database. The "Restore as database" box on the first screen of Enterprise Manager contains the database name. You then choose to restore from a device with multiple backups on it. Backup number 1 is the default choice, so you see the logical names and physical names of the backup on the Options tab. If the backup is of a different database, then you have incorrectly named files. This is problematic when the database from which the backup was taken is on the same server. And, if you choose to use a different FILE number on the device, Enterprise Manager doesn't update the view of the logical and physical files. Now you're left with a database with one name, logical and physical files from a different database, and a FILE number on the backup device from a completely different backup.

---

### *Restoring Using SQL-DMO*

To illustrate how you can do restores in SQL-DMO, we've built a very small application that does most of what Enterprise Manager does. (It actually does refreshing better than Enterprise Manager.) We've added a tab for you to look at the data definition language (DDL) you're creating so it can serve as a learning tool for what you need to do in T-SQL as well. Figure 2-10 shows our application prompting for a database to restore.

**Restore**

General Options DDL

Server: AM5

Restore As Database: MyDMODatabase

Backup Location Or Device: My Backup Device

Type Of Restore: ☒ Database ☐ Log

Available Backups

Backup Type	Position	Database Name	Backup Finish Date
FULL	1	MyDMODatabase	2002-08-10 11:15:04.000
FULL	2	MyDMODatabase	2002-08-10 11:15:16.000
DIFF DB	3	MyDMODatabase	2002-08-10 11:15:30.000
LOG	4	MyDMODatabase	2002-08-10 11:15:41.000

Generate Restore

Figure 2-10. The first screen of our application

The first screen is where you choose the server, database, and backup device that you want to restore from. You can, if there are multiple backups on the device (as there are in Figure 2-10), choose which backup device you want to restore from. Figure 2-11 shows our application asking where we want to move a file.

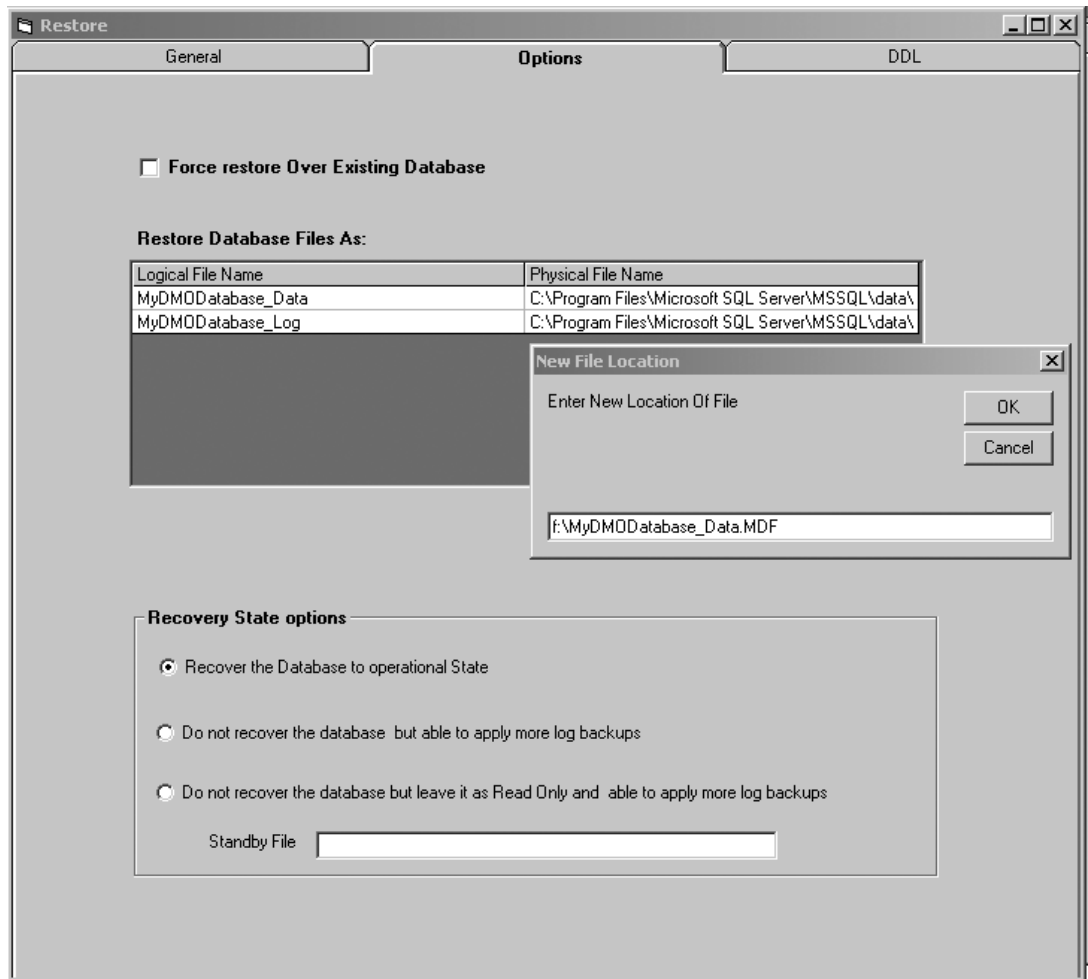


Figure 2-11. Prompting for a restore location

The second screen is where you indicate when you restore the files whether you want to keep them in the same physical location or move them. (In Figure 2-11, you can see that we've chosen to move the file. We made the other screen pop up by clicking the Physical File Name we wanted to change.) The second screen is also the place you determine what condition you want the database left in after the restore.

The DDL screen (see Figure 2-12) generates a T-SQL statement to show you what it is you'll be executing against SQL Server. This can be an excellent learning tool for people who are more comfortable with the GUI but who are looking to learn the T-SQL equivalent.

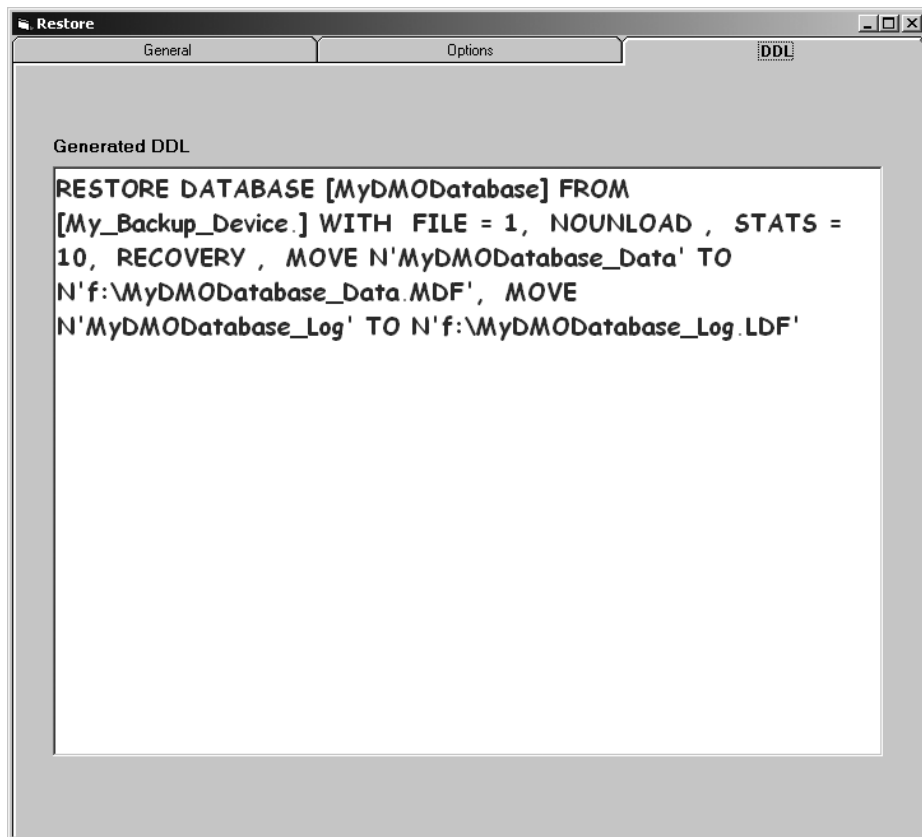


Figure 2-12. The DDL screen

A lot of the code for the application follows. We've chosen not to go through every single line of code. Rather, we explain those code lines that are pertinent and that we think are useful.

### *Specifying the Backup Origin*

Once you've chosen your server and database for the restore, you need to specify where the backup will be coming from (i.e., its origin). In our application, that

origin can be one of two places: a predefined backup or a physical file. If you click the ellipses to the right of the Backup Location or Device list box, you're taken to the screen shown in Figure 2-13. This screen allows you to select the location.

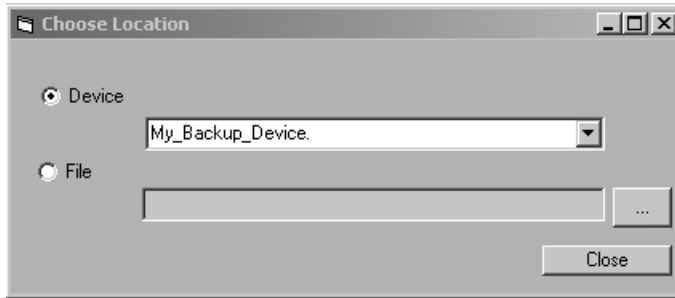


Figure 2-13. Selecting the location of the backup

To populate the combo box with the list of backup devices available, you simply loop through the backup devices of the server. The following code goes through the backup devices defined on the server and adds them to the combo box display in the application:

```
Private Sub ShowBackupDevices(servername As String)

Set oServer = New SQLDMO.SQLServer
oServer.LoginSecure = True
oServer.Connect servername

cboDevices.Clear

For Each oDevice In oServer.BackupDevices
    cboDevices.AddItem oDevice.Name
Next oDevice

oServer.DisConnect
Set oServer = Nothing
Set oDevice = Nothing

End Sub
```

You can choose either a device or a file. Once you've chosen a device or a file, your choice is entered in the list box showing backup devices back on the first

screen. You need to have a look at what backups, if any, exist on this device, so you need to execute the equivalent of RESTORE HEADERONLY for the device you select.

```
Private Sub ShowContentsOfBackup(servername As String, devicename As String)

Dim oHeader As SQLDMO.QueryResults
Dim DeviceCounter As Integer
Dim i As Integer
Dim Backup_type As String
DeviceCounter = 0
msf_Available.Rows = 1

Set oServer = New SQLDMO.SQLServer
Set oRestore = New SQLDMO.Restore

oServer.LoginSecure = True
oServer.Connect servername
```

### *Looping Through the Devices*

You'll need to check if the name you passed is a device or an ad hoc file, as this will influence which property of the Device object you supply. Here you check whether there is a device in the Devices collection with the same name as the text you chose in the list box. If there is, you set DeviceCounter = 1 to indicate that.

```
For Each oDevice In oServer.BackupDevices
    If oDevice.Name = devicename Then
        DeviceCounter = 1
    End If
Next oDevice
```

Once you find the device with the same name, you want to know what type of device it is.

```
If DeviceCounter = 0 Then
    oRestore.Files = devicename
Else
    oRestore.Devices = devicename
End If
```



Here's where you read the contents of the backup header into a QueryResults object ready for processing. A QueryResults object is very much like a table with rows and columns, so you just go through it processing the rows and columns.

This is the RESTORE HEADERONLY part. It takes a connected SQLServer object as an argument.

```
Set oHeader = oRestore.ReadBackupHeader(oServer)
```

```
For i = 1 To oHeader.Rows
```

Part of the results of ReadBackupHeader is the type of backup. Unfortunately, this is returned to you as an integer. You convert the unintelligible integer value for the backup type into something you can read and comprehend without opening a reference book. These are all the possible values you could ever have. What Microsoft has done with the number 3, we don't know.

### *Iterating Through the Possible Backup Types*

```
Select Case oHeader.GetColumnString(i, 3)
  Case 1
    Backup_type = "FULL"
  Case 2
    Backup_type = "LOG"
  Case 4
    Backup_type = "FILE"
  Case 5
    Backup_type = "DIFF DB"
  Case 6
    Backup_type = "DIFF FILE"
End Select
```

Next, add your required column and row values to the flex grid, which shows you the contents of the backup device you specified. Here we chose to only process certain values from the results, as we wanted to imitate Enterprise Manager. The GetColumnString method of the QueryResults object takes two arguments: row and column, in that order.

```
msf_Available.AddItem Backup_type & vbTab &
oHeader.GetColumnString(i, 6) & vbTab &
oHeader.GetColumnString(i, 10) & vbTab &
oHeader.GetColumnString(i, 19)
Next i
```

```
oServer.DisConnect
Set oServer = Nothing
Set oRestore = Nothing
```

```
End Sub
```

### *Choosing the Backup to Restore*

The next thing you need to do once you click the grid with your list of possible backups is have it populate the grid on the Options tab with the logical and physical names of the particular backup you selected. This section contains the calling procedure followed by the actual workhorse.

Because a single backup device may have multiple backups on it, you need to specify which one you're talking about. The way you managed to pass the FileNumber was to capture where you clicked the other grid and read off the FileNumber column for that particular row.

```
Private Sub msf_Available_Click()

If msf_Available.Row <> 0 Then
ShowMeFileDetails cboServers.Text, _
lstdevices.Text, _
msf_Available.TextMatrix(msf_Available.Row, 1)
End If
End Sub


Private Sub ShowMeFileDetails(servername As String,
    devicename As String, filenumber As Integer)

Dim oFileResults As SQLDMO.QueryResults

Dim DeviceCounter As Integer
Dim i As Integer

DeviceCounter = 0

msf_FileList.Rows = 1
```

```
Set oServer = New SQLDMO.SQLServer
Set oRestore = New SQLDMO.Restore
```

```
oServer.LoginSecure = True
oServer.Connect servername
```

Again, as before, you need to process the device name in the Devices list:

```
For Each oDevice In oServer.BackupDevices
    If oDevice.Name = devicename Then
        DeviceCounter = 1
    End If
Next oDevice
```

```
If DeviceCounter = 0 Then
    oRestore.Files = devicename
Else
    oRestore.Devices = devicename
End If
```

```
oRestore.filename = filename
```

### *Configuring the Restore*

Up until now, the code has been exactly the same as when you wanted to populate the other grid with details of your backup headers, but this is the part that you do the equivalent of RESTORE FILELISTONLY. It returns the result as a QueryResults object and takes a connected SQLServer object as an argument.

```
Set oFileResults = oRestore.ReadFileList(oServer)
```

```
For i = 1 To oFileResults.Rows
```

Add your required column and row values to the flex grid, which shows you the file details of the backup device you specified. You only want two of the returned column values: the logical and physical names of the backup you specified.

```

        msf_FileList.AddItem
    oFileResults.GetColumnString(i, 1) &
vbTab & oFileResults.GetColumnString(i, 2)
Next i

End Sub

```

On the Options tab, you have the ability to change the physical location of files once they're restored. The way you do that in the application is to click the location you want to change and enter into the prompt the new location (you can see this in Figure 2-11). This action is the equivalent of the WITH MOVE option in T-SQL. Here's the very simple code that does this:

```

Private Sub msf_FileList_Click()
Dim strNewFile As String

```

Only if you click the column with the physical location should you be prompted. If you don't enter anything, then you want to leave the value as is.

```

If msf_FileList.Col = 1 Then
    strNewFile = _
    InputBox("Enter New Location Of File", _
    "New File Location", msf_FileList.Text)
    If strNewFile = "" Or msf_FileList.Col <> 1 Then
        msf_FileList.Text = msf_FileList.Text
    Else
        msf_FileList.Text = strNewFile
    End If
End If

End Sub

```

All the other options are specific to your restore and will only be relevant once you tell the application to do the restore itself, so let's now look at the procedure that calls the actual restore procedure. We've used this to do the checking:

```

Private Sub cmdRDDL_Click(Index As Integer)

```

On the form you've created a control array of the button that will generate the DDL only, and the one that will generate DDL and do the restore. The reason for this is that it means you can reuse the same code.

```
If cboServers.Text <> "" And cboDatabases.Text <> ""  
And lstdevices.Text <> "" And msf_Available.Rows > 1  
Then
```

You need to log onto your SQL Server:

```
Set oServer = New SQLDMO.SQLServer  
Set oRestore = New SQLDMO.Restore
```

```
oServer.LoginSecure = True  
oServer.Connect cboServers.Text
```

```
Dim iLeftInState As Integer  
Dim iDBRestore As Integer  
Dim iForce As Integer  
Dim ideviceYN As Integer  
Dim iMoveFiles As Integer  
Dim iBackupType As Integer  
Dim strMoveFiles As String  
Dim qry_Comparison As SQLDMO.QueryResults  
Dim i As Integer
```

```
iForce = 0  
ideviceYN = 0  
iBackupType = 1  
strMoveFiles = ""  
iMoveFiles = 0
```

Here you check to see if you've opted to force your restore over the top of an existing database (the REPLACE option in T-SQL):

```
Select Case chkForce.Value  
  
Case vbChecked  
iForce = 1  
End Select
```

You need to check your devices again because later you're going to need to do some comparisons with the ReadFileList of the Restore object:

```
For Each oDevice In oServer.BackupDevices
    If oDevice.Name = lstdevices.Text Then
        ideviceYN = 1
    End If
Next oDevice
```

### *Choosing the Restore Type*

What type of restore do you want to do? Your options are a database backup and a log backup, and they're set by the option buttons on the first screen of the application.

```
Select Case optDatabase.Value

Case False
    iBackupType = 0
End Select
```

How do you want to leave the database? You set this on the Options tab of the application.

The following code shows how to recover the database after a restore:

```
If optrecover.Value = True Then
    iLeftInState = 1
```

This code will leave the database nonoperational but in a state in which you can apply more log backups afterward.

```
ElseIf optNonop.Value = True Then
    iLeftInState = 2
```

This option will leave the database in a read-only state, which is good for reporting. You can also restore more backups to the database.

```
ElseIf optStandby = True Then
    iLeftInState = 3
End If
```

If you choose to leave the database in a standby state, then make sure you have an undo file ready.

```

If iLeftInState = 3 And txtStandby = "" Then
    MsgBox "You have chosen to place the database in standby."
    & vbCrLf & "You need to specify an undo file",
    vbInformation, "Missing Undo File"
    Exit Sub
End If

```

Now for the tricky part. You need to compare the values for the placement of the files in the grid with those that you know are in the backup set for the backup you've chosen row for row. If you find any values that are different, then this will need to be your move string for the restore.

```

If ideviceYN = 1 Then
    oRestore.Devices = lstdevices.Text
Else
    oRestore.Files = lstdevices.Text
End If

```

```
oRestore.filename = msf_Available.TextMatrix(msf_Available.Row, 1)
```

Here you're doing the equivalent of RESTORE FILELISTONLY. The results are read into a QueryResults object.

```
Set qry_Comparison = oRestore.ReadFileList(oServer)
```

This is the comparison. If you find a difference, you add the logical name and the new physical name from the grid to your string, which will tell SQL Server to move the files. You need to wrap the logical name and physical name in square brackets to cater for spaces in either name. SQL Server, like a lot of other products, can find it difficult to work with spaces, so wrapping in square brackets smoothes things over (don't worry—they don't come out in your code).

```

For i = 1 To qry_Comparison.Rows
    If msf_FileList.TextMatrix(i, 1) <> qry_Comparison.GetColumnString(i, 2) Then
        strMoveFiles = _
strMoveFiles & "[" & msf_FileList.TextMatrix(i, 0) & _
        "],[" & msf_FileList.TextMatrix(i, 1) & "], "
    End If
Next i

```

If you found any files that need moving, then after you've built up this string you're going to have an extra comma at the end of the string, so here you trim

that off and set an indicator that you'll later use to indicate that you have files that need moving.

```
If Len(strMoveFiles) > 0 Then
    strMoveFiles = Mid(strMoveFiles, 1, Len(strMoveFiles) - 1)
    iMoveFiles = 1
End If
```

Here all you do is check whether the button you clicked was the Generate DDL only or the Restore button. Based on that, you execute the procedure that does what you've been building up to.

```
Select Case Index
```

The following code only generates the DDL:

```
Case 0
GenerateDDLAndRestore "", cboDatabases.Text,
iForce, lstdevices.Text, ideviceYN, iBackupType,
oRestore.filename, iLeftInState, iMoveFiles, strMoveFiles, txtStandby.Text
```

The following code will generate some DDL statements and actually do the restore:

```
Case 1
GenerateDDLAndRestore "", cboDatabases.Text,
iForce, lstdevices.Text, ideviceYN, iBackupType,
oRestore.filename, iLeftInState, iMoveFiles,
strMoveFiles, txtStandby.Text
GenerateDDLAndRestore cboServers.Text,
cboDatabases.Text, iForce, lstdevices.Text,
ideviceYN, iBackupType, oRestore.filename,
iLeftInState, iMoveFiles, strMoveFiles,
txtStandby.Text
```

```
End Select
```

```
End If
End Sub
```

There remains only one piece of code to go through: the actual restore procedure.



```

Private Sub GenerateDDLAndRestore(servername
As String, DatabaseName As String, Force
As Integer, location As String, deviceYN As Integer,
DBRestore As Integer, filenumber As Integer,
LeftInState As Integer, MoveFiles As Integer,
movestring As String, standbyfilesloc As String)

On Error GoTo err_handler

Set oRestore = New SQLDMO.Restore

```

You now check with the following code to see if you're using a device or a file:

```

If deviceYN = 1 Then
    oRestore.Devices = location
Else
    oRestore.Files = location
End If

```

Here you indicate whether or not you force this backup over the top of the existing database:

```

If Force = 1 Then
    oRestore.ReplaceDatabase = True
End If

```

Here you indicate what type of backup you're doing:

```

If DBRestore = 1 Then
    oRestore.Action = SQLDMORestore_Database
Else
    oRestore.Action = SQLDMORestore_Log
End If

```

Are you moving the files at all? If the answer is yes (MoveFiles = 1), then you need to supply a string in the form of *logical file name, physical file name* for each file you're moving (remember the comparison you did between the results of the ReadFileList and the values in the grid in the previous procedure)—that is, Logical\_file\_1, c:\MyData\physical\_file\_1, logical\_file\_2, c:\MyData\physical\_file\_2.

```

If MoveFiles = 1 Then
    oRestore.RelocateFiles = movestring
End If

```

In the following code you indicate which database you want to restore:

```
oRestore.Database = DatabaseName
```

Here you set the recovery option:

```
Select Case LeftInState
```

If you specify the following option you recover the database:

```
Case 1  
oRestore.LastRestore = True
```

Specifying the following option leaves the database nonoperational:

```
Case 2  
oRestore.LastRestore = False
```

Finally, choosing this option leaves the database in Standby mode:

```
Case 3  
oRestore.LastRestore = False  
oRestore.StandbyFiles = standbyfilesloc
```

```
End Select  
oRestore.filename = filename
```

Because you want to reuse this procedure for DDL only or DDL and actual restore, you include the following clause here. If no servername is passed, you only need to generate the DDL.

```
If servername <> "" Then  
    Set oServer = New SQLDMO.SQLServer  
    oServer.LoginSecure = True  
    oServer.Connect servername
```

Here's the actual restore statement that takes an argument of a connected SQLServer object:

```
oRestore.SQLRestore oServer  
End If
```

This is the statement that will generate your T-SQL on the third tab of the application:

```

txtDDL.Text = oRestore.GenerateSQL

Exit Sub

err_handler:
MsgBox "The restore has failed because... " & vbCrLf & Err.Description
Exit Sub

End Sub

```

## The 15-Minute Database Checker

When we're out and about at client sites, we often find that databases are created on servers and no one will bother to tell us. Without fail, you can guarantee that when something goes wrong with the database and the client needs it restored, then ours will be the first door they come knocking on. Even if we didn't know anything about the database, it's still rather embarrassing to find that we can't recover the client's data. This is the reason we built this little application. We have made it a rule that if nobody tells us they've built a database server, and hence we haven't registered it in our Enterprise Manager console, then as far as we're concerned it doesn't exist. Fortunately, people do provide us with this information for the most part.

Imagine this scenario: You have a server that has a series of databases that are reliant upon a full database backup being done on a night and a differential backup being done every day at 2-hour intervals between 8:00 A.M. and 8:00 P.M. The full backups are done and scheduled through maintenance plans, but unfortunately you can't do the same with a differential backup. This means you need to write your own procedure, but you want to just let it run and capture any databases on the server, and back it up when the time comes. The procedure to do so may look like this:

```

DECLARE @DBName sysname
DECLARE @EXECString varchar(255)
DECLARE cur_Differentials CURSOR FAST_FORWARD
FOR
select
    Name from master..sysdatabases where name not in ('master','model','tempdb')

OPEN cur_Differentials

FETCH NEXT FROM cur_Differentials INTO @DBName

```

```

WHILE (@@FETCH_STATUS<>-1)
BEGIN

    set @EXECString = _
    'BACKUP DATABASE
    [' + @DBName + '] to DISK = ''\\NetworkServer\diffBackups\Server1\' +
    [' + @DBName + '].DIFF' WITH INIT,differential'
    EXEC(@EXECString)

    FETCH NEXT FROM cur_Differentials INTO @DBName
End

Close cur_Differentials
DEALLOCATE cur_Differentials

```

If you were to create a database during the working day, then at the next interval you would attempt to back up that database using a differential backup. Shortly after that, you would get a message telling you that the differential backup job had failed because of the new database.

This application will search through all databases on all your registered servers and look at when they were created. If it finds one that was created in the past 15 minutes, then it will back it up to a default location, *\\MyNetworkServer\FullBackupShare\<Server\_Name>\<Database\_Name>\_<yyyymmdd>.bak*.

The following code shows how to call the procedure:

```

Option Explicit
Sub main()
ListNewDatabases 15
End Sub

```

The following code shows the procedure:

```

Private Sub ListNewDatabases(MinsAgo As Integer)
Dim oServer As SQLDMO.SQLServer
Dim oBackup As SQLDMO.Backup
Dim oDatabase As SQLDMO.Database
Dim OutputString As String
Dim oRserver As SQLDMO.RegisteredServer

On Error GoTo err_handler

```

```

Set oServer = New SQLDMO.SQLServer
Set oBackup = New SQLDMO.Backup

For Each oRserver In SQLDMO.ServerGroups(1).RegisteredServers

    oServer.LoginSecure = True
    oServer.Connect oRserver.name

```

The procedure is self-explanatory so far. Here's where you check each database, and if it isn't Master, Model, msdb, or tempdb, then you log that fact and back it up:

```

    For Each oDatabase In oServer.Databases
        If oDatabase.Name <> "master" and
oDatabase.name <> "tempdb" and
oDatabase.Name <> "model" and oDatabase.Name <> 'msdb'
Then

```

In the following code you check for any newly created databases:

```

If DateDiff("n", Left(oDatabase.CreateDate, 19),
Format(Now(), "dd-mm-yyyy hh:mm:ss")) <= MinsAgo Then
    OutputString = OutputString & "Database " &
oDatabase.Name & " on " & oServer.Name &
" was created on " & Left(oDatabase.CreateDate, 19)
& " by " & oDatabase.Owner & vbCrLf &
"The database was backed Up to \\MyNetworkServer\FullBackupShare\" & _
oServer.Name & "\" & oDatabase.Name &
"_" & Format(Now(), "yyyymmmy") & ".bak" &
vbCrLf & vbCrLf
oBackup.Database = oDatabase.Name
oBackup.Files = "\\MyNetworkServer\FullBackupShare\"
& oServer.Name & "\" & oDatabase.Name & "_" &
Format(Now(), "yyyymmmy") & ".bak"
oBackup.SQLBackup oServer
End If

End If
Next oDatabase

oServer.DisConnect

Next oRserver

```

If OutputString has anything in it, this means something has been created:

```
If OutputString <> "" Then  
  
Open "c:\NewDBs.txt" For Output As #1  
  
Print #1, OutputString  
  
End If  
  
Exit Sub  
Close #1  
  
err_handler:  
Resume Next  
  
End Sub
```

This is a wonderfully simple and amazingly useful application. It enables you to be proactive in your duties over any number of servers. You complement this application by having SQL Server check for the existence of the log output file from the application, and if it finds the file, it sends it to you so you know what's going on.

### *Permissions for Backing Up the Database*

BACKUP DATABASE and BACKUP LOG permissions default to members of the db\_owner fixed database role, who can transfer permissions to other users, and members of the db\_backupoperator fixed database role.

### *Permissions for Restoring the Database*

If the database being restored doesn't exist, the user must have CREATE DATABASE permissions. If the database does exist, RESTORE permissions default to members of the sysadmin fixed server role and members of the db\_owner fixed database role.

## Summary

Doing backups and restores is something we believe strongly in. In this chapter, we showed you some useful applications in SQL-DMO. We didn't cover every option that you could possibly use for backups and restores, but we did cover those points we think are most relevant. We also managed to develop a learning tool as a side effect of the restore application. Being asked to restore a database we have no idea about is something we would like to avoid, as we said earlier, and thankfully we have by using the application in this chapter.

In the next chapter you're going to add users to the databases and the server. Once the users are in there, you'll look to give them some permissions.



<http://www.springer.com/978-1-59059-040-9>

Real-World SQL-DMO for SQL Server

Mitchell, A.; Allison, M.

2003, XXVII, 432 p. 174 illus., Softcover

ISBN: 978-1-59059-040-9

A product of Apress