http://www.sqlservercentral.com/articles/Administration/2605/
Printed 2007/10/03 08:41PM

Moving the SQL 2005 System Databases

By Vince Iacoboni, 2006/09/25

A typical installation of Microsoft SQL 2005 ends up with system databases buried five levels deep under the C:\Program Files directory. Most DBAs prefer their data, including system databases, reside on a different drive than the executables. But moving the system databases, including master and the new hidden mssqlsystemresource database, is non-trivial, even scary. After doing this several times manually, I wrote a script to perform this task.

System Databases

The system databases, according to Microsoft (System Databases) include:

- Master - Holds the description of other databases, logins, and master-only system tables.
- Model - The snapshot of new databases that are created.
- Msdb - Contains code and data for the SQL Server Agent and SQL Server Management Studio.
- Tempdb - Where #tables and ##tables get created, and internal sorting space is written.
- Mssqlresourcedb - Internal hidden read-only database containing system objects

Other databases, such as distribution, may also be included depending upon your configuration. In this article, I refer to only the above five databases.

Why would you want to move these databases? A few reasons. Personally, I don't like the idea of important database changes happening somewhere in the depths of the Program Files directory. I'd rather know that all my databases are on an easy to find directory structure. More importantly, database storage is usually in a RAID array or SAN separate from the C: boot disk. Why not have the protection and performance those disks afford on the critical system data? If you lose your boot disk, you can reinstall the OS and SQL and after pointing to the right locations, you can be back up quickly with no data loss.

A Special Database Requires Special Care

The newest member of the system databases is mssqlsystemresource. Its hard to tell exactly what it does, but Books Online (Resource Database) indicates that "SQL Server system objects, such as sys.objects, are physically persisted in the Resource database." We are also told there that the only supported operation on this database is to move it, so we're not breaking any rules! Another tidbit: this database can't be backed up. It must be copied like an EXE file.

A very important fact that is easy to miss is that this database MUST appear in the same location as the master database. I didn't notice this restriction at first, and it caused some strange errors when we upgraded to SQL 2005 SP1. Turns out the service pack upgraded an older unused copy of the resource database, but not the copy in the same location as master. To play it safe, don't keep unused copies of this database in the default location, but rather keep backup copies in clearly marked backup directory.

In order to move this database, we'll need special startup and trace flags and SQL DDL commands. And, of course, we need to ensure the path for the master database (specified in the service startup parameters

in the registry) and the location of the mssqlsystemresource.mdf and .ldf files are the same.

Steps to Moving

The steps to moving the databases are outlined well in Books Online (Moving System Databases). To move the databases, we'll need to do the following:

- Use ALTER DATABASE MODIFY FILE to tell the master database where most of the system databases will reside.
- Update the service startup parameters in the registry so that the service finds the master database and log.
- Stop the service to unlock the files.
- Move the files to the new location.
- Start the service with the -f (minimal configuration) flag and the -T3608 trace flag to prevent automatic recovery.
- Use ALTER DATABASE MODIFY FILE to record the new location of the mssqlsystemresource database.
- Move the mssqlsystemresource file.
- Set mssqlsystemresource database to read only.
- Stop and Start the SQL Server Service in normal mode.

Examining The Code

Download the code

The .CMD script MUST be run on the SQL Server machine itself, and the accompanying .SQL must live in the same directory as the .CMD script. The script requires two parameters - the instance name and the full path where the data files will be moved (leave off the trailing backslash). If using the default instance, provide MSSQLSERVER for the instance name.

After initial setup and parameter checking, SQLCMD is called with the -Q parameter to accomplish step a. for model, msdb, and tempdb. Then SQLCMD is called again using the -i parameter to run the sql script. Interestingly, though, it is called from the FOR command. This powerful but arcane command can do a lot in a batch script, but it is often far from obvious just what its doing.

In this case, FOR is used as means to communicate from SQL back to the calling script. The script will return the old path of the system files, information we will need to move the files in a later step. But how to get the variable from SQL to the calling batch script? This was enough of a challenge that I considered writing it all as a SQL script using SQLCMD's new parameters, but again, going from SQL @variables to SQLCMD $(variables) proved difficult. I considered writing the script in Perl, but didn't want to require everyone to download the Perl runtime. I finally settled on the ugly, but working, FOR /F command to execute SQLCMD with an input file and return its result as an environment variable that can be used later in the script.

With the /F switch, FOR will run the single-quoted command specified in the parenthesis, then set the batch variable (%%s) to the output of the command. Specifying -h-1 on the SQLCMD indicates no column headers and dash divider lines on output. We also use the "delims=;" option to have the batch variable read up to the first semicolon it sees, since the default action of delimiting on spaces would choke in the presence of "Program Files" and "Microsoft SQL Server".

The .SQL Script first uses xp_regread procedure to get the directory path (MSSQL.n) associated with an

instance name. Notice the use of the $(InstName) variable. SQLCMD, unlike OSQL, can easily read environment variables. The value of this key tells us where to search in the registry for the SQL Server service parameters. Plugging in the values, we read the SQLArg0, SQLArg1, and SQLArg2 values that contain the -d, -l, and -e parameters. There is no set order to these parameters, so we need to iterate through them all. We obtain the old path for sending back to the calling script, replace it with the new path, and call the xp_regwrite procedure to write it out.

The script can then proceed in a straightforward way to accomplish steps c-i.

Caveats

The code assumes that all of the system databases are in the same location. If you've already moved TEMPDB, for example, you might want to comment that portion of the code from the .CMD script.

The assumption is also made that the -d and -l parameters occur within the first three parameters of the service startup. Its not likely, but other options such as trace flags could be specified before these parameters and break this part of the code.

Before running this code, always backup your data files. Get a screen print of the Advanced tab in the SQL Server Configuration Manager, so that you can restore the registry values manually if necessary. Run it on a test server first if at all possible. If you don't have a test server, consider installing the server on your workstation to test it out.

Conclusion

Moving the system databases offers some real advantages but needs to be done carefully and consistently. The script presented here can help to accomplish this, particularly when run soon after installation of a new SQL instance.
