**A Simple way of Automating Scripts**

**By** [**Sadequl Hussain**](http://www.sqlservercentral.com/Authors/Articles/Sadequl_Hussain/911759/)**, 2011/12/30 (first published: 2009/07/30)**

Total article views: 33522 | Views in the last 30 days: 6867

http://www.sqlservercentral.com/Resources/Images/FilledStar.pnghttp://www.sqlservercentral.com/Resources/Images/FilledStar.pnghttp://www.sqlservercentral.com/Resources/Images/FilledStar.pnghttp://www.sqlservercentral.com/Resources/Images/FilledStar.pnghttp://www.sqlservercentral.com/Resources/Images/EmptyStar.png[Rate this](javascript:;) | http://www.sqlservercentral.com/Resources/Images/Discuss.gif  [Join the discussion](http://www.sqlservercentral.com/Forums/FindPost762163.aspx) | [[http://www.sqlservercentral.com/Resources/Images/Briefcase.gif](javascript:;) Briefcase](javascript:;) | http://www.sqlservercentral.com/Resources/Images/print.gif  [Print](http://www.sqlservercentral.com/articles/Automation/67547/)

**0**diggsdigg

[Delicious](http://delicious.com/post?url=http%3a%2f%2fwww.sqlservercentral.com%2farticles%2fAutomation%2f67547%2f&title=A+Simple+way+of+Automating+Scripts)

[Delicious](http://technorati.com/faves?add=http%3a%2f%2fwww.sqlservercentral.com%2farticles%2fAutomation%2f67547%2f)

[in**Share**](javascript:void(0);)

As a DBA, one of my core tasks has always been the rollout of database objects in a live environment or the execution of scripts written by others that would modify data. Typically, the code would be written by a developer, peer-reviewed and tested in a UAT environment, and then after a sign-off, e-mailed to me as a “request for change”. Sometimes the change would be ad-hoc - taking care of some emergency fix, at other times it would be a planned rollout with multiple scripts involved.

Although the latter type of rollouts would be usually accompanied by instructions about what order the scripts should be run, I always found the process tedious and susceptible to human error. If you have thirty to forty scripts to execute separately against five or six databases, chances are that one day you would miss one or two (or execute more than once) along the way.

The code that I am presenting here is a simple method of automating this task. Using this method, you can run multiple scripts in a pre-defined order against one or more databases. The script was written in SQL 2005, but it can be easily modified for SQL 2000.

**Assumptions and Prerequisites**

If you want your scripts to automatically execute in a pre-defined order, the easiest method is to ensure the file names reflect the order of execution. For example, if you have three script files, ScriptA.sql, ScriptB.sql and ScriptC.sql, you would want to name them as

1.ScriptA.sql  
2.ScriptB.sql and  
3.ScriptC.sql

The code shown below also assumes that the source scripts are located in a local folder of the target server. This location is specified at the beginning of the initialisation section. This can be changed to reflect a shared directory in the network; however, the user account running the script needs to have access to that shared folder.

The output of the scripts are captured in files and saved in another location in the server. This can be useful for developers trying to debug a failed installation. These are the same output you would have seen in the “Results” pane of the SSMS if you executed them manually. Like the source directory, the output folder also does not have to be local to the server.

The code makes use of both xp\_cmdshell and the sqlcmd utility. If xp\_cmdshell is turned off in the target server, it needs to be enabled temporarily for the script to work.

The sqlcmd utility was first shipped with SQL Server 2005 and so unavailable in prior versions. When modifying the script for SQL Server 2000 based systems, you can make use of the osql utility.

**Code Walkthrough**

The script is divided into two sections: an initialisation section where relevant metadata is assigned to data structures and an execution section where the source code snippets are run against one or more databases.

1. After setting source and destination folders and the target server name, the code checks if it is actually running against the intended server. This prevents scripts accidentally running in the wrong server.

2. A list of non-system databases is then compiled. This can be modified to include any number of databases.

3. The list of scripts is then compiled from the source folder along with the order of their execution. Note the use of the DOS DIR command along with the usage of xp\_cmdshell. This is because the script is branching out to the OS level for finding the list of files.

4. Once the initialisation section is completed, the code enters a double loop. The outer loop goes through the list of databases identified in step 2.  
  
5. For each iteration of the database loop, the inner loop executes the source scripts according to their order. It makes use of the sqlcmd utility in doing so.  
  
6. Script outputs are saved as disk files. The output file names are dynamically generated from the current database names.

To make the code more efficient, I have tried to make use of table variables instead of temporary tables. Also, cursors are not used when looping is required.

The connection to the database server can be either trusted or non-trusted. The trusted switch (-E) in the sqlcmd can be replaced with the –Uusername and –Ppassword parameters.

**Script Code**

Okay, here is the actual script:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Script Name: scr\_Run\_Multiple\_Scripts\_AgainstDBs.sql

Author: Sadequl Hussain (C) Sadequl Hussain 2009

Purpose: This script is used to run a series of scripts in a defined order against multiple databases

in a target server.

Assumptions / Requirements:

a) The SQL Server instance where the script is running should have xp\_cmdshell enabled

b) The call to "sqlcmd" is valid for SQL 2005. For SQL 2000, "osql" can be used

c) The SQLCMD / OSQL command to the server can use trusted connection switch or use username & password

d) The script names are prefixed with the order of their execution.

e.g. 1.Script.sql is supposed to be executed first

e) The reference to sys.databases can be replaced with master..sysdatabases for SQL 2000 based systems.

However, for that to work, the field name database\_id needs to be changed to dbid

Notes: This can be turned into an effective DBA stored procedure

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SET NOCOUNT ON

DECLARE @ServerName sysname

DECLARE @DBName sysname

DECLARE @SourceFolder nvarchar(500)

DECLARE @OutputFolder nvarchar(500)

DECLARE @ScriptName nvarchar(500)

DECLARE @ScriptFullPathName nvarchar(500)

DECLARE @Cmd nvarchar(1000)

DECLARE @ScriptList table

(

ScriptRunOrder tinyint NOT NULL,

ScriptName nvarchar (500)NOT NULL

)

DECLARE @ScriptListTemp table

(

ScriptName nvarchar (500)NULL

)

DECLARE @DatabaseList table

(

DatabaseID int,

DBName sysname

)

DECLARE @Pos tinyint

DECLARE @Order tinyint

DECLARE @ScriptRunOrder tinyint

DECLARE @Message nvarchar(1000)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

-- Initialisation...

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SELECT @ServerName = '<target\_server\_name>' -- change it to reflect target server name...

SELECT @SourceFolder = 'C:\Scripts' -- change it to reflect script source location...

SELECT @OutputFolder = 'C:\Script\_Output' -- change it to reflect script output location...

SELECT @Cmd = 'master..xp\_cmdshell ''DIR ' + @SourceFolder + ' /B'''

IF ((SELECT @@SERVERNAME) <> @ServerName)

BEGIN

RAISERROR ('Sorry, wrong server !!!', 17, 1)

RETURN

END

INSERT INTO @DatabaseList (DatabaseID, DBName)

SELECT database\_id, name

FROM sys.databases

WHERE name NOTIN ('master','model', 'msdb','tempdb') -- this can be changed to include one or more databases

ORDERBY database\_id ASC

INSERT INTO @ScriptListTemp EXEC(@Cmd)

DELETE FROM @ScriptListTemp WHERE ScriptName IS NULL

IF NOT EXISTS (SELECT\* FROM @ScriptListTemp)

BEGIN

RAISERROR ('Sorry, no files present in the source folder !!!', 17, 1) WITH NOWAIT

RETURN

END

SELECT TOP 1 @ScriptName = ScriptName FROM @ScriptListTemp ORDER BY ScriptName ASC

WHILE EXISTS(SELECT \* FROM @ScriptListTemp)

BEGIN

SELECT @Pos =CHARINDEX('.', @ScriptName)

SELECT @Order =CONVERT(tinyint,LEFT(@ScriptName, @Pos -1))

SELECT @ScriptFullPathName = @SourceFolder + '\'+ @ScriptName

INSERT INTO @ScriptList (ScriptRunOrder, ScriptName)VALUES (@Order, @ScriptFullPathName)

DELETE FROM @ScriptListTemp WHERE ScriptName = @ScriptName

SELECT TOP 1 @ScriptName = ScriptName FROM @ScriptListTemp ORDER BY ScriptName ASC

END

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

-- Execution of scripts...

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SELECT TOP 1 @DBName = DBName FROM @DatabaseList ORDER BY DatabaseID ASC

WHILE EXISTS (SELECT \*FROM @DatabaseList)

BEGIN

SELECT @Message = 'Database - '+ @DBName

RAISERROR (@Message, 10,1) WITHNOWAIT

SELECT @Message = '====================================================================='

RAISERROR (@Message, 10,1) WITHNOWAIT

SELECT TOP 1 @ScriptName = ScriptName, @Order = ScriptRunOrder FROM @ScriptList ORDER BY ScriptRunOrder ASC

WHILEEXISTS (SELECT ScriptRunOrder FROM @ScriptList WHERE ScriptRunOrder = @Order)

BEGIN

SELECT @ScriptRunOrder = @Order

SELECT @Message = 'Now processing script file ' + @ScriptName

RAISERROR (@Message, 10,1) WITHNOWAIT

SET @Cmd ='sqlcmd -E -S ' + @ServerName +' -d ' + @DBName +' -i "' + @ScriptName +'" -o "' + @OutputFolder + '\'+ @DBName + '\_ScriptRun'+ CONVERT(nvarchar(5),@Order)+ '\_Results.txt"'

EXEC master..xp\_cmdshell @Cmd, no\_output

SELECT TOP 1 @ScriptName = ScriptName, @Order = ScriptRunOrder FROM @ScriptList WHERE ScriptRunOrder > @Order ORDER BY ScriptRunOrder ASC

IF (@Order = @ScriptRunOrder) BREAK

END

PRINT''

DELETE FROM @DatabaseList WHERE DBName = @DBName

SELECT TOP 1 @DBName = DBName FROM @DatabaseList ORDER BY DatabaseID ASC

END

RETURN

**Limitations**

Note that the automation script runs here as the “caller” of other scripts and executes them synchronously. What this means is that it will wait for a source script to finish running before invoking the next. To give an idea about the status, it prints out messages about the current database context and the currently executing script.

If any of the source scripts throws an error at runtime, it will not be trapped because the calling script is unaware of the logic inside. So if you have a situation where a script’s failure should prevent subsequent files from running - that cannot be achieved here. However, the output files can be used for assessing the success or failure of a rollout.

**How it Works**

To give a demonstration of the automation, I have created three extremely simple scripts files:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Script Name: 1.scr\_Script1.sql

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SET NOCOUNT ON

SELECT SERVERPROPERTY('ProductVersion')

SELECT DB\_NAME()

SET NOCOUNT OFF

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Script Name: 2.scr\_Script2.sql

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

SET NOCOUNT ON

SELECT GETDATE()

SELECT @@VERSION

SET NOCOUNT OFF

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Script Name: 4.scr\_Script3.sql

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

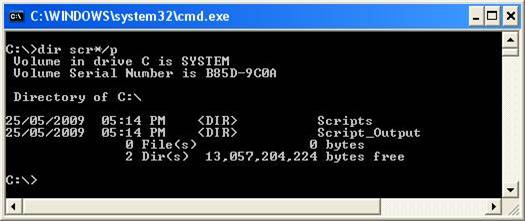
SET NOCOUNT ON

SELECT SUSER\_SNAME()

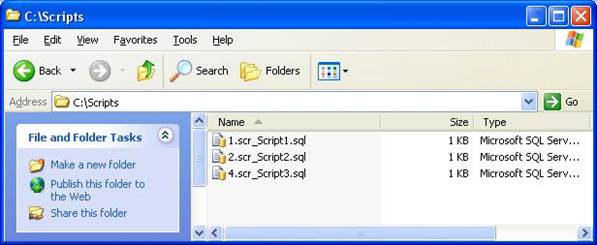
SET NOCOUNT OFF

Note that the first character in each of the script file name is a number, indicating its order of execution. So in this case 1.scr\_Script1.sql is supposed to run first, followed by 2.scr\_Script2.sql and 4.scr\_Script3.sql.

I have also created two folders called “Scripts” and “Script\_Output” under the C:\ drive of my local computer where SQL Server is running. The script files are copied under C:\Scripts.

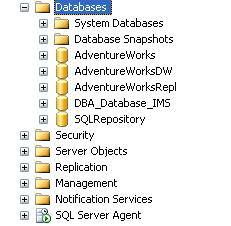


http://www.sqlservercentral.com/Resources/Images/zoom.gif[Zoom in](javascript:;)  |  [Open in new window](javascript:;)

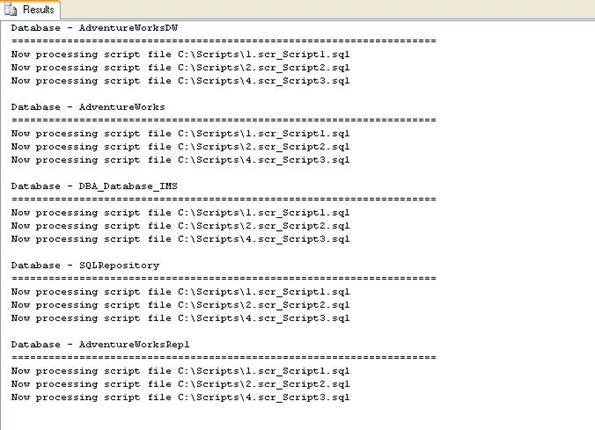


http://www.sqlservercentral.com/Resources/Images/zoom.gif[Zoom in](javascript:;)  |  [Open in new window](javascript:;)

There are five databases in my local copy of SQL Server:

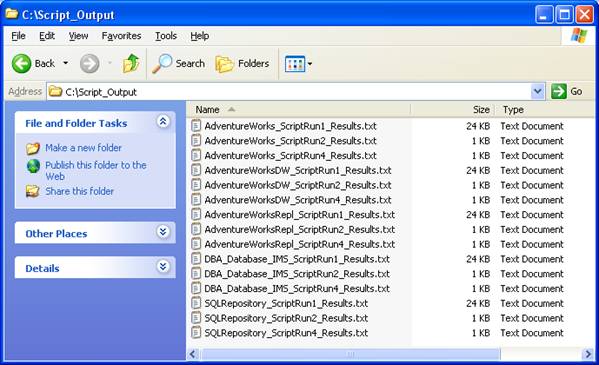


To run the three scripts against these databases in the order specified, I start a new session against the SQL Server from the Management Studio and load the master script. When I hit the Execute button, the Results pane shows me the following:



http://www.sqlservercentral.com/Resources/Images/zoom.gif[Zoom in](javascript:;)  |  [Open in new window](javascript:;)

The script output folder now looks like the following:



http://www.sqlservercentral.com/Resources/Images/zoom.gif[Zoom in](javascript:;)  |  [Open in new window](javascript:;)

I can now browse through these file to see the output generated.