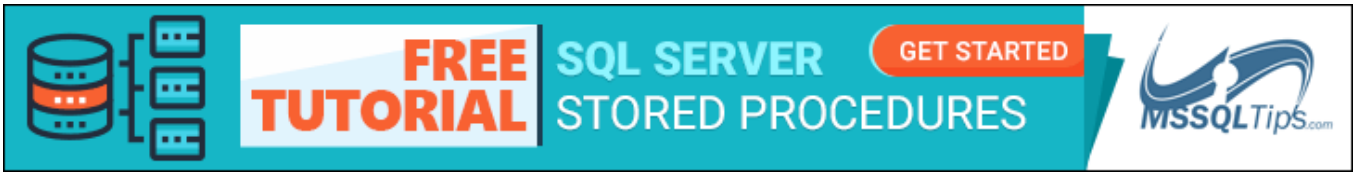
MSSQLTips Free SQL Server Conference Registration + Travel Expenses click to learn more

Backup SQL Server Databases with a Windows PowerShell Script



Free Webcast - Simplify SQL Server Administration

Problem

In a previous tip on Backup and Restore SQL Server databases programmatically with SMO, you've seen how you can use Windows PowerShell to backup and restore SQL Server databases. In this tip, I will cover how to use Windows PowerShell to generate SQL Server backups.

Solution

Generating and maintaining backups are the most important tasks that any DBA has to fulfill. And, as mentioned in the previous tip, SMO can be used to backup and restore a database. There are different ways to backup a database, depending on company policies on disaster recovery. We will be introducing a new SMO namespace that will allow us to create the Backup object. This is an added line in our previous PowerShell scripts to create a new instance of the Backup object.

```
$dbBackup = new-object ("Microsoft.SqlServer.Management.Smo.Backup")
```

The first property of the Backup object that we need to assign is the name of the database that we need to do a backup on. This is defined by the Database property of the Backup object. The code snippet below defines a Database property for the Backup object to perform a backup on the Northwind database.

```
$dbBackup.Database = "Northwind"
```

Whenever you generate or create backups, SMO considers this as a Device. This requires us to add the Backup file to the Devices collection of the Backup object, specifying the DeviceType as File. The first parameter in the AddDevice method of the Devices collection is the file name of the backup file with the full path as stored in the file system whereas the second parameter is the DeviceType. While SQL Server supports the use of devices, I normally recommend using the file system to easily identify the backup sets based on the file name using the proper naming convention (one thing you would not want to be worried about during disaster recovery is scrambling to find the valid backup sets).

```
$dbBackup.Devices.AddDevice("yourDBbackupFile.bak", "File")
```

The next important property of the Backup object that we are interested in is the Action property. The Action property sets the type of backup to be performed - Database, Log or File. The default value of the Action property is set to Database. This property is of importance based on the definition of our service level agreement as you don't just generate backups without understanding the defined SLAs for a specific database or application. For our example, we will define a FULL database backup

```
$dbBackup.Action = "Database"
```

Once we have defined the properties of our Backup object, it's just a matter of calling the SqlBackup method of the Backup object. The SqlBackup method performs the backup operation as defined by the properties you have set in the Backup object. It accepts a parameter of type Server which represents the SQL Server instance that you have defined as the source of the backup operation.

```
$dbBackup.SqlBackup($s)
```

Combining the code snippets above, here is a working Windows PowerShell script to perform a FULL database backup against the Northwind database, storing the backup file in your file system.

```
[System.Reflection.Assembly]::LoadWithPartialName('Microsoft.SqlServer.SMO') | out-null
$s = New-Object ('Microsoft.SqlServer.Management.Smo.Server') "LOCALHOST\SQL2005_1"

#Create a Backup object instance with the Microsoft.SqlServer.Management.Smo.Backup namespace
$dbBackup = new-object ("Microsoft.SqlServer.Management.Smo.Backup")
```

```
#Set the Database property to Northwind
$dbBackup.Database = "Northwind"

#Add the backup file to the Devices collection and specify File as the backup type
$dbBackup.Devices.AddDevice("D:\PSScripts\backups\NWind_FULL.bak", "File")

#Specify the Action property to generate a FULL backup
$dbBackup.Action="Database"

#Call the SqlBackup method to generate the backup
$dbBackup.SqlBackup($s)
```

Now, since you won't be performing backups of just a single database, it would be better if we loop the entire script in a **For-Each** cmdlet iterating thru the **Databases** collection of the **Server** object.

```
[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.SqlServer.SMO") | out-null
$s = new-object ("Microsoft.SqlServer.Management.Smo.Server") $instance

$bkdir = "D:\PSScripts\backups" #We define the folder path as a variable
$dbs = $s.Databases
foreach ($db in $dbs)
{
     if($db.Name -ne "tempdb") #We don't want to backup the tempdb database
     {
     $dbname = $db.Name
     $dt = get-date -format yyyyMMddHHmm #We use this to create a file name based on the timestamp
     $dbBackup = new-object ("Microsoft.SqlServer.Management.Smo.Backup")
     $dbBackup.Action = "Database"
     $dbBackup.Database = $dbname
     $dbBackup.Devices.AddDevice($bkdir + "\" + $dbname + "_db_" + $dt + ".bak", "File")
     $dbBackup.SqlBackup($s)
     }
}
```

While my only condition for generating backups is to exclude the **tempdb** database, you can include other conditions such as querying the database property if it is used for database mirroring (IsMirroringEnabled property) or specifying database names if you are sure enough that you won't be needing backups for them. While not really necessary, you also might want to include other Backup object properties such as BackupSetName, BackupSetDescription, and MediaDescription as they can be of help in case you wanted to find out the contents of your backup sets.

NOTE: If you have SQL Server 2008 Client Tools installed on your workstation where you intend to run this PowerShell script, it is important that you add a reference to the **Microsoft.SqlServer.SmoExtended** namespace. The Backup object and a few other objects were moved from the **Microsoft.SqlServer.Smo** to the **Microsoft.SqlServer.SmoExtended**. In the example I provided, I have SQL Server 2005 and SQL Server 2008 running on my machine, thus, requiring that I add a reference to the **Microsoft.SqlServer.SmoExtended** namespace. SQL Server MVP Allen White has blogged about Loading

SMO Assemblies in PowerShell and explains using a custom script to load appropriate SMO assemblies depending on the SQL Server version.

```
[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.SqlServer.SMO")| out-null
[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.SqlServer.SmoExtended")| out-null
```

Generating Differential and Log Backups

Depending on your service level agreement, you would also need to define either Differential or Log backups to complement your Full backups. This can be done by changing the Action property of the Backup object to either Log for Log backups or setting the Incremental property of the Backup object to a value of 1. Below is a code snippet for generating Log backups, simply by changing the Action property.

```
$dbBackup.Action = "Log"
```

Note that you can only do Log backups on databases that are not configured to use the SIMPLE recovery model. This means that your condition for generating Log backups would include a check on the RecoveryModel property

```
if($db.RecoveryModel -ne 3) #Don't issue Log backups for DBs with RecoveryModel=3 or SIMPLE
{
$dbname = $db.Name
$dt = get-date -format yyyyMMddHHmm #Create a file name based on the timestamp
$dbBackup = new-object ("Microsoft.SqlServer.Management.Smo.Backup")
$dbBackup.Action = "Log"
$dbBackup.Database = $dbname
$dbBackup.Devices.AddDevice($bkdir + "\" + $dbname + "_log_" + $dt + ".trn", "File")
$dbBackup.SqlBackup($s)
}
```

For Differential backups, you would need to replace the Action property with the Incremental property and set the value to 1

```
if($db.Name -ne "tempdb")
{
$dbname = $db.Name
$dt = get-date -format yyyyMMddHHmm #Create a file name based on the timestamp
$dbBackup = new-object ("Microsoft.SqlServer.Management.Smo.Backup")
$dbBackup.Incremental = 1
$dbBackup.Database = $dbname
$dbBackup.Devices.AddDevice($bkdir + "\" + $dbname + "_diff_" + $dt + ".bak", "File")
$dbBackup.SqlBackup($s)
}
```

Your backup strategies, depending on your service level agreement, will definitely include a combination of either Full and Differential and/or Log backups.

Next Steps

- Read more on the Backup class in SMO
- Review the tip Backup and Restore SQL Server databases programmatically with SMO
- Test how you can use Windows PowerShell to generate your SQL Server backups.

Last Update: 2009-10-20

You're invited
to our next free
SQL Server webcast!

click to learn more

Next Tip

Click Here

About the author

Edwin M Sarmiento is a Microsoft SQL Server MVP and Microsoft Certified Master from Ottawa, Canada specializing in high availability, disaster recovery and system infrastructures.

View all my tips

Related Resources

- More SQL Server DBA Tips...

Post a comment or let the author know this tip helped.

All comments are reviewed, so stay on subject or we may delete your comment. Note: your email address is not published. Required fields are marked with an asterisk (*).

*Name *Email Notify for updates

*** NOTE *** - If you want to include code from SQL Server Management Studio (SSMS) in your post, please copy the code from SSMS and paste the code into a text editor like NotePad before copying the code below to remove the SSMS formatting.

p

Send me SQL tips:

I'm not a robot

reCAPTCHA
Privacy - Terms

Save Comment

Tuesday, August 20, 2013 - 3:00:15 PM - bass_player Back To Top

Did you restart the SQL Server service? Plus, a simple TELNET test on the port number that SQL Server Express is listening on will tell you if the port is opened. Check the SQL Server error log for the IP address and the port number that SQL Server Express is listening on

Tuesday, August 20, 2013 - 12:42:01 PM - Patrick B Back To Top

@bass_player

TCP/IP Is already enabled. I have tried specifying the database by IP address and by it's UNC.

Tuesday, August 20, 2013 - 11:51:40 AM - bass_player Back To Top

Try these steps

1. **Open up the SQL Server Configuration Manager.** Programs / Microsoft SQL Server 2008 / Configuration Tools / SQL Server Configuration Manager.

2. **Expand** the SQL Server Network Configuration node and select the Protocols for SQLEXPRESS

3. **Enable TCP/IP** by right-clicking and selecting Enable, then OK.

4. Click on the SQL Server Services node and in the right panel right-click SQL Server (EXPRESS) and select restart to **restart the service**.

5. Right-click on the SQL Server Browser and select start to **start the browser service** if it isn't started already. This will allow you to access the SQL Express instance by the computer name.

Tuesday, August 20, 2013 - 8:45:47 AM - Patrick B Back To Top

I am trying to get this to work on a remote instance of SQL Server Express 2008 R2. I have SQL Manegement Tools for SQL 2005 installed on my local machine. I am able to create the new object but it will not connect to my remote server.

First question, is there a setting I need to enable to allow remote connections or is that even possible with a SQL Server Express 208 R2?

Monday, July 29, 2013 - 6:41:31 PM - Mark R Back To Top

I took the code listed in the box right under "it would be better if we loop the entire script in a **For-Each** cmdlet iterating thru the **Databases** collection of the **Server** object." and attempted to run that under powershell. The below is the resulting errors which look like to a 30+ year UNIX vet as "you didn't load a series of libraries/this is one of them thar object-driven shells and you didn't load the correct objects". Given this is my 1st attempt at powershell, my knowledge of what should/should not be pre-loaded is lacking.

Assume a new Windows 8 box, Microsoft 2012 SQL server and someone who's never touched Powershell. So what are the other magic incantations to make the script you give above work?

As taken from the above:

```
[System.Reflection.Assembly]::LoadWithPartialName("Microsoft.SqlServer.SMO") | out-null
$s = new-object ("Microsoft.SqlServer.Management.Smo.Server") $instance

$bkdir = "c:\Backups SQL" #We define the folder path as a variable
$dbs = $s.Databases
foreach ($db in $dbs)
{
    if($db.Name -ne "tempdb") #We don't want to backup the tempdb database
```