**Space Capacity Automation**

|  |  |
| --- | --- |
| **Index** | |
| **Procedure Parameter** | **Short Description** |
| [General Info](#Get_Info) | Describes usage of this procedure |
| [@help](#Get_Help) | Find Help |
| [@getInfo](#Analyze_DataFiles_Distribution) | Analyze Data Files Distribution |
| [@getLogInfo](#Analyze_LogFiles_Distribution) | Analyze Log Files Distribution |
| [@addDataFiles](#Add_Data_Files_on_New_Volume) | Add Data Files on New Volume |
| [@restrictDataFileGrowth](#Restrict_Data_Files) | Restrict Data Files on Old Volume |
| [@addLogFiles](#Add_Log_Files_on_New_Volume) | Add Log Files on New Volume |
| [@restrictLogFileGrowth](#Restrict_Log_Files) | Restrict Log Files on Old Volume |
| [@generateCapacityException](#Generate_Capacity_Exception) | Generate Capacity Exception for Old Volume |
| [@unrestrictFileGrowth](#Remove_Growth_Restriction) | Remove growth restriction for files on Old Volume |
| @removeCapacityException | Generate TSQL code for Removing Already added Capacity Exception |
| @UpdateMountPointSecurity | Help for providing access of new volume for SQL Service Account |
| @restrictMountPointGrowth | Resize the data files to better utilize the Data or Log Volume |
| [@expandTempDBSize](#Resize_Tempdb_Files) | Resize tempdb data files for both Ideal and Non-ideal Scenarios |
|  |  |

General Info]

This automation has been designed to eliminate manual efforts on Space Capacity ESC tickets where DBA has to add new data or log files on new volume, and restrict data or log files on old volume. Latest code of procedure can be found at path [\sqlserver\scripts\sql\Maintenance](file:///\\gdv01fil01\d101\dba\sqlserver\scripts\sql\Maintenance).

For example, say, on server dbTest1774, a new data volume ‘E:\Data1\’ has been added. So, DBA has to add new data files on @newVolume (E:\Data1\) and restrict data files on @oldVolume (E:\Data\). This can be accomplished by below methods:-

*EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @addDataFiles = 1 ,@newVolume = 'E:\Data1\' ,@oldVolume = 'E:\Data\';*

This generates TSQL code for adding data files on @newVolume for data files present on @oldVolume for each combination of database and filegroup.

In case, we don’t want TSQL code generation, rather wish to execute it right away, we can execute procedure with *@forceExecute* parameter.

*EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @addDataFiles = 1 ,@newVolume = 'E:\Data1\' ,@oldVolume = 'E:\Data\' ,@forceExecute = 1;*

Similarly the procedure [dbo].[usp\_AnalyzeSpaceCapacity] can be used for multiple activities related to space capacity.

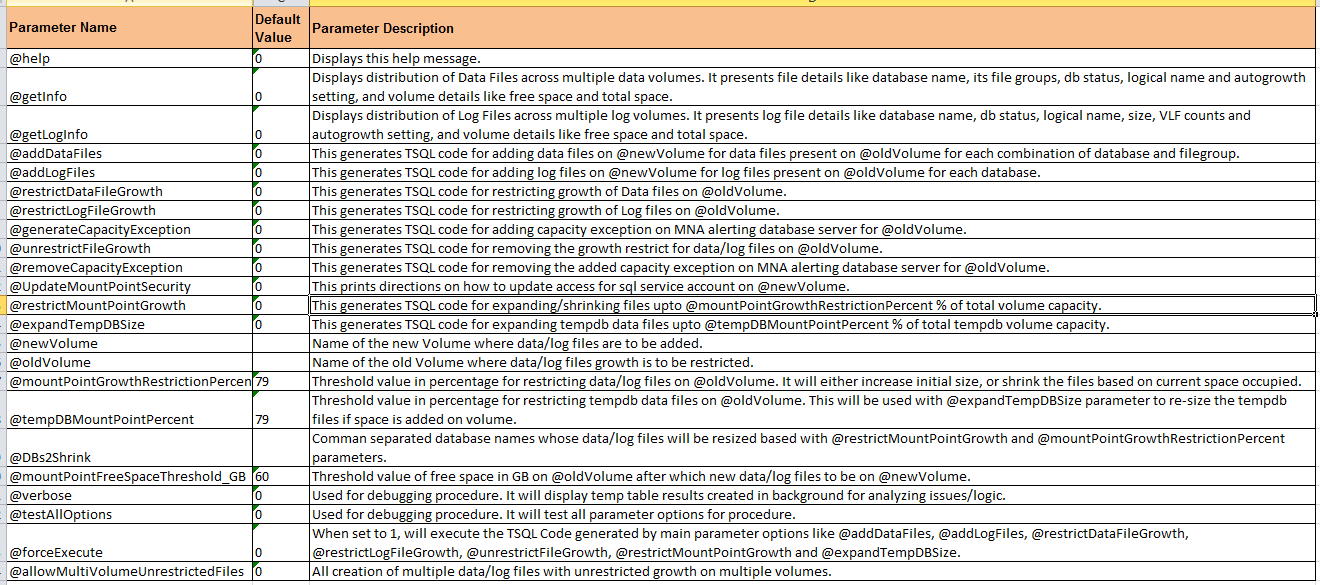
[Goto Index](#Index)

Find Help (@help)

The parameter provides directions on how to use this procedure. It also presents 12 examples in it.

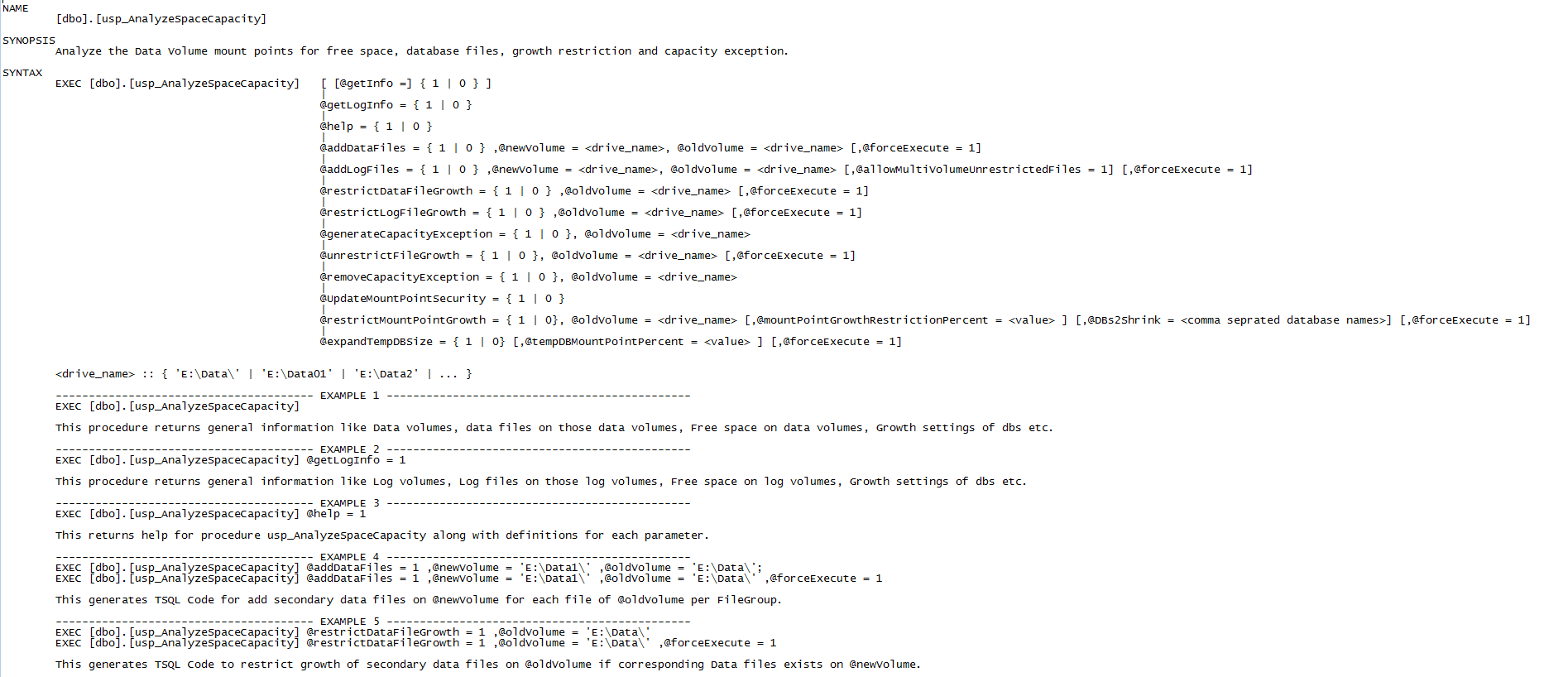
*exec dbo.[usp\_AnalyzeSpaceCapacity] @help = 1*

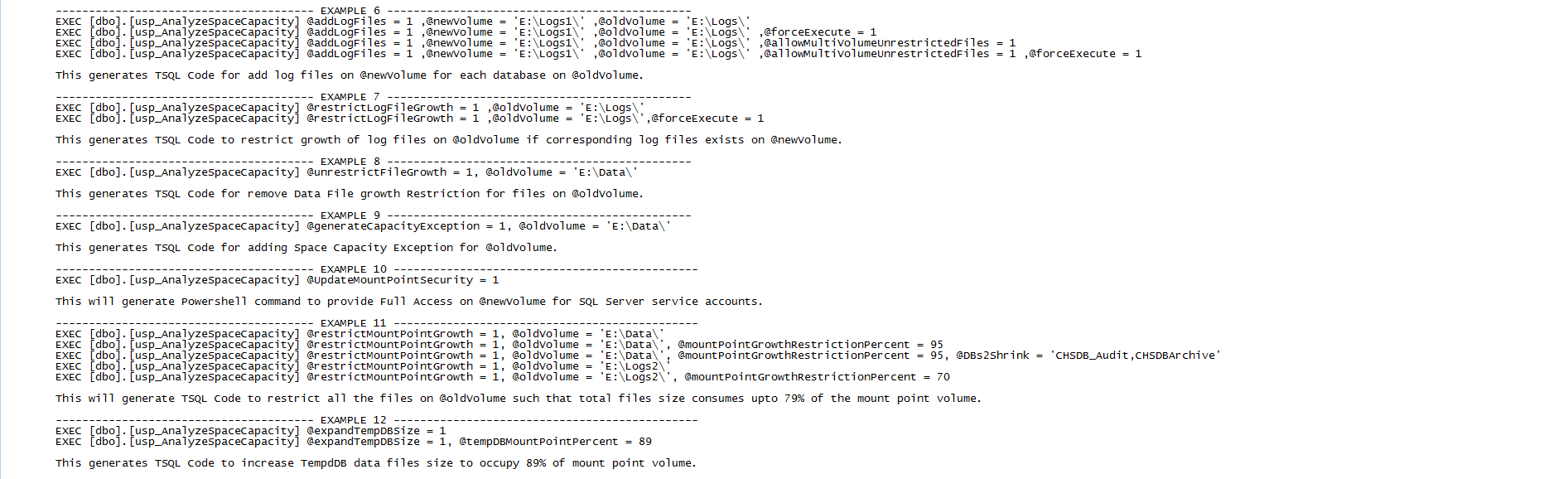
Also, below are the parameters for procedure with default values:-



For best result, always take out help from procedure using **@help** parameter.

And, below are the examples





[Goto Index](#Index)

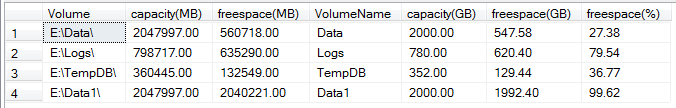
Analyze Data Files Distribution (@getInfo)

This parameter is used to display distribution of Data Files across multiple data volumes. It presents file details like database name, its file groups, db status, logical name and auto growth setting, and volume details like free space and total space. Check the sample output in below excel:-



**Logical Flow**:-

1. Store space analysis for all volumes into @mountPointVolumes variable table.



1. Find out all data files for each database, and store the data in intermediate table with one record for each combination of database, filegroup and Data volume. Then combine the result of space analysis in same line using dynamic pivoting. Check the below excel for intermediate table before final pivoted result.



[Goto Index](#Index)

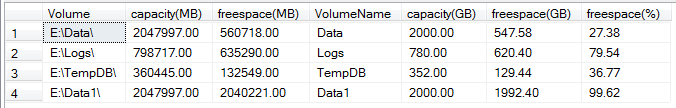
Analyze Log Files Distribution (@getLogInfo)

This parameter is used to display distribution of Log Files across multiple log volumes. It presents file details like database name, db status, logical name, size, auto growth setting and VLF counts, and volume details like free space and total space. Check the sample output in below excel:-



**Logical Flow**:-

1. Store space analysis for all volumes into @mountPointVolumes variable table.



1. Find out all log files for each database, and store the data in intermediate table with one record for each combination of database and Log volume. Then combine the result of space analysis in same line using dynamic pivoting. Check the below excel for intermediate table before final pivoted result.



[Goto Index](#Index)

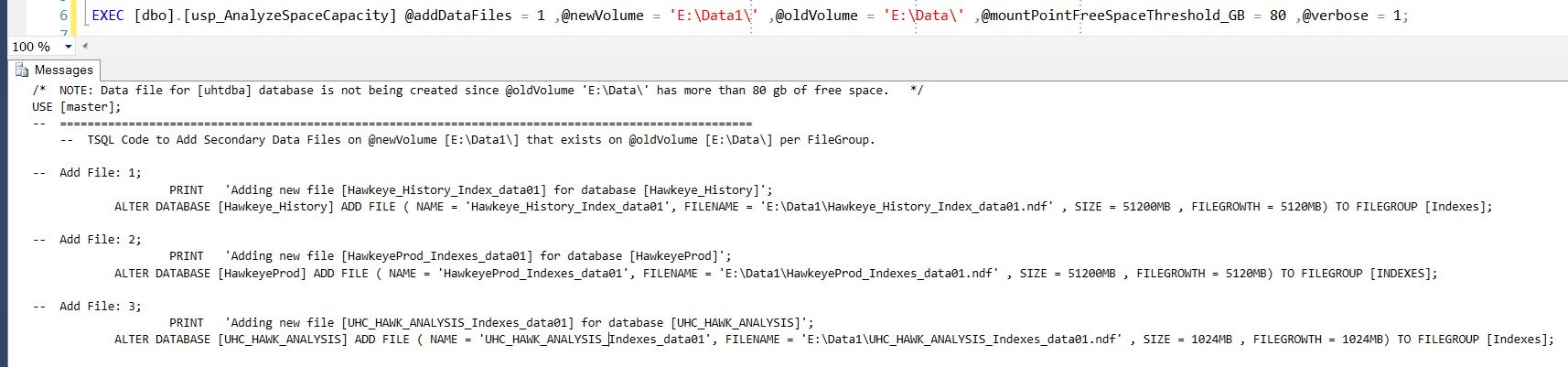
Add Data Files on New Volume (@addDataFiles)

This generates TSQL Code for add secondary data files on @newVolume for each file of @oldVolume per FileGroup.

EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @addDataFiles = 1 ,@newVolume = 'E:\Data1\' ,@oldVolume = 'E:\Data\';

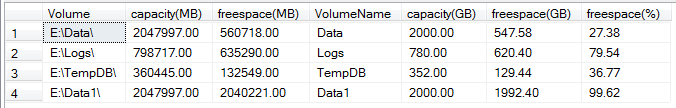
EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @addDataFiles = 1 ,@newVolume = 'E:\Data1\' ,@oldVolume = 'E:\Data\' ,@forceExecute = 1;

Output:-



**Logical Flow**:-

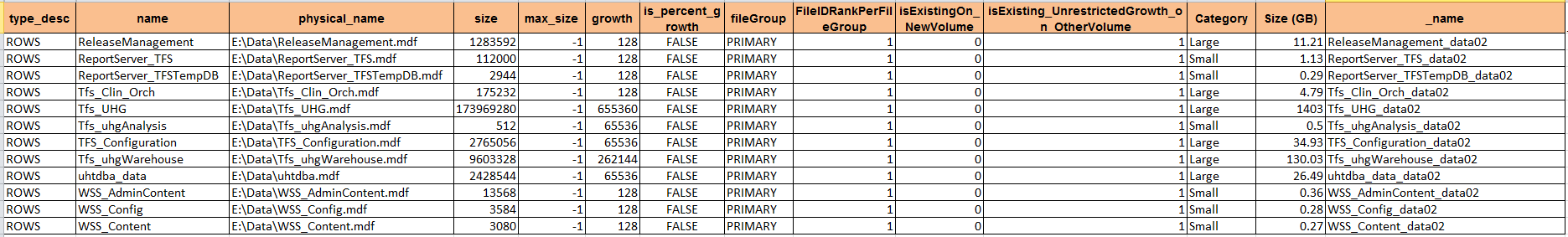
1. Store space analysis for all volumes into @mountPointVolumes variable table.



1. Find out all data files on @oldVolume with various details like



* 1. **FileIDRankPerFileGroup** = Auto generated number starting from 1 for files of @oldVolume for same combination of database and filegroup



* 1. **isExistingOn\_NewVolume** = 1 if there is already another data file existing on @newVolume for same combination of database and filegroup, else 0.
  2. **isExisting\_UnrestrictedGrowth\_on\_OtherVolume** = 1 if there is already another data file existing on other volumes (except @oldVolume) with un-restricted growth, else 0.
  3. **\_name** = logical name for new data file by observing existing naming pattern. If no naming pattern found, then it would be like *databaseName\_data01*.
  4. **\_physical\_name** = path of @newVolume + **\_name** (from above point)
  5. **[Size (GB)]** = database size in GB
  6. **maxfileSize\_oldVolumes\_MB** = Max used space in MB by any data file for particular database and filegroup
  7. **\_initialSize** = case when **[Size (GB)]** < 2 then ‘256MB’

case when **[Size (GB)]** between 2 and 10 then ‘512MB’

case when **[Size (GB)]** > 10 and **maxfileSize\_oldVolumes\_MB** less than 50 gb then ‘1024MB’

case when **maxfileSize\_oldVolumes\_MB** between 50 gb and 200 gb then ‘10240MB’

case when **maxfileSize\_oldVolumes\_MB** > 200 gb then ‘51200MB’

* 1. **\_autoGrowth** = case when **[Size (GB)]** < 2 then ‘256MB’

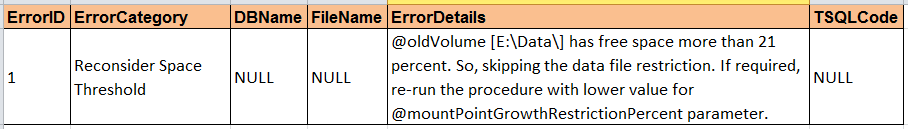
case when **[Size (GB)]** between 2 and 10 then ‘512MB’

case when **[Size (GB)]** > 10 and **maxfileSize\_oldVolumes\_MB** less than 50 gb then ‘1024MB’

case when **maxfileSize\_oldVolumes\_MB** between 50 gb and 200 gb then ‘2048MB’

case when **maxfileSize\_oldVolumes\_MB** > 200 gb then ‘5120MB’

1. Generate TSQL Code to add new data files for @newVolume, or to remove growth restriction
   1. Generate code for adding file if ***isExistingOn\_NewVolume = 0*** AND ***isExisting\_UnrestrictedGrowth\_on\_OtherVolume = 0*** AND **[FileIDRankPerFileGroup] = 1**
   2. Generate code for un-restricting growth of files @newVolume if **isExistingOn\_NewVolume = 1** AND **isExisting\_UnrestrictedGrowth\_on\_OtherVolume = 0**
   3. Do not generate code for [uhtdba] files if **[freespace(GB)]** for @oldVolume **>= @mountPointFreeSpaceThreshold\_GB**.
2. If @forceExecute = 1, then execute generated TSQL code from above step. Else print all the TSQL code to console (messages tab of Results Pane).
3. If some error occurred, the procedure will return 1 otherwise 0 in case of successful execution. Also, an error messages table will be returned in case of error. So, it may happen that TSQL code is generated for 12 databases, and only 8 got successfully executed. For error messages tab will return all exact error with TSQL code for failed 4 databases. Some sample cases could be error for offline databases, read only databases, or database in mirror role.



[Goto Index](#Index)

Add Log Files on New Volume (@addLogFiles)

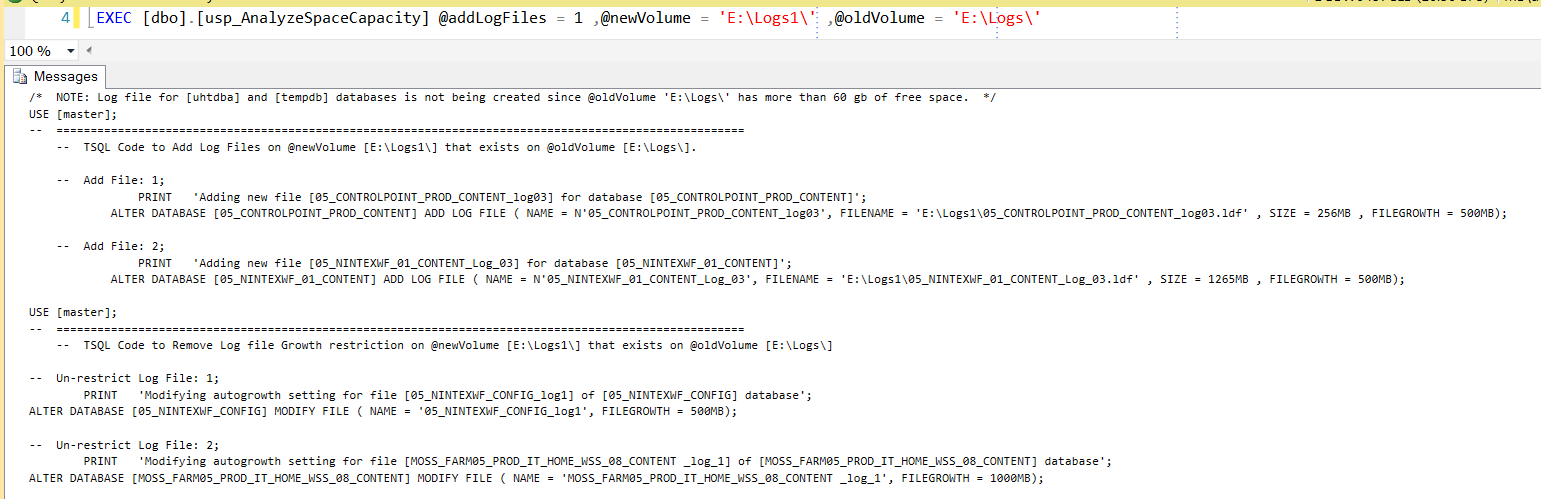
This generates TSQL Code for add log files on @newVolume for each log file of @oldVolume.

EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @addLogFiles = 1 ,@newVolume = 'E:\Data1\' ,@oldVolume = 'E:\Data\';

EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @addLogFiles = 1 ,@newVolume = 'E:\Data1\' ,@oldVolume = 'E:\Data\' ,@forceExecute = 1;

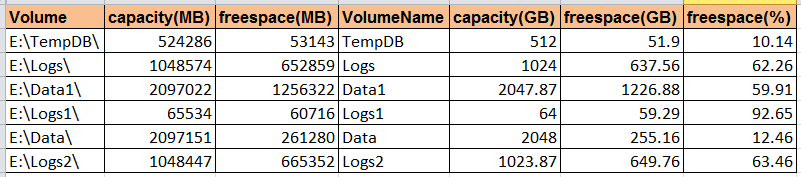
EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @addLogFiles = 1 ,@newVolume = 'E:\Data1\' ,@oldVolume = 'E:\Data\' ,@allowMultiVolumeUnrestrictedFiles = 1;

Output:-



**Logical Flow**:-

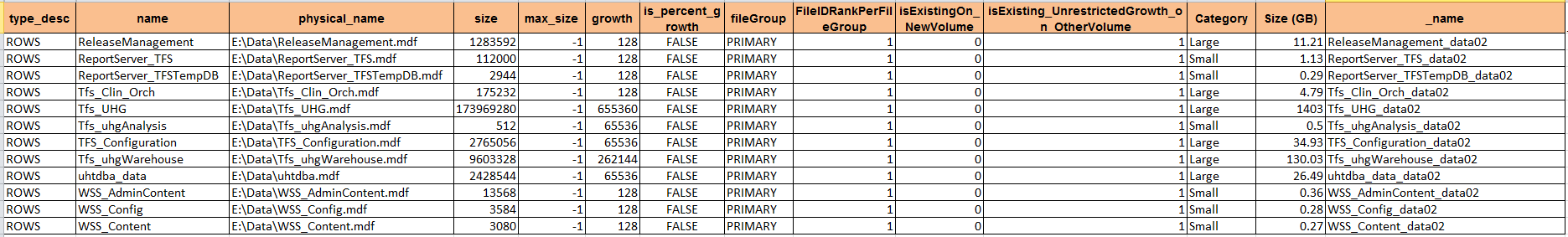
1. Store space analysis for all volumes into @mountPointVolumes variable table.



1. Find out all log files on @oldVolume with various details like



* 1. **FileIDRankPerFileGroup** = Auto generated number starting from 1 for files of @oldVolume for same database



* 1. **isExistingOn\_NewVolume** = 1 if there is already another log file existing on @newVolume for same database, else 0.
  2. **isExisting\_UnrestrictedGrowth\_on\_OtherVolume** = 1 if there is already another log file existing on other volumes (except @oldVolume) with un-restricted growth, else 0.
  3. **\_name** = logical name for new file by observing existing naming pattern. If no naming pattern found, then it would be like *databaseName\_log01*.
  4. **\_physical\_name** = path of @newVolume + **\_name** (from above point)
  5. **[Size (GB)]** = database size in GB
  6. **maxfileSize\_oldVolumes\_MB** = Max current size in MB among log files for particular database
  7. **TotalSize\_All\_DataFiles\_MB** = Sum(Used Space of all data files) for particular database
  8. **TotalSize\_All\_LogFiles\_MB** = Sum(Current size of all Log files) for particular database
  9. **New\_Max\_Log\_Size\_Threadhold** = size in MB based on below logic

*Case when* ***maxfileSize\_oldVolumes\_MB*** *>= 16000 then* ***maxfileSize\_oldVolumes\_MB***

*Case when quarter of* ***TotalSize\_All\_DataFiles\_MB*** *<* ***maxfileSize\_oldVolumes\_MB*** *then* ***maxfileSize\_oldVolumes\_MB***

*Case when (quarter of* ***TotalSize\_All\_DataFiles\_MB*** *-* **TotalSize\_All\_LogFiles\_MB)** *<=* **maxfileSize\_oldVolumes\_MB** *then* **maxfileSize\_oldVolumes\_MB**

*Else quarter of* ***TotalSize\_All\_DataFiles\_MB*** *-* **TotalSize\_All\_LogFiles\_MB**

* 1. **\_initialSize** = size in MB based on below logic

case when **New\_Max\_Log\_Size\_Threadhold** < 256 then ‘256 MB’

case when **New\_Max\_Log\_Size\_Threadhold** < 1000 then **New\_Max\_Log\_Size\_Threadhold** MB

case when **New\_Max\_Log\_Size\_Threadhold** = 8192 then ‘4000 MB’

case when **New\_Max\_Log\_Size\_Threadhold** < 16000 then half of **New\_Max\_Log\_Size\_Threadhold MB**

else ‘8000 MB’

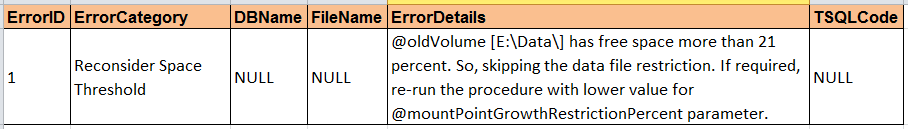
* 1. **\_autoGrowth** = size in MB based on below logic

case when **New\_Max\_Log\_Size\_Threadhold** < 8000 then ‘500 MB’

case when **New\_Max\_Log\_Size\_Threadhold** < 16000 then ‘1000 MB’

else ‘8000 MB’

1. Generate TSQL Code to add new data files for @newVolume, or to remove growth restriction
   1. Generate code for adding file if ***isExistingOn\_NewVolume = 0*** AND ***isExisting\_UnrestrictedGrowth\_on\_OtherVolume = 0*** AND **[FileIDRankPerFileGroup] = 1**
   2. Generate code for un-restricting growth of files @newVolume if **isExistingOn\_NewVolume = 1** AND **isExisting\_UnrestrictedGrowth\_on\_OtherVolume = 0**
   3. Do not generate code for [uhtdba] files if **[freespace(GB)]** for @oldVolume >= **@mountPointFreeSpaceThreshold\_GB**.
   4. Ignore value of **isExisting\_UnrestrictedGrowth\_on\_OtherVolume** for above steps if **@allowMultiVolumeUnrestrictedFiles** is set to 1
2. If @forceExecute = 1, then execute generated TSQL code from above step. Else print all the TSQL code to console (messages tab of Results Pane).
3. If some error occurred, the procedure will return 1 otherwise 0 in case of successful execution. Also, an error messages table will be returned in case of error. So, it may happen that TSQL code is generated for 12 databases, and only 8 got successfully executed. For error messages tab will return all exact error with TSQL code for failed 4 databases. Some sample cases could be error for offline databases, read only databases, or database in mirror role.



[Goto Index](#Index)

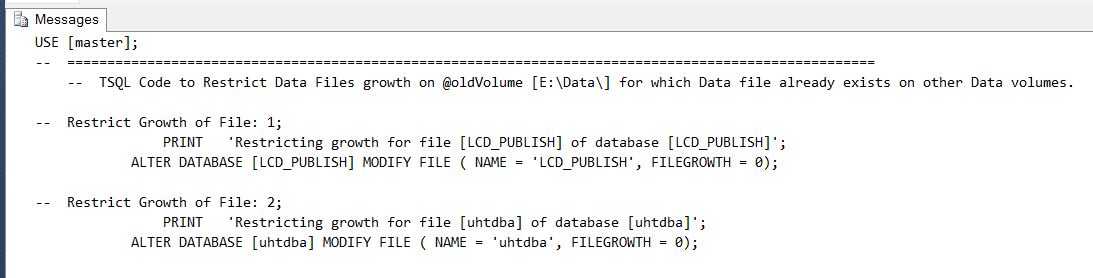
Restrict Data Files (@restrictDataFileGrowth)

This generates TSQL Code to restrict growth of secondary data files on @oldVolume if alternate data files with un-restricted growth are present on other volumes.

EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @restrictDataFileGrowth = 1 ,@oldVolume = 'E:\Data\';

EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @restrictDataFileGrowth = 1 ,@oldVolume = 'E:\Data\' ,@forceExecute = 1;

Output:-



**Logical Flow**:-

1. Find out all data files on @oldVolume with various details like



1. **isExisting\_UnrestrictedGrowth\_on\_OtherVolume** = 1 if there is already another data file existing on other volumes (except @oldVolume) with un-restricted growth for same database and filegroup, else 0.
2. Generate TSQL Code to restrict data files for @oldVolume using below steps
   1. Throw an error for all databases & file groups for which new data files are yet to be created if **isExisting\_UnrestrictedGrowth\_on\_OtherVolume** is 0
   2. Generate code to restrict growth of file if

**isExisting\_UnrestrictedGrowth\_on\_OtherVolume** is 1

1. If @forceExecute = 1, then execute generated TSQL code from above step. Else print all the TSQL code to console (messages tab of Results Pane).

[Goto Index](#Index)

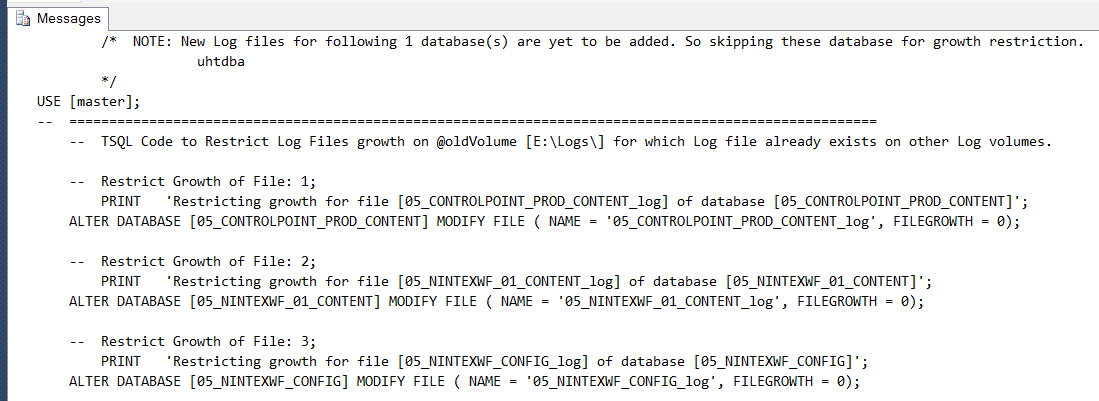
Restrict Log Files (@restrictLogFileGrowth)

This generates TSQL Code to restrict growth of log files on @oldVolume alternate log files with un-restricted growth are present on other volumes.

EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @restrictLogFileGrowth= 1 ,@oldVolume = 'E:\Logs\';

EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @restrictLogFileGrowth = 1 ,@oldVolume = 'E:\Logs\' ,@forceExecute = 1;

Output:-



**Logical Flow**:-

1. Find out all log files on @oldVolume with various details like



1. **isExisting\_UnrestrictedGrowth\_on\_OtherVolume** = 1 if there is already another log file existing on other volumes with un-restricted growth for same database, else 0.
2. Generate TSQL Code to restrict log files of @oldVolume using below steps
   1. Throw an error for all databases & file groups for which new data files are yet to be created if **isExisting\_UnrestrictedGrowth\_on\_OtherVolume** is 0
   2. Generate code to restrict growth if **isExisting\_UnrestrictedGrowth\_on\_OtherVolume** is 1
3. If @forceExecute = 1, then execute generated TSQL code from above step. Else print all the TSQL code to console (messages tab of Results Pane).

[Goto Index](#Index)

Generate Capacity Exception (@generateCapacityException)

This generates TSQL code for adding capacity exception on MNA alerting database server for @oldVolume.

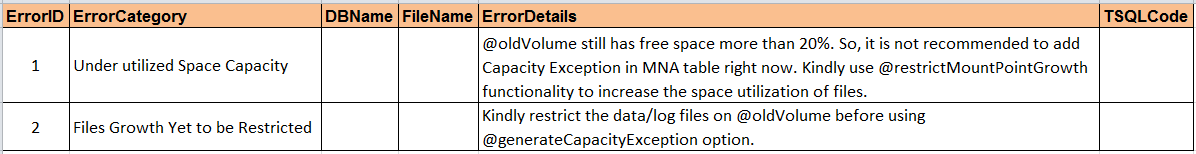
EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @generateCapacityException = 1, @oldVolume = 'E:\Data\';

EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @generateCapacityException = 1, @oldVolume = 'E:\Logs\';

Output:-

**Logical Flow**:-

1. Add warning message into **#ErrorMessages** table if free space on @oldVolume is more than 20%
2. Throw an error if there are data/log files with auto growth enabled. So skip Capacity Exception Generation.



1. Generate Capacity Exception using below values:-
   1. **@pWarningThreshold** = Current Used space on @oldVolume
   2. **@pCriticalThreshold** = **@pWarningThreshold** + 2

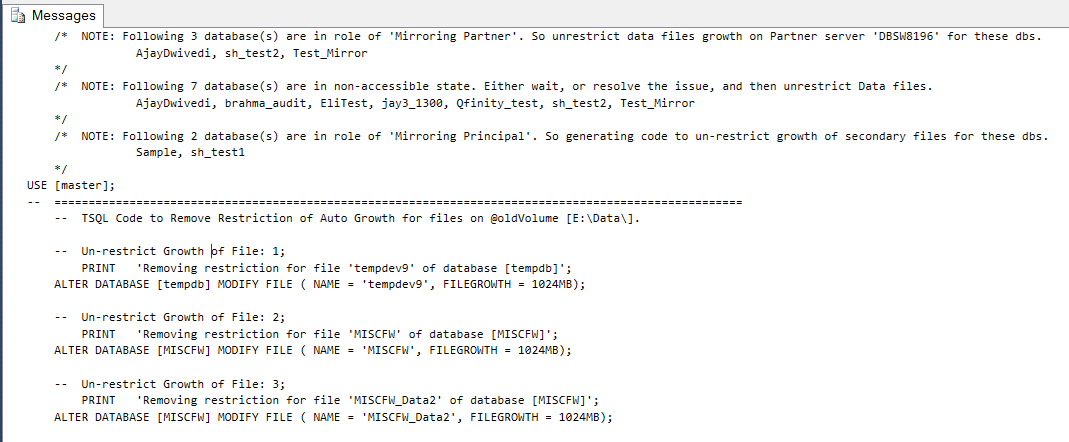
[Goto Index](#Index)

Remove growth restriction (@unrestrictFileGrowth)

This generates TSQL code for removing the growth restriction for data/log files on @oldVolume.

EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @unrestrictFileGrowth = 1, @oldVolume = 'E:\Data\';

Output:-



**Logical Flow**:-

1. Find all data/log files created on @oldVolume with below details:-



1. **\_autoGrowth** = Same values as for option [@addDataFiles](#Add_Data_Files_on_New_Volume)[/@addLogFiles](#Add_Log_Files_on_New_Volume).
2. If growth = 0, then generate TSQL code to un-restrict data/log files using autoGrowth setting value of [**\_autoGrowth**].

[Goto Index](#Index)

Resize tempdb files (@expandTempDBSize)

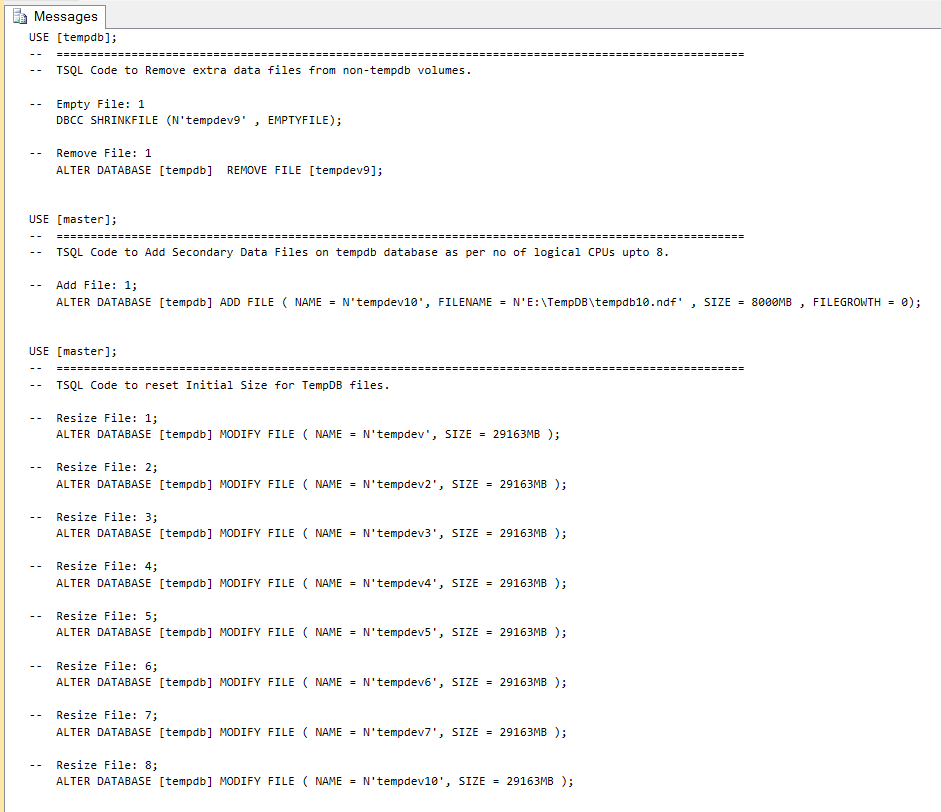
This generates TSQL code for expanding tempdb data files up to **@tempDBMountPointPercent** of total tempdb volume capacity. When @output4IdealScenario set to 1, will generate TSQL code to add/remove data files based on the number Logical cores on server up to 8, and delete extra data files created on non-tempdb volumes.

EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @expandTempDBSize = 1;

EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @expandTempDBSize = 1, @output4IdealScenario = 1;

EXEC [dbo].[usp\_AnalyzeSpaceCapacity] @expandTempDBSize = 1, @tempDBMountPointPercent = 89;

Output:-



**Logical Flow**:-

1. Find out all data files for [tempdb] database with various details like below



1. **isToBeDeleted** = 1 if the volume of the data file is non-tempdb like E:\Data\ etc.., else 0.
2. **@\_logicalCores** = cpu\_count from [sys].[dm\_os\_sys\_info]
3. **@\_counts\_of\_Files\_To\_Be\_Created** = no of extra data files to be created based on current existing file count and **@\_logicalCores**.
4. Create an intermediate temp table with final data files with below details:-



* 1. **isToBeCreated** = 1 if value of **@\_counts\_of\_Files\_To\_Be\_Created** is not zero, and the data files has to be added.
  2. **isExtraFile** = 1 if the file is extra file than the number of **@\_logicalCores**. Selecting latest added files to be deleted.
  3. **Logical Name**, and **Physical Name**
     1. For pre-existing data file, same logical and physical name
     2. For New data file, Logical Name and Physical name generated as per pattern starting from last used integer number in previous files

1. Generate TSQL code with below logic if **@output4IdealScenario** is set to 1:-
   1. If **isToBeDeleted** = 1, then generated TSQL code to empty the data file, and then remove the same data file.
   2. If **isExtraFile** = 1, then generated TSQL code to empty the data file, and then remove the same data file.
   3. If **isToBeCreated** = 1, then generate TSQL code to add the data file.
2. Generate TSQL code for resizing the data files executed, and present after step 3. Check Output screenshot above.

[Goto Index](#Index)