



MINION BACKUP : QUICK START

Minion Backup by MidnightDBA is a stand-alone backup solution that can be deployed on any number of servers, for free. Minion Backup is comprised of SQL Server tables, stored procedures, and SQL Agent jobs. For links to downloads, tutorials, and articles, see www.MinionWare.net.

This document explains Minion Backup by MidnightDBA (“Minion Backup”), its uses, features, moving parts, and examples.



To install Minion Backup:

1. Download **MinionBackup1.0.sql** from www.MinionWare.net.
2. Edit the script to customize the backup drive and path.
3. Run the script on your target server.

For simplicity, this Quick Start guide assumes that you have installed Minion Backup on one server, named “YourServer”.

Note: You can also use the “**MinionMassInstall.ps1**” PowerShell script provided with the Minion Backup download to install Minion Backup on dozens or hundreds of servers at once, just as easily as you would install it on a single instance.

System requirements:

- SQL Server 2008* or above.
- The sp_configure setting **xp_cmdshell** must be enabled**.
- PowerShell 2.0 or above; execution policy set to **RemoteSigned**.

Once MinionBackup1.0.sql has been run, nothing else is required. From here on, Minion Backup will run nightly to back up **all** non-TempDB databases. The backup routine automatically handles databases as they are created, dropped, or renamed.

**There is a special edition of Minion Backup specifically for SQL Server 2005.
But, be aware that this edition will not be enhanced or upgraded,
some functionality is reduced, and it will have limited support.*

*** xp_cmdshell can be turned on and off with the database
PreCode / PostCode options, to help comply with security policies.*

Change Schedules

Minion Backup offers a choice of scheduling options. This quick start section covers the default method of scheduling: table based scheduling. We will cover parameter based schedules, and hybrid schedules, in the section titled "[How To: Change Backup Schedules](#)".

Table based scheduling

In conjunction with the "MinionBackup-AUTO" job, the Minion.BackupSettingsServer table allows you to configure flexible backup scheduling scenarios. By default, Minion Backup comes installed with the following scenario:

- The *MinionBackup-Auto* job runs once every 30 minutes, checking the Minion.BackupSettingsServer table to determine what backup should be run.
- In Minion.BackupSettingsServer:
 - Full system backups are scheduled **daily at 10:00pm**.
 - Full user backups are scheduled on **Saturdays at 11:00pm**.
 - Differential backups for user databases are scheduled **daily except Saturdays (weekdays and on Sunday) at 11:00pm**.
 - Log backups for user databases run daily as **often as the MinionBackup-AUTO job runs (every 30 minutes)**.

The following table displays the first few columns of this default scenario in Minion.BackupSettingsServer:

ID	DBType	BackupType	Day	ReadOnly	BeginTime	EndTime	MaxForTimeframe
1	System	Full	Daily	1	22:00:00	22:30:00	1
2	User	Full	Saturday	1	23:00:00	23:30:00	1
3	User	Diff	Weekday	1	23:00:00	23:30:00	1
4	User	Diff	Sunday	1	23:00:00	23:30:00	1
5	User	Log	Daily	1	00:00:00	23:59:00	48

Let's walk through three different schedule changes.

Scenario 1: Run log backups every 15 minutes, instead of half hourly. To change the default setup in order to run log backups every 15 minutes, change the *MinionBackup-AUTO* job schedule to run once every 15 minutes, and update the BackupType='Log' row in Minion.BackupSettingsServer to increase the "MaxForTimeframe" value to 96 or more (as there will be a maximum of 96 log backups per day).

Scenario 2: Run full backups daily, and no differential backups. To change the default setup in order to run daily full backups and eliminate differential backups altogether:

1. Update the DBType='User', BackupType='Full' row in Minion.BackupSettingsServer, setting the Day field to "Daily".
2. Update the BackupType='Diff' rows in Minion.BackupSettingsServer, setting the isActive fields to **0**.

Scenario 3: Run differential backups twice daily. To change the default setup in order to differential backups twice daily, insert two new rows to Minion.BackupSettingsServer for BackupType='Diff', one for weekdays and one for Sundays, as follows:

```
INSERT INTO Minion.BackupSettingsServer
( [DBType],
  [BackupType] ,
  [Day] ,
  [ReadOnly] ,
  [BeginTime] ,
  [EndTime] ,
  [MaxForTimeframe] ,
  [SyncSettings] ,
  [SyncLogs] ,
  [IsActive] ,
  [Comment]
)
SELECT 'User' AS DBType,
       'Diff' AS [BackupType] ,
       'Weekday' AS [Day] ,
       1 AS [ReadOnly] ,
       '06:00:00' AS [BeginTime] ,
       '07:00:00' AS [EndTime] ,
       1 AS [MaxForTimeframe] ,
       0 AS [SyncSettings] ,
       0 AS [SyncLogs] ,
       1 AS [IsActive] ,
       'Weekday morning differentials' AS [Comment];
```

```
INSERT INTO Minion.BackupSettingsServer
( [DBType],
  [BackupType] ,
  [Day] ,
  [ReadOnly] ,
  [BeginTime] ,
  [EndTime] ,
  [MaxForTimeframe] ,
```

```

[SyncSettings] ,
[SyncLogs] ,
[IsActive] ,
[Comment]
)
SELECT 'User' AS DBType,
'Diff' AS [BackupType] ,
'Sunday' AS [Day] ,
1 AS [ReadOnly] ,
'06:00:00' AS [BeginTime] ,
'07:00:00' AS [EndTime] ,
1 AS [MaxForTimeframe] ,
0 AS [SyncSettings] ,
0 AS [SyncLogs] ,
1 AS [IsActive] ,
'Sunday morning differentials' AS [Comment];

```

These will provide a second differential backup at 6:00am on weekdays and Sundays, to supplement the existing differential backup in the evenings. The contents of Minion.BackupSettingsServer will then look (in part) like this:

ID	DBType	BackupType	Day	ReadOnly	BeginTime	EndTime	MaxForTimeframe
1	System	Full	Daily	1	22:00:00	22:30:00	1
2	User	Full	Saturday	1	23:00:00	23:30:00	1
3	User	Diff	Weekday	1	23:00:00	23:30:00	1
4	User	Diff	Sunday	1	23:00:00	23:30:00	1
5	User	Log	Daily	1	00:00:00	23:59:00	48
6	User	Diff	Weekday	1	06:00:00	07:00:00	1
7	User	Diff	Sunday	1	06:00:00	07:00:00	1

Important notes:

- **Always set the MaxForTimeframe field.** This determines how many of the given backup may be taken in the defined timeframe. In the insert statement above, MaxForTimeframe is set to 1, because we only want to allow 1 differential backup operation during the 6:00am hour.
- **The backup job should run as often as your most frequent backup.** For example, if log backups should run every 5 minutes, schedule the job for every 5 minutes. And be *sure* to set the MaxForTimeframe sufficiently high enough to allow all of the log backups. In this case, we take log backups every 5 minutes for each 24 hour period, meaning up to 288 log backups a day; so, we could set MaxForTimeframe = 288, or any number higher (just to be sure).

Change Default Settings

Minion Backup stores default settings for the entire instance in a single row (where DBName='MinionDefault' and BackupType='All') in the Minion.BackupSettings table.

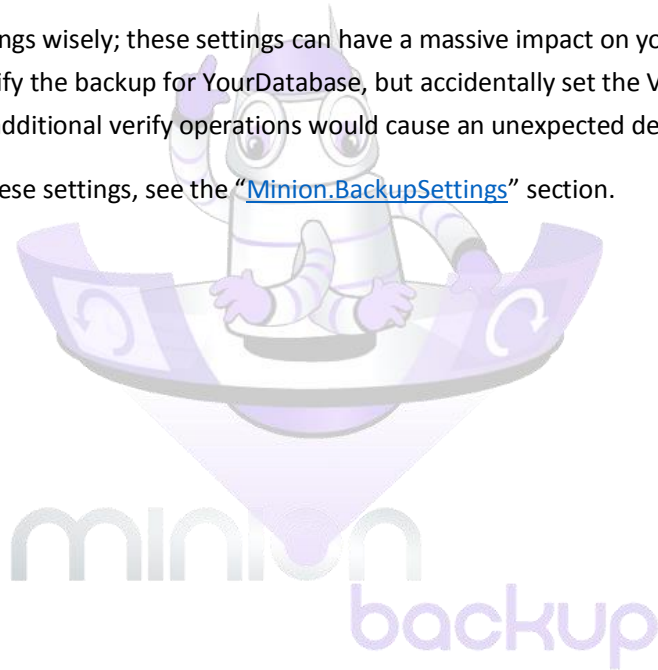
Warning: Do not delete the MinionDefault row, or rename the DBName for the MinionDefault row, in Minion.BackupSettings!

To change the default settings, run an update statement on the MinionDefault row in Minion.BackupSettings. For example:

```
UPDATE Minion.BackupSettings
SET   [Exclude] = 0 ,
      [LogLoc] = 'Local' ,
      [HistRetDays] = 60 ,
      [ShrinkLogOnLogBackup] = 0 ,
      [ShrinkLogThresholdInMB] = 0 ,
      [ShrinkLogSizeInMB] = 0
WHERE [DBName] = 'MinionDefault'
      AND BackupType = 'All';
```

Warning: Choose your settings wisely; these settings can have a massive impact on your backups. For example, if you want to verify the backup for YourDatabase, but accidentally set the Verify option for the default instance, all of the additional verify operations would cause an unexpected delay.

For more information on these settings, see the [“Minion.BackupSettings”](#) section.



MINION BACKUP

Contents in Brief

Quick Start.....	1
Top 20 Features	6
Architecture Overview.....	8
“How To” Topics: Basic Configuration.....	15
“How To” Topics: Backup Mirrors and File Actions.....	33
“How To” Topics: Advanced	47
Moving Parts	66
“About” Topics.....	113
FAQ.....	123
About Us.....	126



Top 20 Features

Minion Backup by MidnightDBA is a stand-alone database backup module. Once installed, Minion Backup automatically backs up all online databases on the SQL Server instance, and will incorporate databases as they are added or removed.

Twenty of the very best features of Minion Backup are, in a nutshell:

1. **Live Insight** – See what Minion Backup is doing every step of the way. You can even see the percent complete for each backup as it runs.
2. **Dynamic Backup Tuning** – Configure thresholds and backup tuning settings. Minion Backup will adjust the tuning settings based on your thresholds! Tuning settings can be configured even down to the time of day for maximum control of your resources.
3. **Stripe, mirror, copy, and/or move backup files** – Minion Backup provides extensive file action functionality, all without additional jobs. You even get to choose which utility performs the operations.
4. **Flexible backup file delete and archive** – Each database and backup type can have an individual backup file retention setting. And, you can mark certain backup files as “Archived”, thus preventing Minion Backup from deleting them.
5. **Shrink log file on backup** – Specify the size threshold for shrinking your log file. Minion Backup logs the before and after log sizes.

6. **Backup certificates** – Back up your server and database certificates with secure, encrypted passwords.
7. **Backup ordering** – Back up databases in exactly the order you need.
8. **Extensive, useful logging** – Use the Minion Backup log for estimating the end of the current backup run, troubleshooting, planning, and reporting. And errors get reported in the log table instead of in text files. There's almost nothing MB doesn't log.
9. **Run "missing" backups only** – Did some of your database backups fail last night? The "missing" keyword allows you to rerun a backup operation, catching those backups that failed in the last run (for that database type and backup type). You can even tell MB to check for missing backup automatically.
10. **HA/DR Aware** – Our new *Data Waiter* feature synchronizes backup settings, backup logs, or both among Availability Group nodes; mirroring partners; log ship targets; or any other SQL Server instance. There are other features that enhance your HA/DR scenarios as well.
11. **Flexible include and exclude** – Backup only the databases you want, using specific database names, LIKE expressions, and even regular expressions.
12. **Run code before or after backups** – This is an extraordinarily flexible feature that allows for nearly infinite configurability.
13. **Integrated help** – Get help on any Minion Backup object without leaving Management Studio. And, use the new CloneSettings procedure to generate template insert statements for any table, based on an example row in the table.
14. **Built-in Verify** – If you choose to verify your backups, set the verify command to run after each backup, or after the entire set of backups.
15. **Single-job operation** – You no longer need multiple jobs to run your backups. MB allows you to configure fairly complex scenarios and manage only a single job.
16. **Encrypt backups** – In SQL Server 2014 and beyond, you can choose to encrypt your backups.
17. **Compatible with Availability Groups** – Minion Backup takes full backup of Availability Group scenarios. You can not only use the preferred AG replica for your backups, but you can also specify specific replicas for each backup type.
18. **Scenario testing**— Dynamic tuning, file delete, and file paths all have facilities for testing your configuration before you rely on it.
19. **Automated operation** – Run the Minion Backup installation scripts, and it just goes. You can even rollout to hundreds of servers almost as easily as you can to a single server.
20. **Granular configuration without extra jobs** – Configure extensive settings at the default, and/or database levels with ease. Say good-bye to managing multiple jobs for specialized scenarios. Most of the time you'll run MB with a single job.

For links to downloads, tutorials and articles, see www.MinionWare.net.



Minion Enterprise Hint

Minion Enterprise (ME) is our enterprise management solution for centralized SQL Server management and alerting. This solution allows your database administrator to manage an enterprise of one, hundreds, or even thousands of SQL Servers from one central location. ME provides not just alerting and reporting, but backups, maintenance, configuration, and enforcement. ME integrates with Minion Backup.

See www.MinionWare.net for more information, or email us today at Support@MidnightDBA.com for a demo!

Architecture Overview

Minion Backup is made up of SQL Server stored procedures, functions, tables, and jobs. There is also an optional PowerShell script for mass installation (MinionMassInstall.ps1) included in the download. The tables store configuration and log data; functions encrypt and decrypt sensitive data; stored procedures perform backup operations; and the jobs execute and monitor those backup operations on a schedule.

This section provides a brief overview of Minion Backup elements at a high level: configuration hierarchy, include/exclude precedence, run time configuration, logging and alerting.

Note: Minion Backup is installed in the master database by default. You certainly can install Minion in another database (like a DBAdmin database), but when you do, you must also verify that the job points to the appropriate database.

Configuration Settings Hierarchy

The basic configuration for backup – including most of the BACKUP DATABASE and BACKUP LOG options – is stored in a table: **Minion.BackupSettings**. A default row in Minion.BackupSettings (DBName='MinionDefault') provides settings for any database that doesn't have its own specific settings.

There is a hierarchy of granularity in Minion.BackupSettings, where more specific configuration levels completely override the less specific levels. That is:

1. **The MinionDefault row** applies to all databases that do NOT have any database-specific rows.
2. **A MinionDefault row with BackupType='Full'** (or Log, or Diff) provides settings for that backup type, for all databases that do NOT have any database-specific rows. This overrides the MinionDefault / All row.

3. **A database-specific row with BackupType='All'** causes all of that database's backup settings to come from that particular row (not from a MinionDefault row).
4. **A database-specific row with BackupType='Full'** (or Log, or Diff) causes all of that database's backup settings *for that backup type* to come from that particular row (not from a MinionDefault row, nor from the database-specific row where backupType='All').

The Configuration Settings Hierarchy Rule

If you provide a database-specific row, be sure that all backup types are represented in the table for that database. For example, if you insert a row for DBName='DB1', BackupType='Full', then also insert a row for DBName='DB1', BackupType='All' (or, alternately, two rows for DBName='DB1': one for Diff, and one for Log). Once you configure the settings context at the database level, the context *stays* at the database level, and not the default 'MinionDefault' level.

This document refers to the Configuration Hierarchy Settings Rule throughout, in situations where we must insert additional row(s) to provide for all backup types.

Note: "Exclude" is a minor exception to the hierarchy rules. If Exclude=1 for a database where BackupType='All', then all backups for that database are excluded.

Other tables hold additional backup configuration settings, and follow a *similar* hierarchy pattern.

Example 1: Proper Configuration

Let us take a simple example, in which these are the contents of the Minion.BackupSettings table (not all columns are shown here):

ID	DBName	BackupType	Exclude	DBPreCode
1	MinionDefault	All	0	'Exec SP1;'
2	DB1	All	0	'Exec SP1;'
3	DB1	Full	0	NULL

There are a total of 30 databases on this server. As backups run throughout the week, the settings for individual databases will be selected as follows:

- **Full backups** of database **DB1** will use only the settings from the row with ID=3.
- **Differential and log backups** of database **DB1** will use only the settings from the row with ID=2.
- **All other database backups (full, log, and differential)** will use the settings from the row with ID=1.

Note that a value left at *NULL* in one of these fields means that Minion Backup will use the setting that the SQL Server instance itself uses. So in our example, full backups of DB1 will run no precode; while all other backups will run 'Exec SP1;' as the database precode.

Example 2: Improper Configuration

Now let's walk through another simple example, in which these are the contents of the Minion.BackupSettings table (not all columns are shown here):

ID	DBName	BackupType	Exclude	DBPreCode
1	MinionDefault	All	0	'Exec SP1;'
2	DB1	Diff	0	'Exec SP1;'
3	DB1	Full	0	NULL

There are a total of 30 databases on this server. As backups run throughout the week, the settings for individual databases will be selected as follows:

- **Full backups** of database **DB1** will use only the settings from the row with ID=3.
- **Differential backups** of database **DB1** will use only the settings from the row with ID=2.
- **Log backups** of database **DB1** will *fail*, because no row exists that covers DB1 / log backups. Again: because we have specified settings for DB1 at the database level, Minion Backup will NOT use the MinionDefault settings for DB1.
- **All other database backups (full, log, and differential)** will use the settings from the row with ID=1.

DB1 log backup failures will show up in the log tables (most easily viewable in Minion.BackupLogDetails, which will show a status that begins with "FATAL ERROR").

Example 3: The "Exclude" Exception

Here we will demonstrate the effect of "Exclude" in rows of BackupType='All'. In this example, these are the contents of the Minion.BackupSettings table (not all columns are shown here):

ID	DBName	BackupType	Exclude	DBPreCode
1	MinionDefault	All	0	'Exec SP1;'
2	DB1	All	1	'Exec SP1;'
3	DB1	Full	0	NULL

There are a total of 30 databases on this server. As backups run throughout the week, the settings for individual databases will be selected as follows:

- **Backups of all types** for database **DB1** will be excluded, because of the row with ID=2. The log will not display failed backups for DB1; there will simply be no entry in the log for DB1 backups, as they are excluded.
- **Even full backups** of database **DB1** will be excluded.
- **All other database backups (full, log, and differential)** will use the settings from the row with ID=1.

For more information, see the configuration sections in ["How To" Topics: Basic Configuration](#) (such as ["How to: Configure settings for a single database"](#)), and ["Minion.BackupSettings"](#)

Include and Exclude Precedence

Minion Backup allows you to specify lists of databases to include in a backup routine, in several different ways. First of all, databases are always divided into "system" and "user" databases.

Include and Exclude strings

Within those divisions, the primary means of identifying what databases should be backed up in a given operation is by the use of Include and Exclude strings. As noted in the following section ([“Run Time Configuration”](#)), Include and Exclude can be defined as part of either a table configured schedule, or a parameter based schedule.

The important point to understand now, however, is how Include and Exclude work at a basic level. Include and Exclude may each have one of three kinds of values:

- ‘All’ or *NULL* (which also means ‘All’)
- ‘Regex’
- An explicit, comma-delimited list of database names and LIKE expressions (e.g., @Include=‘DB1,DB2%’).

Note: For this initial discussion, we are ignoring the existence of the Exclude *bit*, while we introduce the Include and Exclude *strings*. We’ll fold the Exclude bit concept back in at the end of the section.

The following table outlines the interaction of Include and Exclude:

	Exclude=‘All’ or IS NULL	Exclude=Regex	Exclude=[Specific list]
Include=‘All’ or IS NULL	Run all backups	Run all, minus regex exclude	Run all, minus explicit exclude
Include=Regex	Run only databases that match the configured RegEx expression	Run only databases that match the configured RegEx expression	Run only databases that match the configured RegEx expression
Include=[Specific list]	Run only specific includes	Run only specific includes	Run only specific includes

Note that regular expressions phrases are defined in a special settings table (Minion.DBMaintRegexLookup).

Let us look at a handful of scenarios, using this table:

- **Include IS NULL, Exclude IS NULL** – Run all backups.
- **Include = ‘All’, Exclude = ‘DB%’** – Run all backups except those beginning with “DB”.
- **Include=‘Regex’, Exclude=‘DB2’** – Run only databases that match the configured RegEx expression. (The Exclude is ignored.)

IMPORTANT: You will note that Exclude is ignored in any case where Include is not ‘All’/NULL. **Whether Include is Regex or is a specific list, an explicit Include should be the final word.** The reason for this rule is that we never want a scenario where a database simply *cannot* be backed up.

Exclude bit

In addition to the Include and Exclude strings, Minion Backup also provides an “Exclude” bit in the primary settings table (Minion.BackupSettings) that allows you to exclude backups for a particular database, or a particular database and backup type.

The following table outlines the interaction of the Include string and the Exclude *bit*:

	Exclude=0	Exclude=1
Include='All' or IS NULL	Run all backups	Run all, minus excluded databases' backup types
Include=Regex	Run only databases that match the configured RegEx expression	Run only databases that match the configured RegEx expression
Include=[Specific list]	Run only specific includes	Run only specific includes

Let us look at a handful of scenarios, using this table:

- **Include IS NULL, Exclude bit=0** – Run all backups.
- **Include = 'All', Exclude = 1 for DB2 / All** – Run all backups except DB2.
- **Include='Regex', Exclude=1 for DB2 / All** – Run only databases that match the configured RegEx expression. (The Exclude bit is ignored.)

IMPORTANT: You will note that the Exclude bit, like the Exclude string, is ignored in any case where Include is not 'All'/NULL. **Whether Include is Regex or is a specific list, an explicit Include should be the final word.** The reason for this rule is that we never want a scenario where a database simply *cannot* be backed up.

Run Time Configuration

The main Minion Backup stored procedure – Minion.BackupMaster – can be run in one of two ways: with table configuration, or with parameters.

Run Minion.BackupMaster using table configuration: If you run Minion.BackupMaster without parameters, the procedure uses the Minion.BackupSettingsServer table to determine its runtime parameters (including the schedule of backup jobs per backup type, and which databases to Include and Exclude). This is how MB operates by default, to allow for the most flexible backup scheduling with as few jobs as possible.

For more information, see the sections "[How To: Change Backup Schedules](#)", "[Minion.BackupSettingsServer](#)", and "[Minion.BackupMaster](#)".

Run Minion.BackupMaster with parameters: The procedure takes a number of parameters that are specific to the current maintenance run. For example:

- Use @DBType to specify 'System' or 'User' databases.
- Use @BackupType to specify Full, Log, or Diff backups.
- Use @StmtOnly to generate backup statements, instead of running them.
- Use @Include to back up a specific list of databases, or databases that match a LIKE expression. Alternately, set @Include='All' or @Include=NULL to back up all databases.
- Use @Exclude to exclude a specific list of databases from backup.

- Use @ReadOnly to (1) include ReadOnly databases, (2) exclude ReadOnly databases, or (3) only include ReadOnly databases.

For more information, see the section “[How To: Change Backup Schedules](#)” and “[Minion.BackupMaster](#)”.

Logging

As a Minion Backup routine runs, it keeps logs of all activity. The two primary log tables are:

- **Minion.BackupLog** – a log of activity at the batch level.
- **Minion.BackupLogDetails** – a log of activity at the database level.

The Status column for the current backup run is updated continually in each of these tables while the batch is running. This way, status information (**Live Insight**) is available to you while backup is still running, and historical data is available after the fact for help in planning future operations, reporting, troubleshooting, and more.

Minion Backup logs additional information in a number of other tables, including:

- **Minion.BackupDebug** – Log of high level debug data.
- **Minion.BackupDebugLogDetails** – Log of detailed debug data.
- **Minion.BackupFileListOnly** – log of RESTORE FILELISTONLY output for each backup taken
- **Minion.BackupFiles** – a log of all backup files (whether they originate from a database backup, a certificate backup, a copy, or a move). Note that a backup that is striped to 10 files will have 10 rows in this table.
- **Minion.SyncCmds** – a log of commands used to synchronize settings and log tables to configured synchronization servers. This table is both a log table and a work table: the synchronization process uses Minion.SyncCmds to push the synchronization commands to target servers, and it is also a log of those commands (complete and incomplete).
- **Minion.SyncErrorCmds** – a log of synchronization commands that have failed, to be retried again later.

Minion Backup maintains all log tables are automatically. The retention period for all log tables is set in the HistoryRetDays field in Minion.BackupSettings.

Alerting

Minion Backup doesn’t include an alerting mechanism, though you can write one easily using the log tables. The jobs will almost always show a “succeeded” status, even if one or more backups fail; the error and failed backup will be recorded in the log.

Here is one example of an alerting mechanism. Ideally, you could create a stored procedure, and simply call that procedure in step 2 of your backup job(s).

```
---- Declare variables (could be SP parameters)
```

```

DECLARE @profile_name sysname = 'Server DBMail' ,
        @recipients VARCHAR(MAX) = 'SQLsupport@Company.com';

---- Declare and set internal variables
DECLARE @Query NVARCHAR(MAX) ,
        @Subject NVARCHAR(255);

SET @Query = 'SELECT ID ,
              ExecutionDateTime ,
              ServerLabel ,
              @@SERVERNAME AS Servername ,
              STATUS ,
              PctComplete ,
              DBName
FROM master.Minion.BackupLogDetails
WHERE ExecutionDateTime = ( SELECT MAX(ExecutionDateTime)
                          FROM master.Minion.BackupLogDetails
                          )
AND STATUS NOT IN ("All Complete", "Complete");';

SELECT @Subject = @@Servername + ' ALERT: Log backup failed';

---- Execute query to pull the rowcount
EXEC sp_executesql @Query;

---- If query returned rows, email to recipients
IF @@ROWCOUNT > 0
    EXEC msdb.dbo.sp_send_dbmail
        @profile_name = @profile_name,
        @recipients = @recipients,
        @query = @Query ,
        @subject = @Subject,
        @attach_query_result_as_file = 0 ;

```

Important notes:

- This is just one example of how you could code a backup alert for Minion Backup. Review and modify this code for your own use, if you like, or grow your own.
- We do not recommend basing alerts off of Status='Complete', because a successful backup run will not always be marked "Complete". It will be "All Complete" if the backup batch was run by Minion.BackupMaster, and "Complete" if run by Minion.BackupDB.



Minion Enterprise Hint

Minion Backup doesn't include an alerting mechanism, though you can write one easily using the log tables. Minion Enterprise provides central backup reporting and alerting. The ME alert for all databases includes the reasons why any backups fail, across the entire enterprise. Further, you can set customized alerting thresholds at various levels (server, database, and backup type). For example, you might set the alert thresholds for some servers to alert on missing backups after a day; for a handful of databases, to alert at half a day; for log backups, alert on 5 hours; and for development servers, not at all. The choice is yours.

See www.MinionWare.net for more information, or email us today at Support@MidnightDBA.com for a demo!

“How To” Topics: Basic Configuration

How To: Configure settings for a single database

Default settings for the whole system are stored in the Minion.BackupSettings table. To specify settings for a specific database that override those defaults (for that database), insert a row for that database to the Minion.BackupSettings table. For example, we want to fine tune settings for DB1, so we use the following statement:

```
INSERT INTO Minion.BackupSettings
  ([DBName] ,
  [Port] ,
  [BackupType] ,
  [Exclude] ,
  [GroupOrder] ,
  [GroupDBOrder] ,
  [Mirror] ,
  [DelFileBefore] ,
  [DelFileBeforeAgree] ,
  [LogLoc] ,
  [HistRetDays] ,
  [DynamicTuning] ,
  [Verify] ,
  [ShrinkLogOnLogBackup] ,
  [MinSizeForDiffInGB] ,
  [DiffReplaceAction] ,
  [Encrypt] ,
```

```

        [Checksum] ,
        [Init] ,
        [Format] ,
        [IsActive] ,
        [Comment]
    )
SELECT 'DB1' AS [DBName] ,
    1433 AS [Port] ,
    'All' AS [BackupType] ,
    0 AS [Exclude] ,
    50 AS [GroupOrder] ,
    0 AS [GroupDBOrder] ,
    0 AS [Mirror] ,
    0 AS [DelFileBefore] ,
    0 AS [DelFileBeforeAgree] ,
    'Local' AS [LogLoc] ,
    90 AS [HistRetDays] ,
    1 AS [DynamicTuning] ,
    '0' AS [Verify] ,
    1 AS [ShrinkLogOnLogBackup] ,
    20 AS [MinSizeForDiffInGB] ,
    'Log' AS [DiffReplaceAction] ,
    0 AS [Encrypt] ,
    1 AS [Checksum] ,
    1 AS [Init] ,
    1 AS [Format] ,
    1 AS [IsActive] ,
    'DB1 is high priority; better backup order and history retention.' AS [Comment];

```



Minion Backup comes with a utility stored procedure, named `Minion.CloneSettings`, for easily creating insert statements like the example above. For more information, see the “[Minion.CloneSettings](#)” section below.

IMPORTANT:

- If you enter a row for a database and/or backup type, that row completely overrides the settings for that particular database (and/or backup type). For example, the row inserted above will be the source of all settings – even if they are NULL – for all DB1 database backups. For more information, see the “[Configuration Settings Hierarchy](#)” section above.
- Follow [the Configuration Settings Hierarchy Rule](#): If you provide a database-specific row, be sure that all backup types are represented in the table for that database. For example, if you insert a row for `DBName='DB1'`, `BackupType='Full'`, then also insert a row for `DBName='DB1'`, `BackupType='All'` (or individual rows for DB1 log and DB1 differential backups). Once you configure the settings context at the database level, the context *stays* at the database level (and not the default ‘MinionDefault’ level).
- **A quick note about log backups:** In SQL Server, a database must have had a full backup before a log backup can be taken. Minion Backup prevents this; if you try to take a log backup, and the database doesn't have a restore base, then the system will remove the log backup from the list. MB will not

attempt to take a log backup until there's a full backup in place. Though it may seem logical to perform a full backup instead of a full, we do not do this, because log backups can be taken very frequently; we don't want to make what is usually a quick operation into a very long operation.

How To: Configure settings for all databases

When you first install an instance of Minion Backup, default settings for the whole system are stored in the Minion.BackupSettings table row where DBName='MinionDefault' and BackupType='All'. To change settings for all databases on the server, update the values for that default row.

For example, you might want to verify backups after the batch (after all backups for one operation are complete):

```
UPDATE Minion.BackupSettings
SET    Verify='AfterBatch'
WHERE  DBName = 'MinionDefault'
      AND BackupType = 'All';
```

WARNING: “Verify” for backups must be used with caution. Verifying backups can take a long time, and you could hold up subsequent backups while running the verify. If you would like to run verify, we recommend using AfterBatch.

Over time, you may have entered one or more database-specific rows for individual databases and/or backup types (e.g., DBName='DB1' and BackupType='Full'). In this case, the settings in the default “MinionDefault/All” row do not apply to those database/backup types. You can of course update the entire table – both the default row, and any database-specific rows – with new settings, to be sure that the change is universal for that instance. So, if you want the history retention days to be 90 (instead of the default, 60 days), run the following:

```
UPDATE Minion.BackupSettings
SET    HistRetDays = 90;
```



Minion Enterprise Hint

Minion Enterprise, in conjunction with Minion Backup, can manage – not just gather and view, but manage – backup settings across all SQL Server instances, centrally. One classic case: you can change backup location for hundreds of servers, using a simple UPDATE statement in the Minion Enterprise central repository.

See www.MinionWare.net for more information, or email us today at Support@MidnightDBA.com for a demo!

How To: Back up databases in a specific order

You can choose the order in which databases will be maintained. For example, let's say that you want your databases backed up in this order:

1. [YourDatabase] (it's the most important database on your system)
2. [Semi]
3. [Lame]
4. [Unused]

In this case, we would insert a row into the `Minion.BackupSettings` table for each one of the databases, specifying either `GroupDBOrder`, `GroupOrder`, or both, as needed.

NOTE: For `GroupDBOrder` and `GroupOrder`, higher numbers have a greater "weight" - they have a higher priority - and will be backed up earlier than lower numbers. Note also that these columns are `TINYINT`, so weighted values must fall between 0 and 255.

NOTE: When you insert a row for a database, the settings in that row override **all** of the default backup settings for that database. So, inserting a row for [YourDatabase] means that **ONLY** backup settings from that row will be used for [YourDatabase]; none of the default settings will apply to [YourDatabase].

NOTE: Any databases that rely on the default system-wide settings (represented by the row where `DBName='MinionDefault'`) will be backed up according to the values in the `MinionDefault` columns `GroupDBOrder` and `GroupOrder`. By default, these are both 0 (lowest priority), and so non-specified databases would be backed up last.

Because we have so few databases in this example, the simplest method is to assign the heaviest "weight" to YourDatabase, and lesser weights to the other databases, in decreasing order. In our example, we would insert four rows. Note that, for brevity, we use far fewer columns in our examples than you would need in an actual environment:

-- Insert BackupSettings row for [YourDatabase], GroupOrder=255 (first)

```
INSERT INTO [Minion].[BackupSettings]
( [DBName] ,
  [BackupType] ,
  [Exclude] ,
  [GroupOrder] ,
  [GroupDBOrder] ,
  [LogLoc] ,
  [HistRetDays] ,
  [ShrinkLogOnLogBackup] ,
  [ShrinkLogThresholdInMB] ,
  [ShrinkLogSizeInMB]
)
```

```

SELECT 'YourDatabase' AS [DBName] ,
      'All' AS [BackupType] ,
      0 AS [Exclude] ,
      255 AS [GroupOrder] ,
      0 AS [GroupDBOrder] ,
      'Local' AS [LogLoc] ,
      60 AS [HistRetDays] ,
      0 AS [ShrinkLogOnLogBackup] ,
      0 AS [ShrinkLogThresholdInMB] ,
      0 AS [ShrinkLogSizeInMB];

```

-- Insert BackupSettings row for "Semi", GroupOrder=150 (after [YourDatabase])

```

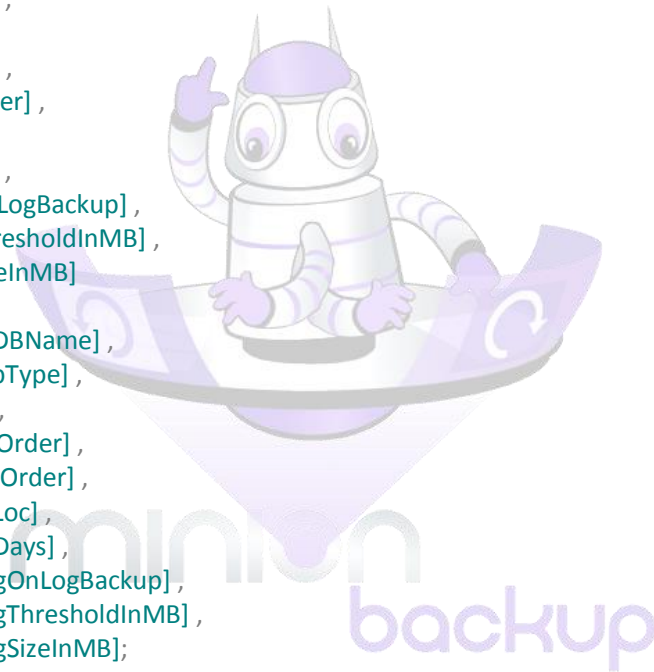
INSERT INTO [Minion].[BackupSettings]
( [DBName] ,
  [BackupType] ,
  [Exclude] ,
  [GroupOrder] ,
  [GroupDBOrder] ,
  [LogLoc] ,
  [HistRetDays] ,
  [ShrinkLogOnLogBackup] ,
  [ShrinkLogThresholdInMB] ,
  [ShrinkLogSizeInMB]
)

```

```

SELECT 'Semi' AS [DBName] ,
      'All' AS [BackupType] ,
      0 AS [Exclude] ,
      150 AS [GroupOrder] ,
      0 AS [GroupDBOrder] ,
      'Local' AS [LogLoc] ,
      60 AS [HistRetDays] ,
      0 AS [ShrinkLogOnLogBackup] ,
      0 AS [ShrinkLogThresholdInMB] ,
      0 AS [ShrinkLogSizeInMB];

```



-- Insert BackupSettings row for "Lame", GroupOrder=100 (after "Semi")

```

INSERT INTO [Minion].[BackupSettings]
( [DBName] ,
  [BackupType] ,
  [Exclude] ,
  [GroupOrder] ,
  [GroupDBOrder] ,
  [LogLoc] ,
  [HistRetDays] ,
  [ShrinkLogOnLogBackup] ,
  [ShrinkLogThresholdInMB] ,
  [ShrinkLogSizeInMB]
)

```

```

SELECT 'Lame' AS [DBName] ,

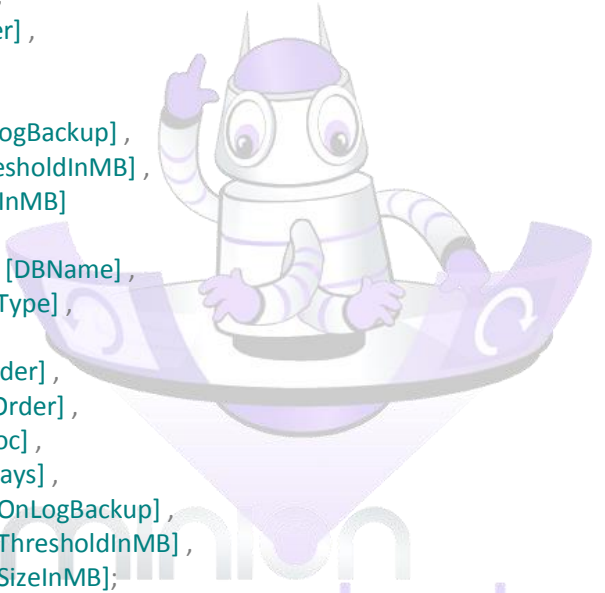
```

```

'All' AS [BackupType] ,
0 AS [Exclude] ,
100 AS [GroupOrder] ,
0 AS [GroupDBOrder] ,
'Local' AS [LogLoc] ,
60 AS [HistRetDays] ,
0 AS [ShrinkLogOnLogBackup] ,
0 AS [ShrinkLogThresholdInMB] ,
0 AS [ShrinkLogSizeInMB];

-- Insert BackupSettings row for "Unused", GroupOrder=50 (after [Lame])
INSERT INTO [Minion].[BackupSettings]
( [DBName] ,
  [BackupType] ,
  [Exclude] ,
  [GroupOrder] ,
  [GroupDBOrder] ,
  [LogLoc] ,
  [HistRetDays] ,
  [ShrinkLogOnLogBackup] ,
  [ShrinkLogThresholdInMB] ,
  [ShrinkLogSizeInMB]
)
SELECT 'Unused' AS [DBName] ,
'All' AS [BackupType] ,
0 AS [Exclude] ,
50 AS [GroupOrder] ,
0 AS [GroupDBOrder] ,
'Local' AS [LogLoc] ,
60 AS [HistRetDays] ,
0 AS [ShrinkLogOnLogBackup] ,
0 AS [ShrinkLogThresholdInMB] ,
0 AS [ShrinkLogSizeInMB];

```



For a more complex ordering scheme, we could divide databases up into groups, and then order the backups both by group, and within each group. The pseudocode for this example might be:

- Insert rows for databases YourDatabase and Semi, both with GroupOrder = 200
 - Row YourDatabase: GroupDBOrder = 255
 - Row Semi: GroupDBOrder = 100
- Insert rows for databases Lame and Unused, both with GroupOrder = 100
 - Row YourDatabase: Lame = 255
 - Row Semi: Unused = 100

The resulting backup order would be as follows:

1. YourDatabase
2. Semi
3. Lame

4. Unused

How To: Change backup schedules

Minion Backup offers you a choice of scheduling options:

- You can use the `Minion.BackupSettingsServer` table to configure flexible backup scheduling scenarios;
- Or, you can use the traditional approach of one job per backup schedule;
- Or, you can use a hybrid approach that employs a bit of both options.

For more information about backup schedules, see "[About: Backup Schedules](#)".

Table based scheduling

When Minion Backup is installed, it uses a single backup job to run the stored procedure `Minion.BackupMaster` with no parameters, every 30 minutes. When the `Minion.BackupMaster` procedure runs without parameters, it uses the `Minion.BackupSettingsServer` table to determine its runtime parameters (including the schedule of backup jobs per backup type). This is how MB operates by default, to allow for the most flexible backup scheduling with as few jobs as possible.

This document explains table based scheduling in the Quick Start section "[Table based scheduling](#)".

Parameter based scheduling (traditional approach)

Other SQL Server native backup solutions traditionally use one backup job per schedule. That usually means at a minimum: one job for system database full backups, one job for user database full backups, and one job for log backups.

To use the traditional approach of one job per backup schedule:

1. Disable or delete the `MinionBackup-Auto` job.
2. Configure new jobs for each backup schedule scenario you need.

Note: We *highly* recommend always using the `Minion.BackupMaster` stored procedure to run backups. While it is possible to use `Minion.BackupDB` to execute backups, doing so will bypass much of the configuration and logging benefits that Minion Backup was designed to provide.

Run `Minion.BackupMaster` with parameters: The procedure takes a number of parameters that are specific to the current maintenance run. (For full documentation of `Minion.BackupMaster` parameters, see the "[Minion.BackupMaster](#)" section.)

To configure traditional, one-job-per-schedule backups, you might configure three new jobs:

- `MinionBackup-SystemFull`, to run full backups for system databases nightly at 9pm. The job step should be something similar to:

```
EXEC Minion.BackupMaster @DBType = 'System'  
    , @BackupType = 'Full'
```

```
, @StmtOnly = 0  
, @ReadOnly = 1;
```

- *MinionBackup-UserFull*, to run full backups for user databases nightly at 10pm. The job step should be something similar to:

```
EXEC Minion.BackupMaster @DBType = 'User'  
, @BackupType = 'Full'  
, @StmtOnly = 0  
, @ReadOnly = 1;
```

- *MinionBackup-Log*, to run log backups for user databases hourly. The job step should be something similar to:

```
EXEC Minion.BackupMaster @DBType = 'User'  
, @BackupType = 'Log'  
, @StmtOnly = 0  
, @ReadOnly = 2;
```

Hybrid scheduling

It is possible to use both methods – table based scheduling, and traditional scheduling – by one job that runs `Minion.BackupMaster` with no parameters, and one or more jobs that run `Minion.BackupMaster` with parameters.

We recommend against this, as hybrid scheduling has little advantage over either method, and increases the complexity of your backup scenario. However, it may be that there are as yet unforeseen situations where hybrid backup scheduling might be very useful.

How To: Generate back up statements only

Sometimes it is useful to generate backup statements and run them by hand, either individually or in small groups. To generate backup statements without running the statements, run the procedure `Minion.BackupMaster` with the parameter `@StmtOnly` set to 1.

Example code - The following code will generate full backup statements for all system databases.

```
EXEC [Minion].[BackupMaster]  
    @DBType = 'System',  
    @BackupType = 'Full',  
    @Include = 'All',  
    @StmtOnly = 1;
```

Running `Minion.BackupMaster` with `@StmtOnly=1` will generate a list of `Minion.BackupDB` execution statements, all set to `@StmtOnly=1`. Running these `Minion.BackupDB` statements will generate the “BACKUP DATABASE” or “BACKUP LOG” statements.

This is an excellent way to discover what settings Minion Backup will use for a particular database (or set of databases). For more information – and another method – for determining the settings Minion Backup will use, see the “Discussion” portion of the [“Minion.BackupStmtGet”](#) section below.

How To: Back up only databases that are not marked READ_ONLY

Using the Minion.BackupMaster stored procedure, you can choose whether or not to include READ_ONLY databases in the backup routine:

- @ReadOnly = 1 will include READ_ONLY databases in the backup routine. This is the default option.
- @ReadOnly = 2 will NOT include READ_ONLY databases in the backup routine.
- @ReadOnly = 3 will ONLY include READ_ONLY databases in the backup routine.

To backup only databases that are not marked READ_ONLY, run the procedure Minion.BackupMaster with the parameter @ReadOnly set to 2. For example, to back up only the read/write user databases, use the following call:

```
EXEC [Minion].[BackupMaster]
    @DBType = 'User',
    @BackupType = 'Full',
    @Include = 'All',
    @ReadOnly = 2;
```

To back up only the READ_ONLY databases, use the following call:

```
EXEC [Minion].[BackupMaster]
    @DBType = 'User',
    @BackupType = 'Full',
    @Include = 'All',
    @ReadOnly = 3;
```

How To: Include databases in backups

By default, Minion Backup is configured to back up all databases. As you fine tune your backup scenarios and schedules, you may want to configure specific subsets of databases to be backed up with different options, or at different times.

You can limit the set of databases to be backed up in a single operation via an explicit list, LIKE expressions, or regular expressions. In the following two sections, we will work through the way to do this first via table based scheduling, and then in traditional scheduling.

NOTE: The use of the regular expressions include and exclude features are not supported in SQL Server 2005.

Include databases in table based scheduling

Table based scheduling pulls backup schedules and other options from the Minion.BackupSettingsServer table. In this table, you have the following options for configuring which databases to include in backup operations:

- **To include all databases in a backup operation, set Include = 'All' (or NULL)** for the relevant row(s).
- **To include a specific list of databases, set Include = a comma delimited list of those database names, and/or LIKE expressions.** (For example: 'YourDatabase, DB1, DB2', or 'YourDatabase, DB%'.)

- **To include databases based on regular expressions, set Include = 'Regex'.** Then, configure the regular expression in the **Minion.DBMaintRegexLookup** table.

We will use the following sample data as we demonstrate each of these options. This is a subset of Minion.BackupSettingsServer columns:

ID	DBType	BackupType	Day	BeginTime	EndTime	Include	Exclude
1	System	Full	Daily	22:00:00	22:30:00	NULL	NULL
2	User	Full	Friday	23:00:00	23:30:00	DB1,DB2	NULL
3	User	Full	Saturday	23:00:00	23:30:00	DB10%	NULL
4	User	Full	Sunday	23:00:00	23:30:00	Regex	NULL
5	User	Log	Daily	00:00:00	23:59:00	NULL	NULL

And, these is the contents of the Minion.DBMaintRegexLookup table:

Action	MaintType	Regex
Include	Backup	DB[3-5](?!d)

Based on this data, Minion Backup would perform backups as follows:

- Full system database backups run daily at 10pm.
- Full user database backups for DB1 and DB2 run Fridays at 11pm.
- Full user database backups for all databases beginning with "DB10" run Saturdays at 11pm.
- Full user database backups for databases included in the regular expressions table (Minion.DBMaintRegexLookup), run Sundays at 11pm. (This particular regular expression includes DB3, DB4, and DB5, but does not include any database with a 2 digit number at the end, such as DB35.)
- User log backups run daily (as often as the backup job runs).

Note that you can create more than one regular expression in Minion.DBMaintRegexLookup. For example:

- **To use Regex to include DB3, DB4, and DB5:** insert a row like the example above, where Regex = 'DB[3-5](?!d)'.
- **To use Regex to include any database beginning with the word "Market" followed by a number:** insert a row where Regex='Market[0-9]'.
- **With these two rows,** a backup operation with @Include='Regex' will backup *both* the DB3-DB5 databases, and the databases Marketing4 and Marketing308 (and similar others, if they exist).

Include databases in traditional scheduling

We refer the common practice of configuring backups in separate jobs (to allow for multiple schedules) as "traditional scheduling". Shops that use traditional scheduling will run Minion.BackupMaster with parameters configured for each particular backup run.

You have the following options for configuring which databases to include in backup operations:

- **To include all databases in a backup operation, set @Include = 'All' (or NULL).**

- To include a specific list of databases, set **@Include** = a comma delimited list of those database names, and/or LIKE expressions. (For example: 'YourDatabase, DB1, DB2', or 'YourDatabase, DB%'.)
- To include databases based on regular expressions, set **@Include** = 'Regex'. Then, configure the regular expression in the **Minion.DBMaintRegexLookup** table.

The following example executions will demonstrate each of these options.

First, to run full user backups on all databases, we would execute Minion.BackupMaster with these (or similar) parameters:

```
-- @Include = NULL for all databases
EXEC Minion.BackupMaster
    @DBType = 'User',
    @BackupType = 'Full',
    @StmtOnly = 1,
    @Include = NULL,
    @Exclude=NULL,
    @ReadOnly=1;
```

To include a specific list of databases:

```
-- @Include = a specific database list (YourDatabase, all DB1% DBs, and DB2)
EXEC Minion.BackupMaster
    @DBType = 'User',
    @BackupType = 'Full',
    @StmtOnly = 1,
    @Include = 'YourDatabase,DB1%,DB2',
    @Exclude=NULL,
    @ReadOnly=1;
```

To include databases based on regular expressions, first insert the regular expression into the Minion.DBMaintRegexLookup table, and then execute Minion.BackupMaster with @Include='Regex':

```
INSERT INTO Minion.DBMaintRegexLookup
    ( [Action],
      [MaintType],
      [Regex]
    )
SELECT 'Include' AS [Action],
       'Backup' AS [MaintType],
       'DB[3-5](?!\d)' AS [Regex]
-- @Include = 'Regex' for regular expressions
EXEC Minion.BackupMaster
    @DBType = 'User',
    @BackupType = 'Full',
    @StmtOnly = 1,
    @Include = 'Regex',
    @Exclude=NULL,
```

```
@ReadOnly=1;
```

For information on Include/Exclude precedence (that applies to both the Minion.BackupSettingsServer columns, and to the parameters), see [“Include and Exclude Precedence”](#).

How To: Exclude databases from backups

By default, Minion Backup is configured to back up all databases. As you fine tune your backup scenarios and schedules, you may want to exclude certain databases from scheduled backup operations, or even from *all* backup operations.

You can exclude databases from all backup operations via the Exclude column in Minion.BackupSettings. Or, you can exclude databases from a backup operation via an explicit list, LIKE expressions, or regular expressions. In the following three sections, we will work through Exclude=1, then excluding databases from table based scheduling, and finally excluding from traditional scheduling.

NOTE: The use of the regular expressions include and exclude features are not supported in SQL Server 2005.

Exclude a database from all backups

To exclude a database – for example, DB13 – from all backups, just insert a database-specific row for that database into Minion.BackupSettings, with BackupType=All and Exclude=1:

```
INSERT INTO Minion.BackupSettings
  ([DBName] ,
  [BackupType] ,
  [Exclude] ,
  [LogLoc] ,
  [HistRetDays] ,
  [IsActive]
  )
```

```
SELECT 'DB13' AS [DBName] ,
  'All' AS [BackupType] ,
  1 AS [Exclude] ,
  'Local' AS [LogLoc] ,
  60 AS [HistRetDays] ,
  1 AS [IsActive] ;
```

This insert has a bare minimum of options, as the row is only intended to exclude DB13 from the backup routine. We recommend configuring individual database rows with the full complement of settings if there is a chance that backups may be re-enabled for that database in the future.

IMPORTANT: Exclude=1 can be overridden by an explicit Include. For more information, see [“Include and Exclude Precedence”](#).

Exclude databases in table based scheduling

Table based scheduling pulls backup schedules and other options from the Minion.BackupSettingsServer table. In this table, you have the following options for configuring which databases to exclude from backup operations:

- **To exclude a specific list of databases, set Exclude = a comma delimited list of those database names, and/or LIKE expressions.** (For example: 'YourDatabase, DB1, DB2', or 'YourDatabase, DB%'.)
- **To exclude databases based on regular expressions, set Exclude = 'Regex'.** Then, configure the regular expression in the **Minion.DBMaintRegexLookup** table.

We will use the following sample data as we demonstrate each of these options. This is a subset of Minion.BackupSettingsServer columns:

ID	DBType	BackupType	Day	BeginTime	EndTime	Include	Exclude
1	System	Full	Daily	22:00:00	22:30:00	NULL	NULL
2	User	Full	Friday	23:00:00	23:30:00	NULL	DB1,DB2
3	User	Full	Saturday	23:00:00	23:30:00	NULL	DB10%
4	User	Full	Sunday	23:00:00	23:30:00	NULL	Regex
5	User	Log	Daily	00:00:00	23:59:00	NULL	NULL

And, these is the contents of the Minion.DBMaintRegexLookup table:

Action	MaintType	Regex
Exclude	Backup	DB[3-5](?!d)

Based on this data, Minion Backup would perform backups as follows:

- Full system database backups run daily at 10pm.
- Full user database backups for all databases – except DB1 and DB2 – run Fridays at 11pm.
- Full user database backups for all databases – except those beginning with “DB10” – run Saturdays at 11pm.
- Full user database backups for all databases – except for those excluded via the regular expressions table (Minion.DBMaintRegexLookup) – run Sundays at 11pm. (This particular regular expression excludes DB3, DB4, and DB5 from backups, but does not exclude any database with a 2 digit number at the end, such as DB35.)
- User log backups run daily (as often as the backup job runs).

Note that you can create more than one regular expression in Minion.DBMaintRegexLookup. For example:

- **To use Regex to exclude DB3, DB4, and DB5:** insert a row like the example above, where Regex = 'DB[3-5](?!d)'.
- **To use Regex to exclude any database beginning with the word “Market” followed by a number:** insert a row where Regex='Market[0-9]'.

- **With these two rows**, a backup operation with `@Exclude='Regex'` will exclude *both* the DB3-DB5 databases, and the databases Marketing4 and Marketing308 (and similar others, if they exist) from backups.

Exclude databases in traditional scheduling

We refer the common practice of configuring backups in separate jobs (to allow for multiple schedules) as “traditional scheduling”. Shops that use traditional scheduling will run Minion.BackupMaster with parameters configured for each particular backup run.

You have the following options for configuring which databases to exclude from backup operations:

- **To exclude a specific list of databases, set `@Exclude` = a comma delimited list of those database names, and/or LIKE expressions.** (For example: ‘YourDatabase, DB1, DB2’, or ‘YourDatabase, DB%’.)
- **To exclude databases based on regular expressions, set `@ Exclude` = ‘Regex’.** Then, configure the regular expression in the **Minion.DBMaintRegexLookup** table.

The following example executions will demonstrate each of these options.

First, to exclude a specific list of databases:

```
-- @Exclude = a specific database list (YourDatabase, all DB1% DBs, and DB2)
EXEC Minion.BackupMaster
    @DBType = 'User',
    @BackupType = 'Full',
    @StmtOnly = 1,
    @Include = NULL,
    @Exclude='YourDatabase,DB1%,DB2',
    @ReadOnly=1;
```

To exclude databases based on regular expressions, first insert the regular expression into the Minion.DBMaintRegexLookup table, and then execute Minion.BackupMaster with `@Exclude='Regex'`:

```
INSERT INTO Minion.DBMaintRegexLookup
    ([Action] ,
    [MaintType] ,
    [Regex]
    )
SELECT 'Exclude' AS [Action] ,
    'Backup' AS [MaintType] ,
    'DB[3-5](?!\d)' AS [Regex]
-- @Exclude = 'Regex' for regular expressions
EXEC Minion.BackupMaster
    @DBType = 'User',
    @BackupType = 'Full',
    @StmtOnly = 1,
    @Include = NULL,
    @Exclude='Regex',
```

@ReadOnly=1;

For information on Include/Exclude precedence (that applies to both the Minion.BackupSettingsServer columns, and to the parameters), see [“Include and Exclude Precedence”](#).

How To: Run code before or after backups

You can schedule code to run before or after backups, using precode and postcode. Pre- and postcode can be configured:

- Before or after database backups (either for one database, or for each of several databases in an operation)
- Before or after the entire backup operation

NOTE: Unless otherwise specified, pre- and postcode will run in the context of the Minion Backup’s database (wherever the Minion Backup objects are stored); it was a design decision not to limit the code that can be run to a specific database. Therefore, always use “USE” statements – or, for stored procedures, three-part naming convention – for pre- and postcode.

Database precode and postcode

Database precode and postcode run before and after an individual database; or, if there are multiple databases in the backup batch, before and after each database backup. Database precode and postcode presents several options:

- run code before or after a single database
- run code before or after each and every database
- run code before or after each of a few databases
- run code before or after all but a few databases

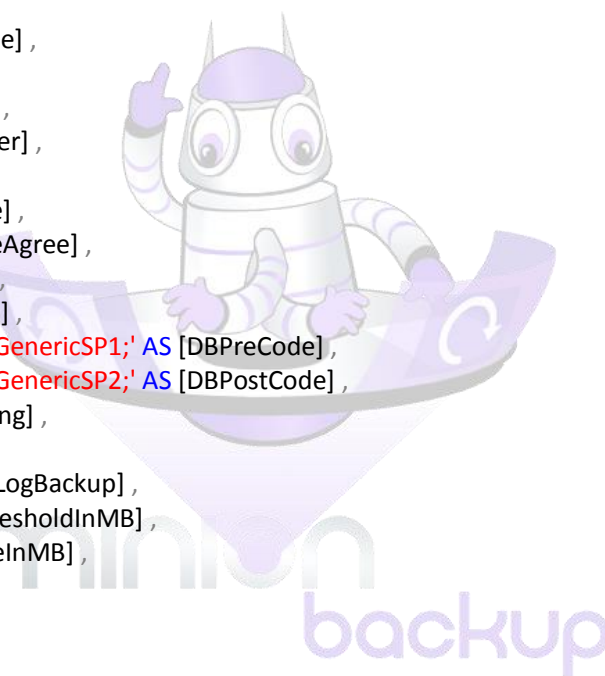
To run code before or after a single database, insert a row for the database into Minion.BackupSettings. Populate the column DBPreCode to run code before the backup operations for that database; populate the column DBPostCode to run code before the backup operations after that database. For example:

```
INSERT INTO Minion.BackupSettings
  ([DBName] ,
  [Port] ,
  [BackupType] ,
  [Exclude] ,
  [GroupOrder] ,
  [GroupDBOrder] ,
  [Mirror] ,
  [DelFileBefore] ,
  [DelFileBeforeAgree] ,
  [LogLoc] ,
  [HistRetDays] ,
```

```

[DBPreCode] ,
[DBPostCode] ,
[DynamicTuning] ,
[Verify] ,
[ShrinkLogOnLogBackup] ,
[ShrinkLogThresholdInMB] ,
[ShrinkLogSizeInMB] ,
[Encrypt] ,
[Checksum] ,
[Init] ,
[Format] ,
[IsActive] ,
[Comment]
)
SELECT 'DB5' AS [DBName] ,
NULL AS [Port] ,
'All' AS [BackupType] ,
0 AS [Exclude] ,
0 AS [GroupOrder] ,
0 AS [GroupDBOrder] ,
0 AS [Mirror] ,
0 AS [DelFileBefore] ,
0 AS [DelFileBeforeAgree] ,
'Local' AS [LogLoc] ,
60 AS [HistRetDays] ,
'EXEC master.dbo.GenericSP1;' AS [DBPreCode] ,
'EXEC master.dbo.GenericSP2;' AS [DBPostCode] ,
1 AS [DynamicTuning] ,
'0' AS [Verify] ,
0 AS [ShrinkLogOnLogBackup] ,
0 AS [ShrinkLogThresholdInMB] ,
0 AS [ShrinkLogSizeInMB] ,
0 AS [Encrypt] ,
1 AS [Checksum] ,
1 AS [Init] ,
1 AS [Format] ,
1 AS [IsActive] ,
NULL AS [Comment];

```



To run code before or after each and every database, update the MinionDefault row AND every database-specific rows (if any) in Minion.BackupSettings, populating the column DBPreCode or DBPostCode. For example:

```

UPDATE [Minion].[BackupSettings]
SET
    DBPreCode = 'EXEC master.dbo.GenericSP1;',
    DBPostCode = 'EXEC master.dbo.GenericSP1;'
WHERE DBName = 'MinionDefault'
AND BackupType = 'All';

```

```

UPDATE [Minion].[BackupSettings]
SET      DBPreCode = 'EXEC master.dbo.GenericSP1;',
        DBPostCode = 'EXEC master.dbo.GenericSP1;'
WHERE DBName = 'DB5'
        AND BackupType = 'All';

```

To run code before or after each of a few databases, insert one row for each of the databases into Minion.BackupSettings, populating the DBPreCode column and/or DBPostCode column as appropriate.

To run code before or after all but a few databases, update the MinionDefault row in Minion.BackupSettings, populating the DBPreCode column and/or the DBPostCode column as appropriate. This will set up the execution code for all databases. Then, to prevent that code from running on a handful of databases, insert a row for each of those databases to Minion.BackupSettings, and keep the DBPreCode and DBPostCode columns set to *NULL*.

For example, if we want to run the stored procedure dbo.SomeSP before each database *except* databases DB1, DB2, and DB3, we would:

1. Update row in Minion.BackupSettings for “MinionDefault”, setting PreCode to ‘EXEC dbo.SomeSP;’
2. Insert a row to Minion.BackupSettings for [DB1], establishing all appropriate settings, and setting DBPreCode to *NULL*.
3. Insert a row to Minion.BackupSettings for [DB2], establishing all appropriate settings, and setting DBPreCode to *NULL*.
4. Insert a row to Minion.BackupSettings for [DB3], establishing all appropriate settings, and setting DBPreCode to *NULL*.

Note: The Minion.BackupSettings columns DBPreCode and DBPostCode are in effect whether you are using table based scheduling – that is, running Minion.BackupMaster without parameters – or using parameter based scheduling. (This is not the case for batch precode and postcode, which the next section covers.)

Batch precode and postcode

Batch precode and postcode run before and after an entire backup operation.

To run code before or after a backup batch, update (or insert) the appropriate row in Minion.BackupSettingsServer. In that row, populate the BatchPreCode column to run code before the backup operation; and populate the column BatchPostCode to run code after the backup operation. For example:

```

UPDATE Minion.BackupSettingsServer
SET      BatchPreCode = 'EXEC master.dbo.BackupPrep;',
        BatchPostCode = 'EXEC master.dbo.BackupCleanup;'
WHERE DBType = 'User'
        AND BackupType = 'Full'
        AND Day = 'Saturday';

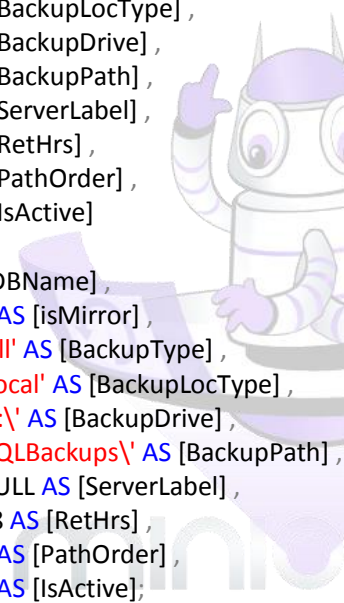
```

IMPORTANT: The Minion.BackupSettingServer columns BatchPreCode and BatchPostCode are *only* in effect for table based scheduling – that is, running Minion.BackupMaster without parameters. If you use parameter based scheduling, the only way to enact batch precode or batch postcode is with additional job steps.

How To: Configure backup file retention

Minion Backup deletes old backup files based on configured settings. Set the backup retention in hours in the Minion.BackupSettingsPath table, using the **RetHrs** (“retention in hours”) field. You can either modify the default “MinionDefault” row, or insert your own database-specific entry:

```
INSERT INTO Minion.BackupSettingsPath
    ([DBName] ,
     [isMirror] ,
     [BackupType] ,
     [BackupLocType] ,
     [BackupDrive] ,
     [BackupPath] ,
     [ServerLabel] ,
     [RetHrs] ,
     [PathOrder] ,
     [IsActive]
    )
SELECT 'DB1' AS [DBName] ,
       0 AS [isMirror] ,
       'All' AS [BackupType] ,
       'Local' AS [BackupLocType] ,
       'C:\' AS [BackupDrive] ,
       'SQLBackups\' AS [BackupPath] ,
       NULL AS [ServerLabel] ,
       48 AS [RetHrs] ,
       0 AS [PathOrder] ,
       1 AS [IsActive];
```



Note: This new RetHrs value does not affect the retention period of existing backup files.

For more information, see [“About: Backup file retention”](#).



Minion Enterprise Hint

Minion Enterprise comes with a suite of queries to pull valuable information. For example, you can easily query to find out how much space is saved when you set backup retention to two days instead of four; or how much space backups take up per server. And this information is available not just for one server, but for your entire enterprise.

See www.MinionWare.net for more information, or email us today at Support@MidnightDBA.com for a demo!

“How To” Topics: Backup Mirrors and File Actions

How to: Set up mirror backups

SQL Server Enterprise edition allows you to back up to two locations simultaneously: the primary location and the mirror location. This is not the same as striping a backup (where a single backup media set is placed across several locations); a mirrored backup creates two independent backup media sets.

To configure mirrored backups:

- Enable mirrored backups in `Minion.BackupSettings`, using the `Mirror` field.
- Configure a backup mirror path in `Minion.BackupSettingsPath`, being sure to set `isMirror = 1`.

For example, to mirror full backups for database DB8, first enable mirrored backups for that database and backup type. We will insert one row for DB8, `BackupType=Full`; and one row for DB8, `BackupType=All`, to provide settings for DB8 diff and log backups (as explained in “[The Configuration Settings Hierarchy Rule](#)”).

-- DB8 BackupType='All', to cover log and differential settings.

```
INSERT INTO Minion.BackupSettings
    ([DBName] ,
     [Port] ,
     [BackupType] ,
     [Exclude] ,
     [GroupOrder] ,
     [GroupDBOrder] ,
     [Mirror] ,
     [DelFileBefore] ,
     [DelFileBeforeAgree] ,
     [LogLoc] ,
     [HistRetDays] ,
     [DynamicTuning] ,
```

```

        [Verify] ,
        [ShrinkLogOnLogBackup] ,
        [MinSizeForDiffInGB] ,
        [DiffReplaceAction] ,
        [Encrypt] ,
        [Checksum] ,
        [Init] ,
        [Format] ,
        [IsActive] ,
        [Comment]
    )
SELECT 'DB8' AS [DBName] ,
    NULL AS [Port] ,
    'All' AS [BackupType] ,
    0 AS [Exclude] ,
    50 AS [GroupOrder] ,
    0 AS [GroupDBOrder] ,
    0 AS [Mirror] , -- Disable mirrored log/diff backups for DB8
    0 AS [DelFileBefore] ,
    0 AS [DelFileBeforeAgree] ,
    'Local' AS [LogLoc] ,
    90 AS [HistRetDays] ,
    1 AS [DynamicTuning] ,
    '0' AS [Verify] ,
    1 AS [ShrinkLogOnLogBackup] ,
    20 AS [MinSizeForDiffInGB] ,
    'Log' AS [DiffReplaceAction] ,
    0 AS [Encrypt] ,
    1 AS [Checksum] ,
    1 AS [Init] ,
    1 AS [Format] ,
    1 AS [IsActive] ,
    NULL AS [Comment];

```

-- DB8 BackupType='Full'; enable full mirrored backups.

```

INSERT INTO Minion.BackupSettings
    ( [DBName] ,
      [Port] ,
      [BackupType] ,
      [Exclude] ,
      [GroupOrder] ,
      [GroupDBOrder] ,
      [Mirror] ,
      [DelFileBefore] ,
      [DelFileBeforeAgree] ,
      [LogLoc] ,
      [HistRetDays] ,
      [DynamicTuning] ,
      [Verify] ,
      [ShrinkLogOnLogBackup] ,

```

```

        [MinSizeForDiffInGB] ,
        [DiffReplaceAction] ,
        [Encrypt] ,
        [Checksum] ,
        [Init] ,
        [Format] ,
        [IsActive] ,
        [Comment]
    )
SELECT 'DB8' AS [DBName] ,
    NULL AS [Port] ,
    'Full' AS [BackupType] ,
    0 AS [Exclude] ,
    50 AS [GroupOrder] ,
    0 AS [GroupDBOrder] ,
    1 AS [Mirror] , -- Enable mirrored full backups for DB8
    0 AS [DelFileBefore] ,
    0 AS [DelFileBeforeAgree] ,
    'Local' AS [LogLoc] ,
    90 AS [HistRetDays] ,
    1 AS [DynamicTuning] ,
    '0' AS [Verify] ,
    1 AS [ShrinkLogOnLogBackup] ,
    20 AS [MinSizeForDiffInGB] ,
    'Log' AS [DiffReplaceAction] ,
    0 AS [Encrypt] ,
    1 AS [Checksum] ,
    1 AS [Init] ,
    1 AS [Format] ,
    1 AS [IsActive] ,
    NULL AS [Comment];

```



Next, we configure a primary backup path in Minion.BackupSettingsPath for DB8. For this particular server, we would like all mirrored backups to go to “M:\SQLMirrorBackups\”. So, we can simply implement a new MinionDefault row where isMirror=1:

```

INSERT INTO Minion.BackupSettingsPath
    ( [DBName] ,
      [IsMirror] ,
      [BackupType] ,
      [BackupLocType] ,
      [BackupDrive] ,
      [BackupPath] ,
      [ServerLabel] ,
      [RetHrs] ,
      [PathOrder] ,
      [IsActive] ,
      [AzureCredential] ,
      [Comment]
    )

```

```

SELECT 'MinionDefault' AS [DBName] ,
       1 AS [IsMirror] ,
       'All' AS [BackupType] ,
       'Local' AS [BackupLocType] ,
       'M:\' AS [BackupDrive] ,
       'SQLMirrorBackups\' AS [BackupPath] ,
       NULL AS [ServerLabel] ,
       24 AS [RetHrs] ,
       0 AS [PathOrder] ,
       1 AS [IsActive] ,
       NULL AS [AzureCredential] ,
       'MinionDefault mirror row.' AS [Comment];

```

Note: If we did not want all mirrored backups going to the same location, we could just as easily have defined the Minion.BackupSettingsPath with DBName='DB1'.

Once these two steps are done, all full backups for DB8 will be mirrored backups, with the mirror backup files going to M:\SQLMirrorBackups\. Minion Backup will manage these mirrored backup files just like the primary files, deleting them once they have exceeded the retention period.

IMPORTANT: Mirrored backups are only supported in SQL Server Enterprise edition.

How to: Copy files after backup (single and multiple locations)

As part of your backup routine, you can choose to copy your backup files to multiple locations, move your backup files to a location, or both. This section will walk you through the steps of setting up file copy operations. For more information, see the section [“About: Copy and move backup files”](#).

Note: Currently, Minion Backup can't copy or move files to or from Microsoft Azure Blobs. However, you can do a primary backup to an Azure Blob.

The basic steps to configure copy operations for backup files are:

1. **Set the FileAction and FileActionTime fields in Minion.BackupSettings, for the appropriate database(s) and backup type(s).**
2. **Insert one row per operation into the Minion.BackupSettingsPath table.**

Note: If you specify one database-specific setting in the Minion.BackupSettings table, you must be sure that all backup types are covered for that database. For example: one row for Full backups, and one row with BackupType='All' to cover differential and log backups. The same rule exists for Minion.BackupSettingsPath. For more information, see the FAQ section [“Why must I supply values for all backup types for a database in the settings tables?”](#)

So for example, we can configure Minion Backup to copy the YourDatabase full backup file to two secondary locations. First, **set the FileAction and FileActionTime fields in Minion.BackupSettings, for the appropriate database(s) and backup type(s):**

1. **Insert a row for YourDatabase into Minion.BackupSettings, in order to enable the file action (“COPY”) and file action time (in this example, “AfterBatch”).** Settings for the “YourDatabase” full backup row should include, in part:
 - a. DBName = ‘YourDatabase’
 - b. BackupType = ‘Full’
 - c. FileAction = ‘Copy’
 - d. FileActionTime = ‘AfterBatch’
2. **Insert a BackupType=‘All’ row into the Minion.BackupSettings table, to cover differential and log backup operations.** As we don’t wish to copy log or differential backups in this example, the settings for this row (DBName= “YourDatabase”, BackupType=“All”) should include, in part:
 - a. DBName = ‘YourDatabase’
 - b. BackupType = ‘All’
 - c. FileAction = NULL
 - d. FileActionTime = NULL

Note: The simplest way to insert a row to a table is to use the Minion.CloneSettings procedure to generate an insert statement for that table, modify the statement to reflect the proper database and specifications, and run it.

So, the contents of the Minion.BackupSettings table would look like this (some columns omitted for brevity):

DBName	BackupType	FileAction	FileActionTime	Comment
MinionDefault	All	NULL	NULL	Minion default. DO NOT REMOVE.
YourDatabase	Full	Copy	AfterBatch	YourDatabase database full backup.
YourDatabase	All	NULL	NULL	YourDatabase database - all backups.

Second, insert one row per operation into the Minion.BackupSettingsPath table:

1. **Insert a BackupType=‘Full’ row into the Minion.BackupSettingsPath table, for the full backup operation.** Settings for the “YourDatabase” row should include, in part:
 - a. DBName = ‘YourDatabase’
 - b. BackupType = ‘Full’
 - c. BackupDrive = the name of your backup drive (e.g., ‘C:\’)
 - d. BackupPath = the full backup path, without the drive letter (e.g., ‘SQLBackups\’)
 - e. FileActionMethod=NULL
2. **Insert a BackupType=‘All’ row into the Minion.BackupSettingsPath table, to cover differential and log backup operations.** (The reason for this is, whenever you specify a database-specific setting in Minion.BackupSettingsPath, all three basic backup types must be represented, one way or another.) Settings for the “YourDatabase” row should include, in part:
 - a. DBName = ‘YourDatabase’
 - b. BackupType = ‘All’
 - c. BackupDrive = the name of your backup drive (e.g., ‘C:\’)

- d. BackupPath = the full backup path, without the drive letter (e.g., 'SQLBackups\')
 - e. FileActionMethod=NULL
3. **Insert a BackupType='Copy' row into the Minion.BackupSettingsPath table, for the first copy operation.** Settings for the "YourDatabase" row should include, in part:
- a. DBName = 'YourDatabase'
 - b. BackupType = 'Copy'
 - c. BackupDrive = the name of your first copy drive (e.g., 'F:\')
 - d. BackupPath = the full backup path, without the drive letter (e.g., 'BackupCopies\')
 - e. FileActionMethod='XCOPY' (*Optional: see the note below for information about this field.*)
4. **Insert a BackupType='Copy' row into the Minion.BackupSettingsPath table, for the second copy operation.** Settings for the "YourDatabase" row should include, in part:
- a. DBName = 'YourDatabase'
 - b. BackupType = 'Copy'
 - c. BackupDrive = the name of your first copy drive (e.g., 'Y:\')
 - d. BackupPath = the full backup path, without the drive letter (e.g., 'MoreBackupCopies\')
 - e. FileActionMethod='XCOPY' (*Optional: see the note below for information about this field.*)

Note: Minion Backup lets you choose what program you use to do your file copy and move operations. So, the FileActionMethod field in Minion.BackupSettings has several valid inputs: NULL (same as COPY), COPY, MOVE, XCOPY, ROBOCOPY, ESEUTIL. Note that "COPY" and "MOVE" use PowerShell COPY or MOVE commands as needed.

So the contents of the Minion.BackupSettingsPath table would look like this (some columns omitted for brevity):

DBName	BackupType	BackupDrive	BackupPath	FileActionMethod	Comment
MinionDefault	All	C:\	SQLBackups\	NULL	Minion default. DO NOT REMOVE.
YourDatabase	Full	C:\	SQLBackups\	NULL	YourDatabase database full backup.
YourDatabase	All	C:\	SQLBackups\	NULL	YourDatabase database - all backups.
YourDatabase	Copy	F:\	BackupCopies\	XCOPY	YourDatabase database full backup copy #1.
YourDatabase	Copy	Y:\	MoreBackupCopies\	XCOPY	YourDatabase database full backup copy #2.

Note: You can view a log of copy and move operations in the Minion.BackupFiles table.

How to: Move files to a location after backup

As part of your backup routine, you can choose to copy your backup files to multiple locations, move your backup files to a location, or both. This section will walk you through the steps of setting up a file move operation. For more information, see the section [“About: Copy and move backup files”](#).

Note: Currently, Minion Backup can't copy or move files to or from Microsoft Azure Blobs. However, you can do a primary backup to an Azure Blob.

The basic steps to configure move operations for backup files are:

1. **Set the FileAction and FileActionTime fields in Minion.BackupSettings, for the appropriate database(s) and backup type(s).**
2. **Insert one row per operation into the Minion.BackupSettingsPath table.**

Note: If you specify one database-specific setting in the Minion.BackupSettings table, you must be sure that all backup types are covered for that database. For example: one row for Full backups, and one row with BackupType='All' to cover differential and log backups. The same rule exists for Minion.BackupSettingsPath. For more information, see the FAQ section [“Why must I supply values for all backup types for a database in the settings tables?”](#)

So for example, we can configure Minion Backup to move the YourDatabase full backup file to one secondary location. (You cannot move the backup file to more than one location; after the first move, the file will no longer be in the original location!) First, **set the FileAction and FileActionTime fields in**

Minion.BackupSettings, for the appropriate database(s) and backup type(s):

1. **Insert a row for YourDatabase into Minion.BackupSettings, in order to enable the file action (“MOVE”) and file action time (in this example, “AfterBackup”).** Settings for the “YourDatabase” full backup row should include, in part:
 - a. DBName = 'YourDatabase'
 - b. BackupType = 'Full'
 - c. FileAction = 'Move'
 - d. FileActionTime = 'AfterBackup'
2. **Insert a BackupType='All' row into the Minion.BackupSettings table, to cover differential and log backup operations.** As we don't wish to move log or differential backups in this example, the settings for this row (DBName= “YourDatabase”, BackupType=“All”) should include, in part:
 - a. DBName = 'YourDatabase'
 - b. BackupType = 'All'
 - c. FileAction = NULL
 - d. FileActionTime = NULL

Note: The simplest way to insert a row to a table is to use the `Minion.CloneSettings` procedure to generate an insert statement for that table, modify the statement to reflect the proper database and specifications, and run it.

So the contents of the `Minion.BackupSettings` table would look like this (some columns omitted for brevity):

DBName	BackupType	FileAction	FileActionTime	Comment
MinionDefault	All	NULL	NULL	Minion default. DO NOT REMOVE.
YourDatabase	Full	Move	AfterBackup	YourDatabase database full backup.
YourDatabase	All	NULL	NULL	YourDatabase database - all backups.

Second, insert one row per operation into the `Minion.BackupSettingsPath` table:

1. **Insert a BackupType='Full' row into the Minion.BackupSettingsPath table, for the full backup operation.** Settings for the "YourDatabase" row should include, in part:
 - a. DBName = 'YourDatabase'
 - b. BackupType = 'Full'
 - c. BackupDrive = the name of your backup drive (e.g., 'C:\')
 - d. BackupPath = the full backup path, without the drive letter (e.g., 'SQLBackups\')
 - e. FileActionMethod=NULL
2. **Insert a BackupType='All' row into the Minion.BackupSettingsPath table, to cover differential and log backup operations.** (The reason for this is, whenever you specify a database-specific setting in `Minion.BackupSettingsPath`, all three basic backup types must be represented, one way or another.) Settings for the "YourDatabase" row should include, in part:
 - a. DBName = 'YourDatabase'
 - b. BackupType = 'All'
 - c. BackupDrive = the name of your backup drive (e.g., 'C:\')
 - d. BackupPath = the full backup path, without the drive letter (e.g., 'SQLBackups\')
 - e. FileActionMethod=NULL
3. **Insert a BackupType='Move' row into the Minion.BackupSettingsPath table, for the move operation.** Settings for the "YourDatabase" row should include, in part:
 - a. DBName = 'YourDatabase'
 - b. BackupType = 'Move'
 - c. BackupDrive = the name of your first copy drive (e.g., 'X:\')
 - d. BackupPath = the full backup path, without the drive letter (e.g., 'MovedBackups\')
 - e. FileActionMethod='ROBOCOPY' (Optional: see the note below for information about this field).

Note: Minion Backup lets you choose what program you use to do your file copy and move operations. So, the `FileActionMethod` field in `Minion.BackupSettings` has several valid inputs: NULL (same as COPY), COPY, MOVE, XCOPY, ROBOCOPY, ESEUTIL. Note that "COPY" and "MOVE" use PowerShell COPY or MOVE commands as needed.

So, the contents of the Minion.BackupSettingsPath table would look like this (some columns omitted for brevity):

DBName	BackupType	BackupDrive	BackupPath	FileActionMethod	Comment
MinionDefault	All	C:\	SQLBackups\	NULL	Minion default. DO NOT REMOVE.
YourDatabase	Full	C:\	SQLBackups\	NULL	YourDatabase database full backup.
YourDatabase	All	C:\	SQLBackups\	NULL	YourDatabase database - all backups.
YourDatabase	Move	X:\	MovedBackups\	ROBOCOPY	YourDatabase database full backup move.

How to: Copy and move backup files

As part of your backup routine, you can choose to copy your backup files to multiple locations, move your backup files to a location, or both. This section will walk you through the steps of setting up a file copy and move operation. For more information, see the section "[About: Copy and move backup files](#)".

Note: Currently, Minion Backup can't copy or move files to or from Microsoft Azure Blobs. However, you can do a primary backup to an Azure Blob.

The basic steps to configure move operations for backup files are:

1. Set the **FileAction** and **FileActionTime** fields in **Minion.BackupSettings**, for the appropriate database(s) and backup type(s).
2. Insert one row per operation into the **Minion.BackupSettingsPath** table.

The two sections above – "[How to: Copy files to a location after backup \(single and multiple locations\)](#)" and "[How to: Move files to a location after backup](#)" – detail the setup for copy and move operations. The only difference for a scenario where you wish to copy *and* move a backup is that **the FileAction field in Minion.BackupSettings must be set to "MoveCopy" (instead of MOVE or COPY).**

How to: Back up to multiple files in a single location

Minion Backup allows you to back up to multiple files. You can configure multi-file backups in just two steps:

1. Configure the number of backup files in the **Minion.BackupTuningThresholds** table.
2. Configure the backup location in the **Minion.BackupSettingsPath** table.

When this is configured, backups will proceed as defined: a database will back up to multiple files.

Let us take the example of backing up the DB1 database to four files, for full backups and differential backups.

First, configure the number of backup files in Minion.BackupTuningThresholds. Log backups in this example will be backed up to one file, while full and differential backups will be backed up to four. We can configure this with two rows – one for BackupType=Log and NumberOfFiles=1, and one for BackupType=All and NumberOfFiles=4:

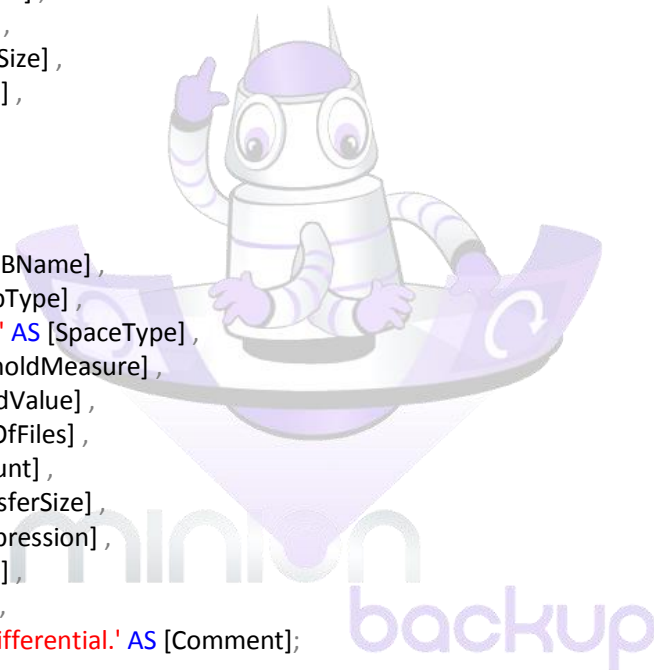
```
INSERT INTO Minion.BackupTuningThresholds
```

```
( [DBName] ,  
  [BackupType] ,  
  [SpaceType] ,  
  [ThresholdMeasure] ,  
  [ThresholdValue] ,  
  [NumberOfFiles] ,  
  [Buffercount] ,  
  [MaxTransferSize] ,  
  [Compression] ,  
  [BlockSize] ,  
  [IsActive] ,  
  [Comment]  
)
```

```
SELECT 'DB1' AS [DBName] ,  
       'All' AS [BackupType] ,  
       'DataAndIndex' AS [SpaceType] ,  
       'GB' AS [ThresholdMeasure] ,  
       0 AS [ThresholdValue] ,  
       4 AS [NumberOfFiles] ,  
       0 AS [Buffercount] ,  
       0 AS [MaxTransferSize] ,  
       NULL AS [Compression] ,  
       0 AS [BlockSize] ,  
       1 AS [IsActive] ,  
       'DB1 full and differential.' AS [Comment];
```

```
INSERT INTO Minion.BackupTuningThresholds
```

```
( [DBName] ,  
  [BackupType] ,  
  [SpaceType] ,  
  [ThresholdMeasure] ,  
  [ThresholdValue] ,  
  [NumberOfFiles] ,  
  [Buffercount] ,  
  [MaxTransferSize] ,  
  [Compression] ,  
  [BlockSize] ,  
  [IsActive] ,  
  [Comment]  
)
```



```

SELECT 'DB1' AS [DBName] ,
      'Log' AS [BackupType] ,
      'DataAndIndex' AS [SpaceType] ,
      'GB' AS [ThresholdMeasure] ,
      0 AS [ThresholdValue] ,
      1 AS [NumberOfFiles] ,
      0 AS [Buffercount] ,
      0 AS [MaxTransferSize] ,
      NULL AS [Compression] ,
      0 AS [BlockSize] ,
      1 AS [IsActive] ,
      'DB1 log.' AS [Comment];

```

Note that the code above omits BeginTime, EndTime, and DayOfWeek. These fields are optional; they may be used to limit the days and times at which the threshold in question applies. As we want these new threshold settings to apply at all time, we can comfortably leave these three fields NULL.

Next, configure the backup location. Determine whether the default location in Minion.BackupSettingsPath (as configured in the row where DBName='MinionDefault' and BackupType='All') is correct for your backups. For this example, we will say that the default location is not correct. So, we will insert a new row to configure the new path:

```

INSERT INTO Minion.BackupSettingsPath
( [DBName] ,
  [IsMirror] ,
  [BackupType] ,
  [BackupLocType] ,
  [BackupDrive] ,
  [BackupPath] ,
  [ServerLabel] ,
  [RetHrs] ,
  [FileActionMethod] ,
  [FileActionMethodFlags] ,
  [PathOrder] ,
  [IsActive] ,
  [AzureCredential] ,
  [Comment]
)

```

```

SELECT 'DB1' AS [DBName] ,
      0 AS [IsMirror] ,
      'All' AS [BackupType] ,
      'Local' AS [BackupLocType] ,
      'E:\' AS [BackupDrive] ,
      'SQLBackups\' AS [BackupPath] ,
      NULL AS [ServerLabel] ,
      24 AS [RetHrs] ,
      NULL AS [FileActionMethod] ,
      NULL AS [FileActionMethodFlags] ,
      0 AS [PathOrder] ,

```

```
1 AS [IsActive] ,  
NULL AS [AzureCredential] ,  
'DB1 location.' AS [Comment];
```

Once the files and paths are configured, the DB1 backups will be placed as follows:

- DB1 full (or differential) backups will stripe to four files on the DB1 location.
- DB1 log backups have only one file defined, so Minion Backup backs up the DB1 log to one file on the DB1 location.

The use of the Minion.BackupTuningThresholds table is detailed much more thoroughly in the [“How to: Set up dynamic backup tuning thresholds”](#) section, and in the [“Minion.BackupTuningThresholds”](#) section.

And for more information on backup paths, see [“Minion.BackupSettingsPath”](#).

How to: Back up to multiple locations

Minion Backup allows you to back up to multiple files, and to back those files up to multiple locations. You can configure multi-location backups in just two steps:

1. Configure the number of backup files in the Minion.BackupTuningThresholds table.
2. Configure the backup locations in the Minion.BackupSettingsPath table.

When this is configured, backups will proceed as defined; multiple backup files will be placed on multiple paths in a round robin fashion.

Let us take the example of backing up the DB1 database to four files on two separate drives, for full backups and differential backups.

First, configure the number of backup files in Minion.BackupTuningThresholds. Log backups in this example will be backed up to one file, while full and differential backups will be backed up to four. We can configure this with two rows – one for BackupType=Log and NumberOfFiles=1, and one for BackupType=All and NumberOfFiles=4:

```
INSERT INTO Minion.BackupTuningThresholds  
( [DBName] ,  
  [BackupType] ,  
  [SpaceType] ,  
  [ThresholdMeasure] ,  
  [ThresholdValue] ,  
  [NumberOfFiles] ,  
  [Buffercount] ,  
  [MaxTransferSize] ,  
  [Compression] ,  
  [BlockSize] ,  
  [IsActive] ,  
  [Comment]
```

```

)
SELECT 'DB1' AS [DBName] ,
      'All' AS [BackupType] ,
      'DataAndIndex' AS [SpaceType] ,
      'GB' AS [ThresholdMeasure] ,
      0 AS [ThresholdValue] ,
      4 AS [NumberOfFiles] ,
      0 AS [Buffercount] ,
      0 AS [MaxTransferSize] ,
      NULL AS [Compression] ,
      0 AS [BlockSize] ,
      1 AS [IsActive] ,
      'DB1 full and differential.' AS [Comment];

```

```

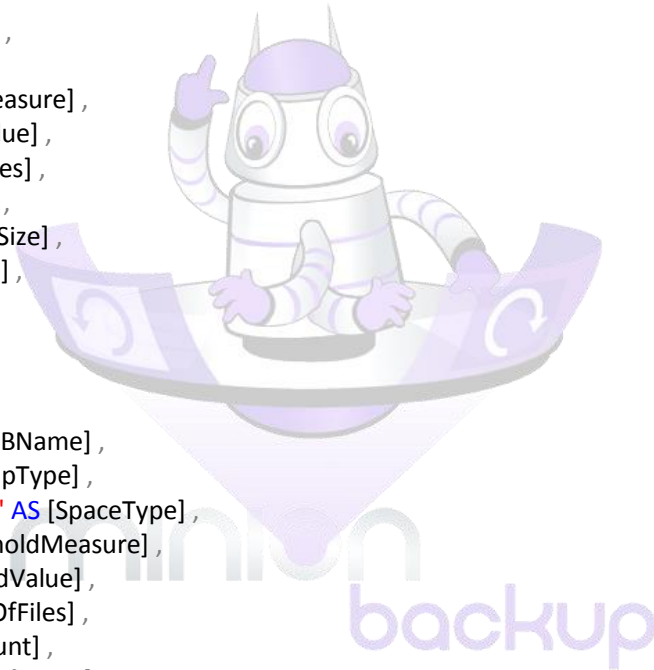
INSERT INTO Minion.BackupTuningThresholds
( [DBName] ,
  [BackupType] ,
  [SpaceType] ,
  [ThresholdMeasure] ,
  [ThresholdValue] ,
  [NumberOfFiles] ,
  [Buffercount] ,
  [MaxTransferSize] ,
  [Compression] ,
  [BlockSize] ,
  [IsActive] ,
  [Comment]
)

```

```

SELECT 'DB1' AS [DBName] ,
      'Log' AS [BackupType] ,
      'DataAndIndex' AS [SpaceType] ,
      'GB' AS [ThresholdMeasure] ,
      0 AS [ThresholdValue] ,
      1 AS [NumberOfFiles] ,
      0 AS [Buffercount] ,
      0 AS [MaxTransferSize] ,
      NULL AS [Compression] ,
      0 AS [BlockSize] ,
      1 AS [IsActive] ,
      'DB1 log.' AS [Comment];

```



Note that the code above omits BeginTime, EndTime, and DayOfWeek. These fields are optional; they may be used to limit the days and times at which the threshold in question applies. As we want these new threshold settings to apply at all time, we can comfortably leave these three fields NULL.

Next, configure the backup locations. We can define multiple backup paths for DB1, and additionally, order the paths (using the PathOrder field) to determine which path will be use first. In this example, we will use two rows to configure two paths:

```
INSERT INTO Minion.BackupSettingsPath
```

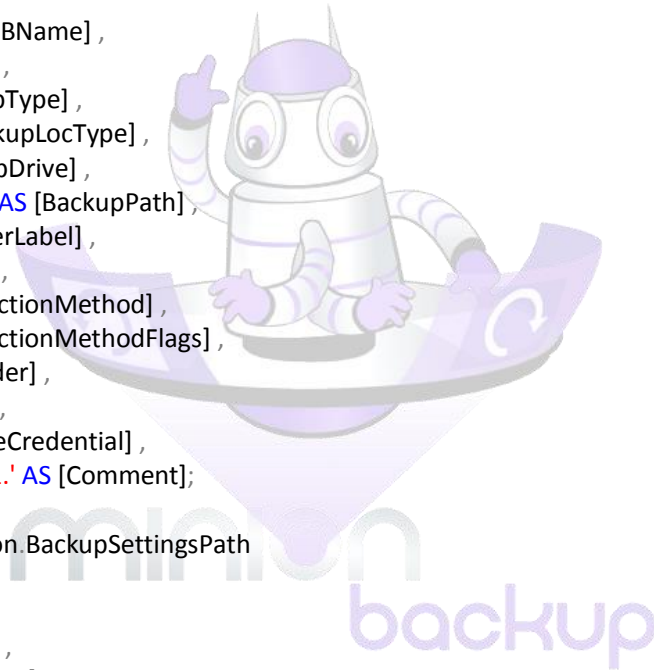
```
( [DBName] ,  
  [IsMirror] ,  
  [BackupType] ,  
  [BackupLocType] ,  
  [BackupDrive] ,  
  [BackupPath] ,  
  [ServerLabel] ,  
  [RetHrs] ,  
  [FileActionMethod] ,  
  [FileActionMethodFlags] ,  
  [PathOrder] ,  
  [IsActive] ,  
  [AzureCredential] ,  
  [Comment]  
)
```

```
SELECT 'DB1' AS [DBName] ,  
       0 AS [IsMirror] ,  
       'All' AS [BackupType] ,  
       'Local' AS [BackupLocType] ,  
       'E:\' AS [BackupDrive] ,  
       'SQLBackups\' AS [BackupPath] ,  
       NULL AS [ServerLabel] ,  
       24 AS [RetHrs] ,  
       NULL AS [FileActionMethod] ,  
       NULL AS [FileActionMethodFlags] ,  
       50 AS [PathOrder] ,  
       1 AS [IsActive] ,  
       NULL AS [AzureCredential] ,  
       'DB1 location 1.' AS [Comment];
```

```
INSERT INTO Minion.BackupSettingsPath
```

```
( [DBName] ,  
  [IsMirror] ,  
  [BackupType] ,  
  [BackupLocType] ,  
  [BackupDrive] ,  
  [BackupPath] ,  
  [ServerLabel] ,  
  [RetHrs] ,  
  [FileActionMethod] ,  
  [FileActionMethodFlags] ,  
  [PathOrder] ,  
  [IsActive] ,  
  [AzureCredential] ,  
  [Comment]  
)
```

```
SELECT 'DB1' AS [DBName] ,  
       0 AS [IsMirror] ,  
       'All' AS [BackupType] ,
```



```
'Local' AS [BackupLocType] ,
'F:\' AS [BackupDrive] ,
'SQLBackups\' AS [BackupPath] ,
NULL AS [ServerLabel] ,
24 AS [RetHrs] ,
NULL AS [FileActionMethod] ,
NULL AS [FileActionMethodFlags] ,
10 AS [PathOrder] ,
1 AS [IsActive] ,
NULL AS [AzureCredential] ,
'DB1 location 2.' AS [Comment];
```

Note that PathOrder is a weighted measure, meaning that higher numbers means higher precedence. DB1 location 1 has PathOrder of 50, while DB1 location 2 has a PathOrder of 10; so, DB1 location 1 will be selected first.

Once the files and paths are configured, the DB1 backups will be placed as follows:

- DB1 full (or differential) backups will stripe to four files. These will be placed on the defined paths in a round robin fashion:
 - file1 is created on location 1;
 - file2 is created on location 2;
 - file3 is created on location 1; and
 - file4 is created on location 2.
- DB1 log backups have only one file defined, so Minion Backup selects the target path for DB1 that has the heaviest weight: in this case, DB1 location 1.

The use of the Minion.BackupTuningThresholds table is detailed much more thoroughly in the [“How to: Set up dynamic backup tuning thresholds”](#) section, and in the [“Minion.BackupTuningThresholds”](#) section.

And for more information on backup paths, see [“Minion.BackupSettingsPath”](#).

“How To” Topics: Advanced

How to: Install Minion Backup across multiple instances

You can install Minion Backup on a single instance, using the MinionBackup1.0.sql T-SQL script. You also have the option of using the **“MinionMassInstall.ps1”** PowerShell script (provided in the Minion Backup download) to install Minion Backup on dozens or hundreds of servers at once, just as easily as you would install it on a single instance.

IMPORTANT: The destination database must exist on each server you install Minion Backup to. Partly for this reason, we recommend installing MB to the *master* database. If you choose to install to another database (for example, a user database named “DBAdmin”), verify that the database exists on all target servers.

To use MinionMassInstall.ps1 to install Minion Backup on multiple instances:

1. Open the script for editing.
2. Alter the `$Servers` variable to reflect the list of SQL Server instances on which you would like to install Minion Backup. For example, to install MB on Server1, Server2, and Server3, the line would look like this:
`$Servers = "Server1", "Server2", "Server3"`
3. Alter the `$DBName` variable to reflect the name of the database in which you would like to install Minion Backup. For example, to install MB in the "master" database on all servers, the line would look like this:
`$DBName = "master"`
4. Make sure that the `$MinionInstallFile` variable reflects the location of the installation script. For example, "C:\MinionBackup\MinionBackup1.0.sql".
5. Save the script and execute it.

How to: Shrink log files after log backup

Minion Backup provides the option of shrinking log files after log backups.

To enable this for the database YourDatabase:

1. If it does not exist already, insert a row into Minion.BackupSettings with DBName = 'YourDatabase' and 'BackupType='All'. (Alternately, you could provide any combination of rows to cover all three types of backups – full, differential, and log – for YourDatabase.)
2. Update the row in Minion.BackupSettings for YourDatabase with the following values:
 - a. **ShrinkLogOnLogBackup** – Set this to 1, to enable the feature.
 - b. **ShrinkLogThresholdInMB** – The minimum size (in MB) the log file must be before Minion Backup will shrink it. For example, you may not want to shrink any log file under 1024MB; so set this field to 1024.
 - c. **ShrinkLogSizeInMB** – The size (in MB) the log file shrink should target.

Notes about log shrink on log backup:

- **The ShrinkLogSizeInMB field** represents how big you would like the log file to be after a file shrink. This setting applies for EACH log file, not for all log files totaled. If you specify 1024 as the size here, and you have three log files for your database, Minion Backup will attempt to shrink each of the three log files down to 1024MB (so you'll end up with at least 3072MB of logs).
- **Minion Backup also helps you monitor your VLFs.** Just before a log backup is taken, we store the number of VLFs in the Minion.BackupLogDetails table, in the "VLFs" column. This can help you troubleshoot log performance issues.
- **MB also tracks the log size before the shrink.** You can find this number in the `_` table, columns "PreBackupLogSizeInMB" and "SizeInMB".

- **The log file shrink on log backup is AG-aware.** If you back up the log on a secondary replica of an Availability Group, SQL Server is unable to shrink that file. So instead, Minion Backup will shrink the log on the AG primary. The AG will then, in its own time, shrink the log file of the replica(s).

How to: Configure certificate backups

As far as Minion Backup is concerned, there are only two types of certificates: server certificates, and database certificates. Once you enable and configure a certificate type for backup, certificates are automatically backed up with every full database backup.

To configure certificate backups:

1. **Enable the certificate backups** in the Minion.BackupCert table.
2. **Configure the certificate backup location** in the Minion.BackupSettingsPath table.

Let's walk through an example. We will first enable, then configure both server certificates and database certificates for backup to a single backup location.

First, enable the certificate backups: Insert one row for each certificate type to the Minion.BackupCert table:

```
INSERT INTO Minion.BackupCert (CertType,CertPword,BackupCert)
SELECT 'ServerCert', Minion.EncryptTxt('S00persecr1tpa55'), 1;

INSERT INTO Minion.BackupCert (CertType,CertPword,BackupCert)
SELECT 'DatabaseCert', Minion.EncryptTxt('duB15secr1tpa55'), 1;
```

Note that the password is stored encrypted, so you must use the Minion.EncryptTxt function to encrypt the password on insert.

Next, configure the certificate backup location: Insert one row per certificate type (BackupType='ServerCert', and BackupType='DatabaseCert') to the Minion.BackupSettingsPath table:

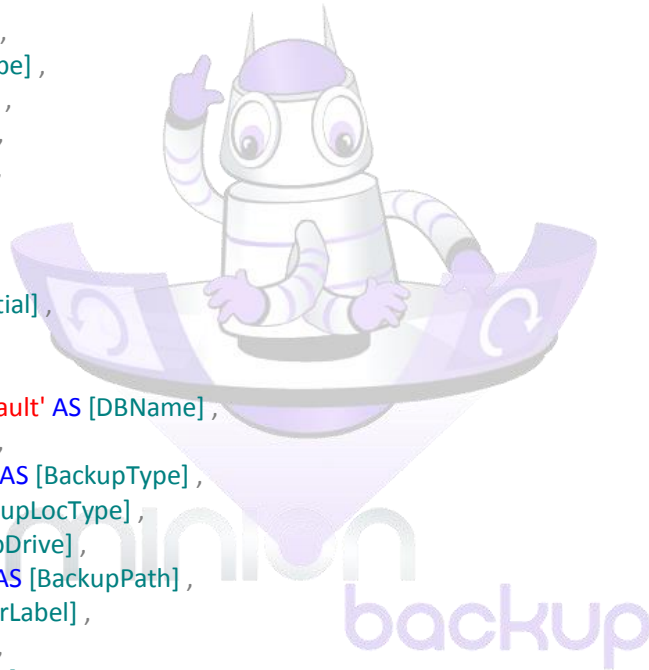
```
-- Server certificate:
INSERT INTO Minion.BackupSettingsPath
( [DBName] ,
  [isMirror] ,
  [BackupType] ,
  [BackupLocType] ,
  [BackupDrive] ,
  [BackupPath] ,
  [ServerLabel] ,
  [RetHrs] ,
  [PathOrder] ,
  [IsActive] ,
  [AzureCredential] ,
  [Comment]
)
SELECT 'MinionDefault' AS [DBName] ,
```

```

0 AS [isMirror] ,
'ServerCert' AS [BackupType] ,
'Local' AS [BackupLocType] ,
'C:\' AS [BackupDrive] ,
'SQLBackups\' AS [BackupPath] ,
NULL AS [ServerLabel] ,
24 AS [RetHrs] ,
0 AS [PathOrder] ,
1 AS [IsActive] ,
NULL AS [AzureCredential] ,
'Server certificate backup target.' AS [Comment];

-- Database certificate:
INSERT INTO Minion.BackupSettingsPath
( [DBName] ,
  [isMirror] ,
  [BackupType] ,
  [BackupLocType] ,
  [BackupDrive] ,
  [BackupPath] ,
  [ServerLabel] ,
  [RetHrs] ,
  [PathOrder] ,
  [IsActive] ,
  [AzureCredential] ,
  [Comment]
)
SELECT 'MinionDefault' AS [DBName] ,
0 AS [isMirror] ,
'DatabaseCert' AS [BackupType] ,
'Local' AS [BackupLocType] ,
'C:\' AS [BackupDrive] ,
'SQLBackups\' AS [BackupPath] ,
NULL AS [ServerLabel] ,
24 AS [RetHrs] ,
0 AS [PathOrder] ,
1 AS [IsActive] ,
NULL AS [AzureCredential] ,
'Database certificate backup target.' AS [Comment];

```



Note: For certificate backup settings paths, the database name (DBName) doesn't really apply; we use DBName='MinionDefault' here, but you could just as easily use DBName='Certificate', or any other non-null value.

How to: Encrypt backups

Starting in SQL Server 2014, you can perform backups that create encrypted backup files. To set up backup encryption in Minion Backup:

1. **Create a Database Master Key for the master database; and create a certificate** to use for backup encryption. For instructions and details, see the MSDN article on Backup Encryption: <https://msdn.microsoft.com/library/dn449489%28v=sql.120%29.aspx>
2. **Enable encryption** for one or more backups by setting Encrypt = 1 in Minion.BackupSettings.
3. **Configure encryption** by inserting one or more rows into Minion.BackupEncryption.

Note: Encrypted backups are only available in SQL Server 2014 and beyond.

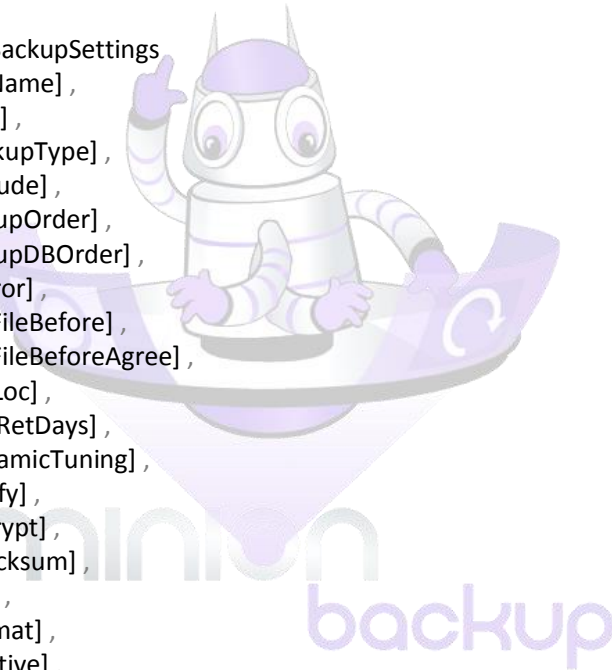
Encrypt backups for one database

First, create a Database Master Key and certificate. See the MSDN article on Backup Encryption for instructions and details: <https://msdn.microsoft.com/library/dn449489%28v=sql.120%29.aspx>

Next, enable encryption for one or more backups. In our example, we will enable encrypted backups for the DB1 database, all backup types:

```
INSERT INTO Minion.BackupSettings
  ([DBName] ,
   [Port] ,
   [BackupType] ,
   [Exclude] ,
   [GroupOrder] ,
   [GroupDBOrder] ,
   [Mirror] ,
   [DelFileBefore] ,
   [DelFileBeforeAgree] ,
   [LogLoc] ,
   [HistRetDays] ,
   [DynamicTuning] ,
   [Verify] ,
   [Encrypt] ,
   [Checksum] ,
   [Init] ,
   [Format] ,
   [IsActive] ,
   [Comment]
  )
```

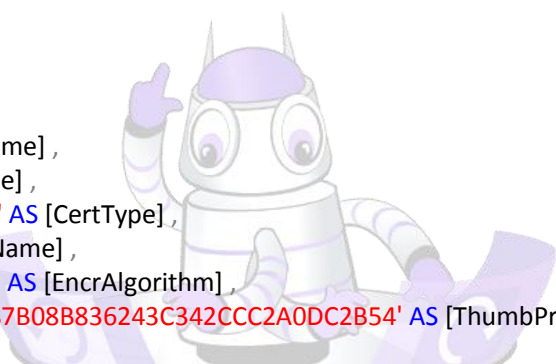
```
SELECT 'DB1' AS [DBName] ,
  NULL AS [Port] ,
  'All' AS [BackupType] ,
  0 AS [Exclude] ,
  0 AS [GroupOrder] ,
  0 AS [GroupDBOrder] ,
  0 AS [Mirror] ,
  0 AS [DelFileBefore] ,
  0 AS [DelFileBeforeAgree] ,
  'Local' AS [LogLoc] ,
  60 AS [HistRetDays] ,
  1 AS [DynamicTuning] ,
  NULL AS [Verify] ,
```



```
1 AS [Encrypt] ,
1 AS [Checksum] ,
1 AS [Init] ,
1 AS [Format] ,
1 AS [IsActive] ,
NULL AS [Comment];
```

Finally, configure encryption. In this example, we will use the same certificate for all DB1 backup types. So, insert one row into Minion.BackupEncryption:

```
INSERT INTO Minion.BackupEncryption
( [DBName] ,
  [BackupType] ,
  [CertType] ,
  [CertName] ,
  [EncrAlgorithm] ,
  [ThumbPrint] ,
  [IsActive]
)
SELECT 'DB1' AS [DBName] ,
       'All' AS [BackupType] ,
       'BackupEncryption' AS [CertType] ,
       'DB1cert' AS [CertName] ,
       'TRIPLE_DES_3KEY' AS [EncrAlgorithm] ,
       '0x63855BE98E7E87B08B836243C342CCC2A0DC2B54' AS [ThumbPrint] ,
       1 AS [IsActive];
```



Note: You can find the thumbprint and certificate name from master.sys.certificates. Check the MSDN article above for valid encryption algorithms.

You can of course use different settings for different backup types: you can use different certificates, and even different encryption algorithms for all databases and all backup types, to maximize security. You could even configure precode to change certificates and algorithms fairly easily even on a monthly basis. The choice is yours.

Encrypt backups for all databases

You also have the option to configure backup encryption for all databases very easily, as long as you don't mind using the same certificate and algorithm for all of them. Just follow the instructions for a single database, as outlined above, with the following changes:

- Instead of inserting a row to Minion.BackupSettings, update the MinionDefault / All row to enable backup encryption.
- Instead of inserting a row to Minion.BackupEncryption for a single database, insert a row for MinionDefault.

How to: Synchronize backup settings and logs among instances

Minion Backup provides a “Data Waiter” feature, which syncs backup settings and backup logs between instances of SQL Server. This is especially useful in failover situations – for example, Availability Groups, replication scenarios, or mirrored partners – so that all the latest backup settings and logs are available, regardless of which node is the primary at any given time.

Note: This feature is informally known as the *Data Waiter*, because it goes around and gives data to all of your destination tables. (*Get it?*)

The basic steps to configure the Data Waiter are:

1. **Install Minion Backup** on each destination instance.
2. **Configure the synchronization partners** in the Minion.SyncServer table.
3. **Enable the Data Waiter** for settings and/or logs, in the Minion.BackupSettingsServer table.
4. **Run the Minion.BackupSyncSettings procedure**, to prepare a snapshot of settings data.
5. **Run Minion.SyncPush** to initialize the servers.

Note: The Minion.SyncServer table itself is not synchronized across nodes; this table identifies synchronization partners – targets – and therefore the data would not be valid once moved off of the primary instance. The debug tables are also not synchronized.

IMPORTANT: There are particular considerations to keep in mind when synchronizing settings. Be sure to see the section “[About: Synchronizing settings and log data with the Data Waiter](#)”.

Example: Data Waiter serves one partner

There are several examples of two-partner scenarios. For example, you might want to sync settings and log data to a log shipping partner. That way, if you ever have to “fail over” to the log partner, you’ll already have Minion Backup installed and configured there with all the latest settings, and with a history of backups complete.

Let’s walk through an example where we want to sync our primary server’s MB settings and logs to a sync partner. The primary instance is Server1, and the target instance is Server2.

First, install Minion Backup on Server2 (the destination instance), just like you would install it on any other instance. MB is smart enough not to attempt backing up databases that are offline.

Next, configure the synchronization partners in the Minion.SyncServer table on Server1:

```
INSERT INTO Minion.SyncServer
  ([Module] ,
  [DBName] ,
  [SyncServerName] ,
  [SyncDBName] ,
  [Port] ,
  [ConnectionTimeoutInSecs]
)
```

```

SELECT 'Backup' AS [Module] ,
      'master' AS [DBName] ,      -- DB in which Minion is installed locally
      'Server2' AS [SyncServerName] , -- Name of the sync partner
      'master' AS [SyncDBName] , -- DB in which Minion is installed on the sync partner
      1433 AS [Port] ,          -- Port of the sync partner
      10 AS [ConnectionTimeoutInSecs];

```

Enable the Data Waiter for settings and/or logs, in the Minion.BackupSettingsServer table on Server1. We are not only enabling the Data Waiter, but also choosing the schedule on which we want the synchronizations to run. It's a good idea to sync logs very frequently, as MB is always adding to the log. But settings can be synchronized less frequently.

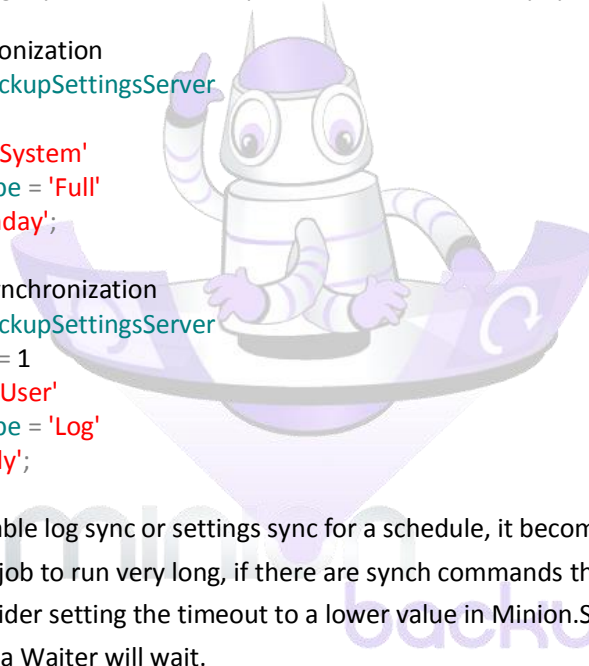
In our example, we will enable log synchronization on a frequent schedule (in this case, an hourly log backup schedule); and enable settings synch on a less frequent schedules (a weekly system database full backup):

```

-- Enable log synchronization
UPDATE Minion.BackupSettingsServer
SET   SyncLogs = 1
WHERE DBType = 'System'
      AND BackupType = 'Full'
      AND Day = 'Sunday';

-- Enable settings synchronization
UPDATE Minion.BackupSettingsServer
SET   SyncSettings = 1
WHERE DBType = 'User'
      AND BackupType = 'Log'
      AND Day = 'Daily';

```



IMPORTANT: When you enable log sync or settings sync for a schedule, it becomes possible for the Data Waiter to cause the backup job to run very long, if there are synch commands that fail (for example, due to a downed sync partner). Consider setting the timeout to a lower value in Minion.SyncServer, to limit the amount of time that the Data Waiter will wait.

Run the Minion.BackupSyncSettings procedure, to prepare a snapshot of settings data.

```
EXEC Minion.BackupSyncSettings;
```

Run Minion.SyncPush on Server1, to initialize the synch partner. This will push the current settings and the contents of the log files to the Server2 sync partner. While we could run Minion.SyncPush once (with @Tables = 'All' and @Process = 'All'), it is more efficient to run it once for logs (with @Process='All') and once for settings (with @Process='New'):

```
EXEC Minion.SyncPush
      @Tables = 'Logs'
      , @SyncServerName = NULL

```

```
, @SyncDBName = NULL  
, @Port = NULL  
, @Process = 'All'  
, @Module = 'Backup';
```

```
EXEC Minion.SyncPush  
    @Tables = 'Settings'  
, @SyncServerName = NULL  
, @SyncDBName = NULL  
, @Port = NULL  
, @Process = 'New'  
, @Module = 'Backup';
```

Note: The three middle parameters – SyncServerName, SyncDBName, and Port – should be left *NULL*, as we have already configured the target sync server in Minion.SyncServer. These parameters are used for ad hoc synchronization scenarios.

From this point forward, Minion Backup will continue to synchronize settings and log data to the Server2 synch partner. If Server2 is unavailable at any point, MB will track those entries that failed to synchronize; when the instance becomes available again, the Data Waiter will roll through the changes to bring Server2 back up to date.

Example: Data Waiter serves Availability Group members

The Data Waiter is perfectly tailored for AG scenarios. After you configure each replica as a synchronization partner, the Availability Group can fail over to any replica. Data Waiter ensures that the Minion Backup settings and logs will already be up to date on that replica when it fails over.

Let's take an example of an Availability Group where any member may become primary. The preferred replica is AG1, and secondary replicas are AG2 and AG3.

It is fairly simple to set up the Data Waiter among all nodes in the Availability Group, using the same process as outlined above. The basic steps are:

1. Install Minion Backup on all replicas. Note that MB is smart enough not to attempt to back up databases where it's not supposed to.
2. Insert a row to Minion.SyncServer on the primary server, to define a synchronization partner.
3. Set the **SyncSettings** and/or **SyncLog** bits in the Minion.BackupSettingsServer table for one or more backup types, to determine how often settings and log tables will synchronize.
4. Run the Minion.BackupSyncSettings procedure, to prepare a snapshot of settings data.
5. Run Minion.SyncPush to initialize the synchronization partners.

Let's walk through these steps in more detail.

First, install Minion Backup on AG2 and AG3 (the destination instances), just like you would install it on any other instance. MB is smart enough not to attempt backing up databases that are offline.

Configure backups normally for any databases that are not a part of the AG. For those databases that ARE members of the AG, you can do nothing at all; Minion Backup defaults to backing up AG databases on the AGPreferred replica, and will not attempt to back up an AG database that is not on the preferred server.

Next, configure the synchronization partners in the Minion.SyncServer table on AG1:

```
INSERT INTO Minion.SyncServer
  ([Module] ,
  [DBName] ,
  [SyncServerName] ,
  [SyncDBName] ,
  [Port] ,
  [ConnectionTimeoutInSecs]
)
SELECT 'Backup' AS [Module] ,
       'master' AS [DBName] ,      -- DB in which Minion is installed locally
       'AGReplica' AS [SyncServerName] , -- Automatically detects all AG replicas.
       'master' AS [SyncDBName] , -- DB in which Minion is installed on the sync partner
       1433 AS [Port] ,          -- Port of the sync partner
       10 AS [ConnectionTimeoutInSecs];
```

IMPORTANT: SyncServerName='AGReplica' causes the Data Waiter to push settings to all nodes of an Availability Group. Minion Backup is smart enough to detect all existing AG nodes. What's more, MB will add a new node that is added subsequent to this configuration. For more information on SyncServerName options, see the "[Minion.SyncServer](#)" section.

Enable the Data Waiter for settings and/or logs, in the Minion.BackupSettingsServer table on AG1. We are not only enabling the Data Waiter, but also choosing the schedule on which we want the synchronizations to run. It's a good idea to sync logs very frequently, as MB is always adding to the log. But settings can be synchronized less frequently.

In our example, we will enable log synchronization on a frequent schedule (in this case, an hourly log backup schedule); and enable settings synch on a less frequent schedules (a weekly system database full backup):

```
-- Enable log synchronization
UPDATE Minion.BackupSettingsServer
SET   SyncLogs = 1
WHERE DBType = 'System'
      AND BackupType = 'Full'
      AND Day = 'Sunday';

-- Enable settings synchronization
UPDATE Minion.BackupSettingsServer
SET   SyncSettings = 1
WHERE DBType = 'User'
      AND BackupType = 'Log'
```



```
AND Day = 'Daily';
```

IMPORTANT: When you enable log sync or settings sync for a schedule, it becomes possible for the Data Waiter to cause the backup job to run very long, if there are synch commands that fail (for example, due to a downed sync partner). Consider setting the timeout to a lower value in Minion.SyncServer, to limit the amount of time that the Data Waiter will wait.

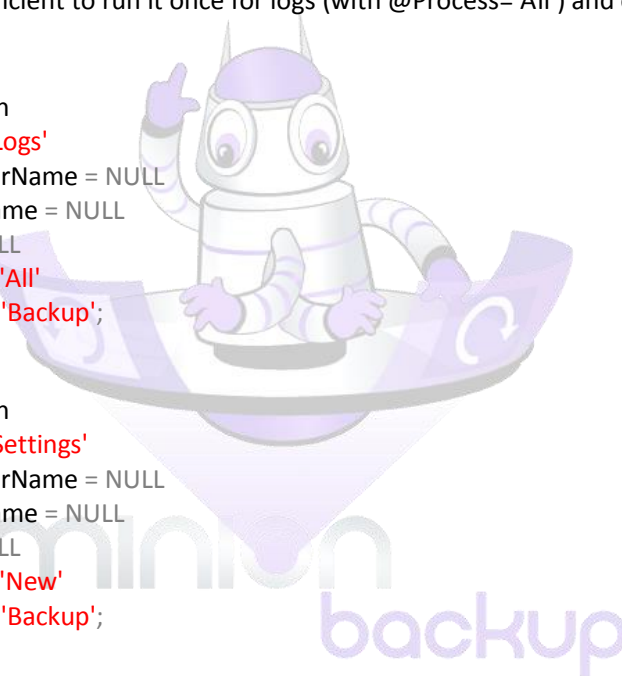
Run the Minion.BackupSyncSettings procedure, to prepare a snapshot of settings data.

```
EXEC Minion.BackupSyncSettings;
```

Run Minion.SyncPush on AG1, to initialize the servers. This will push the current settings and the contents of the log files to the AG2 sync partner. While we could run Minion.SyncPush once (with @Tables = 'All' and @Process = 'All'), it is more efficient to run it once for logs (with @Process='All') and once for settings (with @Process='New'):

```
EXEC Minion.SyncPush
    @Tables = 'Logs'
    , @SyncServerName = NULL
    , @SyncDBName = NULL
    , @Port = NULL
    , @Process = 'All'
    , @Module = 'Backup';
```

```
EXEC Minion.SyncPush
    @Tables = 'Settings'
    , @SyncServerName = NULL
    , @SyncDBName = NULL
    , @Port = NULL
    , @Process = 'New'
    , @Module = 'Backup';
```



Note: The three middle parameters – SyncServerName, SyncDBName, and Port – should be left *NULL*, as we have already configured the target sync server in Minion.SyncServer. These parameters are used for ad hoc synchronization scenarios.

From this point forward, Minion Backup will continue to synchronize settings and log data to the AG2 sync partner.

How to: Set up backups on Availability Groups

In an Availability Group (AG), you can perform backups on any node, including secondary nodes: those that are not currently the primary. In this way you can “offload” backups to conserve resources on your primary node. What’s more, an AG scenario includes the definition of a preferred server, or even a list of weighted preferences.

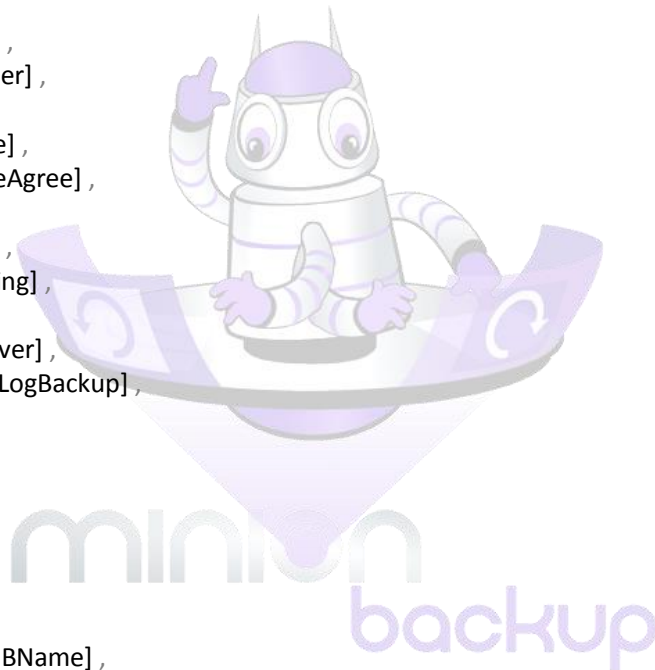
Minion Backup allows you to configure which server you would like to perform backups on in an Availability Group. You can set your backups to run on a specific server, or to run on the AG preferred server (whichever one that happens to be at the time of backup). By default, backups in Availability Groups are performed on the current primary node.

Let's take an example, where DB9 is part of an AG with two nodes. We would like DB9 full and log backups to be performed on the Server1 instance; but assign differential backups to the AG primary. We will then enter one row for DB9 / All, setting the PreferredServer column to 'Server1'; and one row for DB9 / Diff, setting PreferredServer to 'AGPreferred':

```
INSERT INTO Minion.BackupSettings
```

```
( [DBName] ,  
  [Port] ,  
  [BackupType] ,  
  [Exclude] ,  
  [GroupOrder] ,  
  [GroupDBOrder] ,  
  [Mirror] ,  
  [DelFileBefore] ,  
  [DelFileBeforeAgree] ,  
  [LogLoc] ,  
  [HistRetDays] ,  
  [DynamicTuning] ,  
  [Verify] ,  
  [PreferredServer] ,  
  [ShrinkLogOnLogBackup] ,  
  [Encrypt] ,  
  [Checksum] ,  
  [Init] ,  
  [Format] ,  
  [IsActive] ,  
  [Comment]  
)
```

```
SELECT 'DB9' AS [DBName] ,  
       NULL AS [Port] ,  
       'All' AS [BackupType] ,  
       0 AS [Exclude] ,  
       0 AS [GroupOrder] ,  
       0 AS [GroupDBOrder] ,  
       0 AS [Mirror] ,  
       0 AS [DelFileBefore] ,  
       0 AS [DelFileBeforeAgree] ,  
       'Local' AS [LogLoc] ,  
       60 AS [HistRetDays] ,  
       1 AS [DynamicTuning] ,  
       '0' AS [Verify] ,  
       'Server1' AS [PreferredServer] ,  
       0 AS [ShrinkLogOnLogBackup] ,  
       0 AS [Encrypt] ,
```



```

1 AS [Checksum] ,
1 AS [Init] ,
1 AS [Format] ,
1 AS [IsActive] ,
NULL AS [Comment];

```

```

INSERT INTO Minion.BackupSettings

```

```

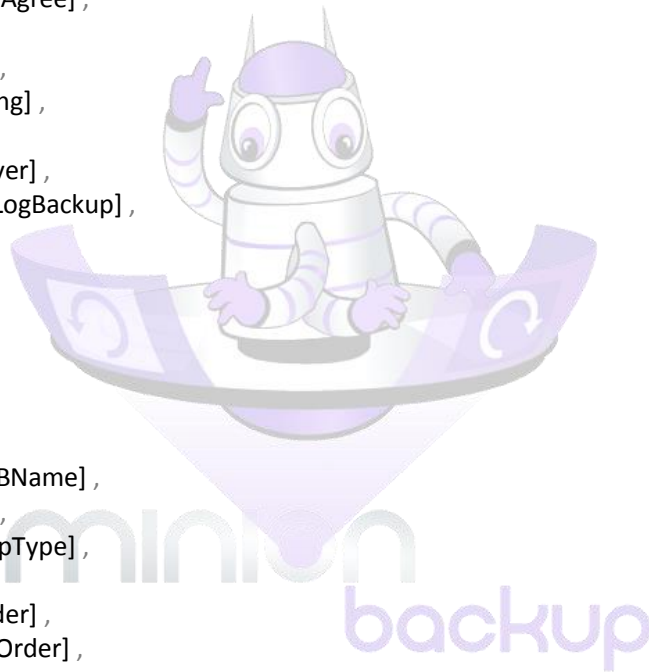
( [DBName] ,
  [Port] ,
  [BackupType] ,
  [Exclude] ,
  [GroupOrder] ,
  [GroupDBOrder] ,
  [Mirror] ,
  [DelFileBefore] ,
  [DelFileBeforeAgree] ,
  [LogLoc] ,
  [HistRetDays] ,
  [DynamicTuning] ,
  [Verify] ,
  [PreferredServer] ,
  [ShrinkLogOnLogBackup] ,
  [Encrypt] ,
  [Checksum] ,
  [Init] ,
  [Format] ,
  [IsActive] ,
  [Comment]
)

```

```

SELECT 'DB9' AS [DBName] ,
NULL AS [Port] ,
'Diff' AS [BackupType] ,
0 AS [Exclude] ,
0 AS [GroupOrder] ,
0 AS [GroupDBOrder] ,
0 AS [Mirror] ,
0 AS [DelFileBefore] ,
0 AS [DelFileBeforeAgree] ,
'Local' AS [LogLoc] ,
60 AS [HistRetDays] ,
1 AS [DynamicTuning] ,
'0' AS [Verify] ,
'AGPreferred' AS [PreferredServer] ,
0 AS [ShrinkLogOnLogBackup] ,
0 AS [Encrypt] ,
1 AS [Checksum] ,
1 AS [Init] ,
1 AS [Format] ,
1 AS [IsActive] ,
NULL AS [Comment];

```



Important notes:

- Availability groups cannot run differential backups on secondary nodes. If you accidentally specify differentials on a server and that server isn't primary, the differential backups simply won't run.
- If you use a specific server name for PreferredServer (as opposed to AGPreferred), it is enforced. In our example above, we set PreferredServer=Server1 for full and log backups. If the Server1 node is down, full and log backups will simply not run for DB9.

How to: Set up dynamic backup tuning thresholds

In SQL Server, we can adjust high level settings to improve server performance. Similarly, we can *adjust settings in individual backup statements* to improve backup performance. A backup tuning primer is beyond the scope of this document; to learn about backup tuning, please see the recording of our Backup Tuning class at <http://bit.ly/1O6Rsh3> (download demo code at <http://bit.ly/1Os6yzz>).

Once you are familiar with the backup tuning process, you can perform an analysis, and then set up specific thresholds in the Minion.BackupTuningThresholds table. It is a "Thresholds" table, because you configure a different collection of backup tuning settings for different sized databases (thereby, defining backup tuning thresholds). **As your database grows and shrinks, Minion Backup will use the settings you've defined for those sizes**, so that backups always stay at peak performance.

IMPORTANT: The "dynamic backup tuning thresholds" topic is a complicated one. We highly recommend you first read the "[About: Dynamic Backup Tuning Thresholds](#)" section before you begin.

The basic steps to set up dynamic backup tuning thresholds are:

1. Perform your backup tuning analysis for a database.
2. Enable backup tuning for that database, if it is not already enabled.
3. Enter threshold settings in Minion.BackupTuningThresholds.

The examples that follow will walk you through a few scenarios of backup tuning threshold use, and demonstrate important features of the dynamic backup tuning module.

NOTE: All of these examples are just for the sake of example; the settings we use for these examples are not recommendations and have no bearing on your particular environment. We DO NOT recommend using these numbers without proper analysis of your particular system.

Example 1: Modify existing, default tuning thresholds

Minion Backup is installed with default backup tuning threshold settings, defined by the row DBName='MinionDefault', BackupType='All', and ThresholdValue=0. You can modify the settings for all backups – assuming, of course, that no new threshold rows have been added – by updating this row. For example, to change the number of files at the default level, run a simple update statement:

```

UPDATE [Minion].BackupTuningThresholds
SET     NumberOfFiles = 2
WHERE  DBName = 'MinionDefault';

```

These default settings will apply for all databases where DynamicTuning is enabled (in Minion.BackupSettings), and that don't otherwise have tuning settings defined.

Note that the threshold you enter represents the *LOWER* threshold (the “floor”). This is why the “MinionDefault” row has a ThresholdValue of 0.

Example 2: Tune backups for one database based on file size

Now, we want to tune backups for our largest database: DB1. You have the option of basing tuning thresholds on data size only, on data and index size, or on file size. (Note that file size includes any unused space in the file; “data and index” does not.) You also have the option to tune specifically for full, differential, or log backups; or for all three (BackupType='All'). We choose to set DB1's backup tuning thresholds based on file size, for all backup types.

First, perform your backup tuning analysis. Minion Backup is a huge help to your analysis, because it gathers and records the backup settings for EVERY backup (including Buffercount, MaxTransferSize, etc.) in Minion.BackupLogDetails. These recorded settings are the actual settings that SQL Server used to take the backup, whether or not those settings were supplied by you, or chosen by SQL Server itself.

Next, enable backup tuning for DB1. In this example, backup tuning is not enabled for the MinionDefault row, and DB1 does not have an individual row. So, we must add a row for DB1 to Minion.BackupSettings.

Generate a template insert statement for DB1 using the Minion.CloneSettings procedure:

```
EXEC Minion.CloneSettings 'Minion.BackupSettings', 1;
```

Modify this generated insert statement for your database, changing the DBName to 'DB1', and setting DynamicTuning to 1:

```

INSERT INTO [Minion].BackupSettings
    ( [DBName] ,
      [Port] ,
      [BackupType] ,
      [Exclude] ,
      [GroupOrder] ,
      [GroupDBOrder] ,
      [Mirror] ,
      [LogLoc] ,
      [HistRetDays] ,
      [DynamicTuning]
    )
SELECT 'DB1' AS [DBName] ,
       1433 AS [Port] ,

```

```
'All' AS [BackupType] ,
0 AS [Exclude] ,
0 AS [GroupOrder] ,
0 AS [GroupDBOrder] ,
0 AS [Mirror] ,
'Local' AS [LogLoc] ,
60 AS [HistRetDays] ,
1 AS [DynamicTuning] ;
```

(Note that the statement above is does not include all available fields.)

The next step is to set the backup tuning thresholds, by entering rows into Minion.BackupTuningThresholds. In this example, our analysis showed that DB1 should have modest backup settings for any file size below 50GB, and slightly more aggressive settings for sizes above 50GB. So, we will enter two rows: one for file size zero to 50GB, and one for file sizes 50GB and above.

IMPORTANT: The threshold you enter represents the *LOWER* threshold (the “floor”). Therefore, you must be sure to enter a threshold for file size 0. If, for this example, we only entered a threshold for file sizes 50GB and above, Minion Backup would use the default (“MinionDefault”) row values for file sizes below 50GB; however, this behavior is only a failsafe, and we do not recommend relying on it. If you specify thresholds for a database, be sure to cover the 0 floor threshold.

The first row has a lower threshold of 0GB, and sets number of files=2, buffercount=30, and max transfer size=1mb (1048576 bytes):

```
INSERT INTO Minion.BackupTuningThresholds
( [DBName] ,
  [BackupType] ,
  [SpaceType] ,
  [ThresholdMeasure] ,
  [ThresholdValue] ,
  [NumberOfFiles] ,
  [Buffercount] ,
  [MaxTransferSize] ,
  [Compression] ,
  [BlockSize] ,
  [IsActive] ,
  [Comment]
)
SELECT 'DB1' AS [DBName] ,
       'All' AS [BackupType] ,
       'File' AS [SpaceType] , -- Tune backups by FILE size
       'GB' AS [ThresholdMeasure] ,
       0 AS [ThresholdValue] ,
       2 AS [NumberOfFiles] ,
       30 AS [Buffercount] ,
       1048576 AS [MaxTransferSize] ,
       1 AS [Compression] ,
       0 AS [BlockSize] ,
```

```

1 AS [IsActive] ,
'Lowest threshold; values above zero.' AS [Comment];

```

The second row has a lower threshold of 50GB, and sets number of files=5, buffercount=50, and max transfer size=2MB (2097152 bytes):

```

INSERT INTO Minion.BackupTuningThresholds
( [DBName] ,
  [BackupType] ,
  [SpaceType] ,
  [ThresholdMeasure] ,
  [ThresholdValue] ,
  [NumberOfFiles] ,
  [Buffercount] ,
  [MaxTransferSize] ,
  [Compression] ,
  [BlockSize] ,
  [IsActive] ,
  [Comment]
)
SELECT 'DB1' AS [DBName] ,
'All' AS [BackupType] ,
'File' AS [SpaceType] , -- Tune backups by FILE size
'GB' AS [ThresholdMeasure] ,
50 AS [ThresholdValue] ,
5 AS [NumberOfFiles] ,
50 AS [Buffercount] ,
2097152 AS [MaxTransferSize] ,
1 AS [Compression] ,
0 AS [BlockSize] ,
1 AS [IsActive] ,
'Higher threshold; values above 50GB.' AS [Comment];

```

Note that these rows are for BackupType = 'All'. If we wished to, we could instead tune different kinds of backups for DB1 separately from one another. In that case, we would have one or more rows each for DB1 full, DB1 differential, and DB1 log backups.

Example 3: Tune backup types for all databases based on data + index size

On another server, we would like to have tuning thresholds not for individual databases, but for different *backup* types. And, we would like to base the thresholds on data and index size, not on file size. The steps for this are the same as before: perform the tuning analysis, then make sure tuning is enabled for the databases, and finally, create the tuning thresholds.

To make sure tuning is enabled for ALL databases on an instance, just run an update statement on Minion.BackupSettings for all rows:

```

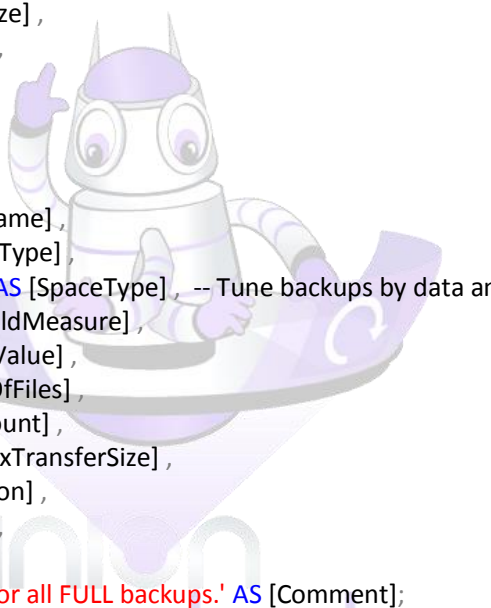
UPDATE Minion.BackupSettings

```

```
SET DynamicTuning = 1; -- Updates ALL rows
```

To set the threshold values per backup type, generate and run one insert statement for each backup type. For our first entry, we configure settings for full backups by setting DBName to “MinionDefault”, BackupType to “Full”, and SpaceType to “DataAndIndex”:

```
INSERT INTO Minion.BackupTuningThresholds
    ([DBName] ,
    [BackupType] ,
    [SpaceType] ,
    [ThresholdMeasure] ,
    [ThresholdValue] ,
    [NumberOfFiles] ,
    [Buffercount] ,
    [MaxTransferSize] ,
    [Compression] ,
    [BlockSize] ,
    [IsActive] ,
    [Comment]
)
SELECT 'MinionDefault' AS [DBName],
'Full' AS [BackupType] ,
'DataAndIndex' AS [SpaceType] , -- Tune backups by data and index size
'GB' AS [ThresholdMeasure] ,
0 AS [ThresholdValue] ,
10 AS [NumberOfFiles] ,
500 AS [Buffercount] ,
2097152 AS [MaxTransferSize] ,
1 AS [Compression] ,
0 AS [BlockSize] ,
1 AS [IsActive] ,
'Default values for all FULL backups.' AS [Comment];
```



And the row for differential backups uses BackupType = ‘Diff’:

```
INSERT INTO Minion.BackupTuningThresholds
    ([DBName] ,
    [BackupType] ,
    [SpaceType] ,
    [ThresholdMeasure] ,
    [ThresholdValue] ,
    [NumberOfFiles] ,
    [Buffercount] ,
    [MaxTransferSize] ,
    [Compression] ,
    [BlockSize] ,
    [IsActive] ,
    [Comment]
)
```



```

)
SELECT 'MinionDefault' AS [DBName] ,
      'Diff' AS [BackupType] ,
      'DataAndIndex' AS [SpaceType] , -- Tune backups by data and index size
      'GB' AS [ThresholdMeasure] ,
      0 AS [ThresholdValue] ,
      5 AS [NumberOfFiles] ,
      100 AS [Buffercount] ,
      1048576 AS [MaxTransferSize] ,
      1 AS [Compression] ,
      0 AS [BlockSize] ,
      1 AS [IsActive] ,
      'Default values for all DIFF backups.' AS [Comment];

```

And the row for log backups uses BackupType = 'Log':

```

INSERT INTO Minion.BackupTuningThresholds
  ([DBName] ,
   [BackupType] ,
   [SpaceType] ,
   [ThresholdMeasure] ,
   [ThresholdValue] ,
   [NumberOfFiles] ,
   [Buffercount] ,
   [MaxTransferSize] ,
   [Compression] ,
   [BlockSize] ,
   [IsActive] ,
   [Comment]
)
SELECT 'MinionDefault' AS [DBName] ,
      'Log' AS [BackupType] ,
      'Log' AS [SpaceType] , -- Log backups ignore this setting
      'GB' AS [ThresholdMeasure] ,
      0 AS [ThresholdValue] ,
      1 AS [NumberOfFiles] ,
      30 AS [Buffercount] ,
      1048576 AS [MaxTransferSize] ,
      1 AS [Compression] ,
      0 AS [BlockSize] ,
      1 AS [IsActive] ,
      'Default values for all LOG backups.' AS [Comment];

```

We have now configured basic tuning settings for each type of backup. Of course, we could add additional rows for each type, for different size thresholds. This is what puts the “dynamic” in “dynamic backup tuning”; Minion Backup will automatically change to the new group of settings when your database passes the defined threshold.

Moving Parts

Overview of Tables

The tables in Minion Backup fall into two categories: those that store configured **settings**, and those that **log** operational information.

The settings tables are:

- **Minion.BackupCert** – This table allows you to configure which types of certificates to back up, and the password to use when backing them up.
- **Minion.BackupEncryption** – This table stores data for each backup encryption scenario you define.
- **Minion.BackupSettings** – This table holds backup settings at the default level, database level, and backup type level. You may insert rows to define backup settings per database, per type, per type and database; or, you can rely on the system-wide default settings (defined in the “MinionDefault” row); or a combination of these.
- **Minion.BackupSettingsPath** – This table holds location configurations for each type of backup. In other words, here is where you define the paths the system will back up to.
- **Minion.BackupSettingsServer** – This table contains server-level backup settings. The backup job (*MinionBackup-AUTO*) runs regularly in conjunction with this table to provide a wide range of backup options, all without introducing additional jobs.
- **Minion.BackupTuningThresholds** – This table holds thresholds used to determine when to change the tuning of a backup; and the tuning settings per threshold.
- **Minion.DBMaintRegexLookup** – Allows you to include or exclude databases from backup (or from reindex, checkdb, or all maintenance), based off of regular expressions.
- **Minion.SyncServer** – This table allows you to define synchronization partners: instances to push settings and/or log data to.

Logs:

- **Minion.BackupFileListOnly** – A Log of RESTORE FILELISTONLY output for each backup taken
- **Minion.BackupFiles** – A log of all backup files (whether they originate from a database backup, a certificate backup, a copy, or a move). Note that a backup that is striped to 10 files will have 10 rows in this table.
- **Minion.BackupLog** – Holds a database-level summary of the backup operation per database. Each row contains the database name, operation status, the start and end time of the backup, and much more. This is updated as each backup occurs, so that you have **Live Insight** into active operations.
- **Minion.BackupLogDetails** – Holds a log of backup activity at the database level.
- **Minion.SyncCmds** – a log of commands used to synchronize settings and log tables to configured synchronization servers. This table is both a log table and a work table: the synchronization process uses Minion.SyncCmds to push the synchronization commands to target servers, and it is also a log of those commands (complete and incomplete).

- **Minion.SyncErrorCmds** – a log of synchronization commands that have failed, to be retried again later.

Settings Tables Detail

Minion.BackupCert

This table allows you to configure which types of certificates to back up, and the password to use when backing them up. As far as Minion Backup is concerned, there are only two types of certificates: ServerCert, and DatabaseCert. So, this table will only ever have two rows: one for server certificates, and one for database certificates.

Certificates that are enabled and configured for backups, are *automatically backed up with every full backup*. For more information on enabling and configuring certificate backups, see the [“How to: Configure certificate backups”](#) section.

Note: The certificate backup password is stored encrypted.

Name	Type	Description
ID	int	Primary key row identifier.
CertType	varchar	Certificate type. Valid inputs: ServerCert DatabaseCert
CertPword	varbinary	Certificate password. This is the password used to protect the certificate backup.
BackupCert	bit	Flag that determines whether or not to back up this certificate type.

Discussion:

You can back certificates up to as many locations as you like. For example, to back up server certificates to two location, insert one row for each target location into **Minion.BackupSettingsPath** with BackupType = ‘ServerCert’, and the remaining fields populated as specified in the [“How to: Configure certificate backups”](#) section.

Note that certificate entries in Minion.BackupSettingsPath do not need to populate DBName. We use DBName=‘MinionDefault’ in the examples given, but one could just as easily use DBName=‘Certificate’, DBName=‘ServerCert’, or any other non-null value. The important thing is that BackupType must be set to ‘ServerCert’ or ‘DatabaseCert’.

Minion.BackupEncryption

This table stores the certificate, encryption, and thumbprint data for each backup encryption scenario you define.

Name	Type	Description
ID	Int	Primary key row identifier.
DBName	sysname	Database name.
CertType	varchar	Certificate type. Valid inputs: BackupEncryption
CertName	varchar	Certificate name.
EncrAlgorithm	varchar	Encryption algorithm. For a list of valid inputs, see the list of key_algorithm entries in the MSDN article https://msdn.microsoft.com/en-us/library/ms189446.aspx
ThumbPrint	varbinary	A globally unique hash of the certificate. See https://msdn.microsoft.com/en-us/library/ms189774.aspx
IsActive	bit	The current row is valid (active), and should be used in the Minion Backup process.

Minion.BackupSettings

Minion.BackupSettings contains the essential backup settings, including backup order, history retention, pre- and postcode, native backup settings (like format), and more.

Minion.BackupSettings is installed with default settings already in place, via the system-wide default row (identified by DBName = "MinionDefault" and BackupType = "All"). If you do not need to fine tune your backups at all, no action is required, and all backups will use this default configuration.

Important: Do not delete the MinionDefault row, or alter the DBName or BackupType columns for this row!

To override these default settings for a specific database, insert a new row for the individual database with the desired settings. Note that any database with its own entry in Minion.BackupSettings retrieves ALL its configuration data from that row. For example, if you enter a row for [YourDatabase] and leave the ShrinkLogOnLogBackup column at NULL, Minion Backup does NOT retrieve that value from the "MinionDefault" row; in this case, ShrinkLogOnLogBackup for YourDatabase would default to off ("no").

Name	Type	Description
ID	int	Primary key row identifier.
DBName	sysname	Database name.
Port	Int	Port number for the instance. If this is NULL, we assume the port number is 1433. Minion Backup includes the port number because certain operations that are shelled out to sqlcmd require it.
BackupType	Varchar	Backup type. Valid inputs: All Full Diff Log

		Note that “All” encompasses full, differential, and log backups.
Exclude	bit	Exclude database from backups. For more on this topic, see “How To: Exclude databases from backups” .
GroupOrder	Int	The backup order within a group. Used solely for determining the order in which databases should be backed up. By default, all databases have a value of 0, which means they’ll be processed in the order they’re queried from sysobjects. Higher numbers have a greater “weight” (they have a higher priority), and will be backed up earlier than lower numbers. We recommend leaving some space between assigned back up order numbers (e.g., 10, 20, 30) so there is room to move or insert rows in the ordering. For more information, see “How To: Backup databases in a specific order” .
GroupDBOrder	int	Group to which this database belongs. Used solely for determining the order in which databases should be backed up. By default, all databases have a value of 0, which means they’ll be processed in the order they’re queried from sysobjects. Higher numbers have a greater “weight” (they have a higher priority), and will be backed up earlier than lower numbers. The range of GroupDBOrder weight numbers is 0-255. For more information, see “How To: Backup databases in a specific order” .
Mirror	Bit	Back up to a secondary mirror location. Note: This option is only available in SQL Server Enterprise edition.
DelFileBefore	bit	Delete the backup file before taking the new backup.
DelFileBeforeAgree	bit	Signifies that you know deleting the backup file first is a bad idea (because it leaves you without a backup, should your current backup fail), but that you agree anyway.

LogLoc	varchar	<p>Determines whether log data is only stored on the local (client) server, or on both the local server and the central Minion (repository) server.</p> <p>Valid inputs: Local Repo</p>
HistRetDays	smallint	<p>Number of days to retain a history of backups (in Minion Backup log tables).</p> <p>Minion Backup does not modify or delete backup information from the MSDB database.</p> <p>Note: This setting is also optionally configurable at the backup level, and also at the BackupType level. So, you can keep log history for different amounts of time for log backups than you do for full backups.</p>
MinionTriggerPath	varchar	<p>UNC path where the Minion logging trigger file is located.</p> <p>Not applicable for a standalone Minion Backup instance.</p>
DBPreCode	Nvarchar	<p>Code to run for a database, before the backup operation begins for that database.</p> <p>For more on this topic, see “How To: Run code before or after backups”.</p>
DBPostCode	nvarchar	<p>Code to run for a database, after the backup operation completes for that database.</p> <p>For more on this topic, see “How To: Run code before or after backups”.</p>
PushToMinion	Bit	<p>Save these values to the central Minion server, if it exists. Modifies values for this particular database on the central Minion server.</p> <p>A value of <i>NULL</i> indicates that this feature is off. Functionality not yet supported.</p>
DynamicTuning	bit	<p>Enables dynamic tuning.</p> <p>For more on dynamic tuning, see “How to: Set up dynamic backup tuning thresholds”.</p>
Verify	Varchar	<p>Specifies when the RESTORE VERIFYONLY operation is to happen.</p> <p>Warning: Just as with the FileActionTime column, this setting must be used with caution. Verifying backups can take a long time, and you</p>

		<p>could hold up subsequent backups while running the verify. We recommend using AfterBatch.</p> <p>(Note that the FileAction operation is processed before the Verify operation.)</p> <p>Valid inputs: <i>NULL</i> (meaning do not run verify) AfterBackup AfterBatch</p> <p>See http://msdn.microsoft.com/en-us/library/ms188902.aspx</p>
PreferredServer	Varchar	<p>The server on which you would like to perform backups in an Availability Group.</p> <p>A <i>NULL</i> in this field defaults to the current AG primary (if in an AG scenario). This field is ignored for databases not in an AG scenario.</p> <p>Valid inputs: <i>NULL</i> AGPreferred <specific server or server\instance name></p> <p>For more on this topic, see “How to: Set up backups on Availability Groups”.</p>
ShrinkLogOnLogBackup	Bit	<p>Turn on log shrink after log backups.</p> <p>For more on this topic, see “How to: Shrink log files after backup”.</p>
ShrinkLogThresholdInMB	int	<p>How big (in MB) the log file is before Minion Backup will shrink it. For example, if a log file is 1% full, but the file is only 1 GB, we probably don’t want to shrink it.</p> <p>Note that you could force a shrink after every log backup by setting this to 0, but we don’t advise it.</p> <p>For more on this topic, see “How to: Shrink log files after backup”.</p>
ShrinkLogSizeInMB	int	<p>The size (in MB) the log file shrink should target. In other words, how big you would like the log file to be after a file shrink.</p> <p>This setting applies for EACH log file, not for all log files totaled. If you specify 1024 as the size here, and you have three log files for your</p>

		<p>database, Minion Backup will attempt to shrink each of the three log files down to 1024MB (so you'll end up with at least 3072MB of logs).</p> <p>For more on this topic, see "How to: Shrink log files after backup".</p>
MinSizeForDiffInGB	bigint	<p>The minimum size of a database (in GB) in order to perform differentials; databases under this size will not get differential backups.</p> <p>A value of NULL or 0 means that there is no restriction on whether to take differential backups.</p>
DiffReplaceAction	varchar	<p>If a database does not meet the MinSizeForDiffInGB limit, perform another action instead of a differential backup (e.g., perform a log backup instead).</p> <p>While Minion Backup allows you to perform a full backup in lieu of a differential, understand that this could increase the expected time of the backup jobs.</p> <p>A <i>NULL</i> value means the same as "Skip".</p> <p>Valid inputs: Full Log Skip <i>NULL</i></p>
LogProgress	bit	<p>Track the progress of backup operations for this database.</p> <p>Status is tracked in the Minion.BackupLog table.</p>
FileAction	varchar	<p>Move or copy the backup file.</p> <p>A value of NULL means this setting has no move or copy operations.</p> <p>If COPY or MOVE is specified, at least one corresponding COPY entry (or a single corresponding MOVE entry, as appropriate) is required in the Minion.BackupSettingsPath table, to determine the path to copy or move to. IMPORTANT: If there is no corresponding COPY or MOVE entry, this setting will generate no error; there will just be no copy.</p> <p>Valid inputs: <i>NULL</i> COPY</p>

		<p>MOVE CopyMove</p> <p>For more on this topic, see “About: Copy and move backup files”.</p>
FileActionTime	Varchar	<p>The time at which to perform the COPY or MOVE FileAction.</p> <p>Valid inputs:</p> <p>AfterBackup AfterBatch</p> <p>For more on this topic, see “About: Copy and move backup files”.</p>
Encrypt	bit	Encrypt the backup.
Name	varchar	<p>The name of the backup set.</p> <p>See http://msdn.microsoft.com/en-us/library/ms186865.aspx</p>
ExpireDateInHrs	int	<p>Number of hours until the backup set for this backup can be overwritten.</p> <p>If both ExpireDateInHrs and RetainDays are both used, RetainDays takes precedence.</p> <p>See http://msdn.microsoft.com/en-us/library/ms186865.aspx</p>
RetainDays	smallint	<p>The number of days that must elapse before this backup media set can be overwritten.</p> <p>See http://msdn.microsoft.com/en-us/library/ms186865.aspx</p>
Descr	varchar	<p>Description of the backup set. Note: this must be no more than 255 characters.</p> <p>See http://msdn.microsoft.com/en-us/library/ms186865.aspx</p>
Checksum	bit	<p>Verify each page for checksum and torn page (if enabled and available) and generate a checksum for the entire backup.</p> <p>See http://msdn.microsoft.com/en-us/library/ms186865.aspx</p>
Init	bit	<p>Overwrite the existing backup set.</p> <p>See http://msdn.microsoft.com/en-us/library/ms186865.aspx</p>

Format	bit	<p>Overwrite the existing media header. Note that Format=1 is equivalent to Format=1 AND Init=1; therefore, FORMAT=1 will override the Init setting.</p> <p>See http://msdn.microsoft.com/en-us/library/ms186865.aspx</p>
CopyOnly	bit	<p>Perform a copy-only backup. Copy only backups do not affect the normal sequence of backups.</p> <p>See http://msdn.microsoft.com/en-us/library/ms186865.aspx</p>
Skip	bit	<p>Skip the check of the backup set's expiration before overwriting.</p> <p>See http://msdn.microsoft.com/en-us/library/ms186865.aspx</p>
BackupErrorMgmt	varchar	<p>Rollup of the two BACKUP flags – STOP_ON_ERROR and CONTINUE_AFTER_ERROR.</p> <p>See http://msdn.microsoft.com/en-us/library/ms186865.aspx</p>
MediaName	varchar	<p>The backup set's media name.</p> <p>See http://msdn.microsoft.com/en-us/library/ms186865.aspx</p>
MediaDescription	varchar	<p>Description of the media set. Note: this must be no more than 255 characters.</p> <p>See http://msdn.microsoft.com/en-us/library/ms186865.aspx</p>
IsActive	bit	<p>The current row is valid (active), and should be used in the Minion Backup process.</p>
Comment	Varchar	<p>For your reference only. You can label each row with a short description and/or purpose.</p>

Discussion:

The Minion.BackupSettings table comes with a row with “MinionDefault” as the DBName value, and “All” as the BackupType. This row defines the system-wide defaults.

Important: Any row inserted for an individual database overrides only ALL of the values, whether or not they are specified. Refer to the following for an example:

ID	DBName	BackupType	Exclude ...	DBPreCode
1	MinionDefault	All	0 ...	EXEC specialCode;
2	YourDatabase	Full	0 ...	NULL

The first row, “MinionDefault”, is the set of default values to use for all the databases in the SQL Server instance. These values will be used for backup for all databases that do not have an additional row in this table.

The second row, [YourDatabase], specifies some values for YourDatabase. This row completely overrides the “DefaultMinion” values for Full backups on YourDatabase.

When full backups are performed for YourDatabase, *only* the values from the YourDatabase/Full row will be used. So, even though the system-wide default (as specified in the MinionDefault row) for DBPreCode is ‘EXEC specialCode;’, Full backups on YourDatabase will NOT use that default value. Because DBPreCode is NULL for YourDatabase/Full, Full backups will perform no pre code for YourDatabase.

For more information, see the [“Configuration Settings Hierarchy”](#) section in [“Architecture Overview”](#).

Example: Set custom configuration for Full backups on database ‘YourDatabase’.

```

INSERT INTO [Minion].[BackupSettings]
( [DBName] ,
  [BackupType] ,
  [Exclude] ,
  [LogLoc] ,
  [HistRetDays] ,
  [ShrinkLogOnLogBackup] ,
  [ShrinkLogThresholdInMB] ,
  [ShrinkLogSizeInMB] ,
  [Name] ,
  [ExpireDateInHrs] ,
  [RetainDays] ,
  [Descr] ,
  [Checksum] ,
  [Init] ,
  [Format] ,
  [MediaName] ,
  [MediaDescription]
)
SELECT 'YourDatabase' AS [DBName] ,
       'All' AS [BackupType] ,
       0 AS [Exclude] ,
       'Local' AS [LogLoc] ,
       60 AS [HistRetDays] ,

```

```

1 AS [ShrinkLogOnLogBackup] ,
1 AS [ShrinkLogThresholdInMB] ,
1024 AS [ShrinkLogSizeInMB] ,
'Backup name' AS [Name] ,
5 AS [ExpireDateInHrs] ,
2 AS [RetainDays] ,
'backup desc' AS [Descr] ,
1 AS [Checksum] ,
1 AS [Init] ,
1 AS [Format] ,
'MediaName' AS [MediaName] ,
'MediaDesc' AS [MediaDescription];

```

Minion.BackupSettingsPath

This table allows you to configure backup path destinations; and backup file copy and move settings. You may insert rows for individual databases, backup types, and copy/move settings, to override the default path settings for that database and backup type.

IMPORTANT: We highly recommend backing up to UNC paths, instead of to locally defined drives. Especially in the context of the Data Waiter feature, UNC paths allow a smoother transition between replicas or to a warm failover server. For more information, see [“About: Synchronizing settings and log data with the Data Waiter”](#).

Several “How To” sections provide instructions for copy, move, and mirror scenarios that use the Minion.BackupSettingsPath table:

- [How to: Set up mirror backups](#)
- [How to: Copy files after backup \(single and multiple locations\)](#)
- [How to: Move files to a location after backup](#)
- [How to: Copy and move backup files](#)
- [How to: Back up to multiple files in a single location](#)
- [How to: Back up to multiple locations](#)

Also see the discussion below, after the columns description.

Name	Type	Description
ID	Int	Primary key row identifier.
DBName	sysname	Database name.
isMirror	bit	Is a backup mirror location.
BackupType	Varchar	Backup type. Valid inputs: ALL

		<p>Full Diff Log ServerCert DatabaseCert Move Copy</p> <p>Note that ALL encompasses full, differential, and log backups.</p>
BackupLocType	varchar	<p>Backup location type.</p> <p>Valid inputs: Local NAS URL</p> <p>Note: URL is the most important of these, and is used by the Minion Backup process. The remaining inputs are just information for you. However, once combined with Minion Enterprise, these are all important for reporting.</p>
BackupDrive	Varchar	<p>Backup drive. This is only the drive letter of the backup destination.</p> <p>IMPORTANT: If this is drive, this must end with colon-slash (for example, 'M:\'). If this is URL, use the base path (for example, '\\server2\')</p>
BackupPath	varchar	<p>Backup path. This is only the path (for example, 'SQLBackups\') of the backup destination.</p>
ServerLabel	Varchar	<p>A user-customized label for the server name. It can be the name of the server, server\instance, or a label for a server.</p> <p>This is used for the backup file path.</p> <p>This comes in handy especially in Availability groups; if on day 1 we are on AG node 1, and on day 2 we are on AG node 2, we don't want the backups to save to different physical locations based on that name change. We instead provide a label for all databases on the instance – whether or not they're in an AG – so backups will all be in a central place (and so that cleaning up old backups is not an onerous chore).</p> <p>As this is just a label meant to group backup files, you could conceivably use it any which</p>

		<p>way you like; for example, one label for AG databases, and another for non-AG, etc.</p>
FileActionMethod		<p>Used to specify the program to use to perform the COPY/MOVE actions.</p> <p>Note: NULL and COPY are the same. And while the setting is called COPY, it uses PowerShell COPY or MOVE commands as needed.</p> <p>Valid inputs: NULL (same as COPY) COPY MOVE XCOPY ROBOCOPY ESEUTIL</p> <p>Note that ESEUTIL requires additional setup. For more on this topic, see “How to Topics: Backup Mirrors and File Actions” and “About: Copy and move backup files”.</p>
FileActionMethodFlags		<p>Used to supply flags for the method specified in <i>FileActionMethod</i>. The flags will be appended to the end of the command; this is the perfect way to provide specific functionality like preserving security, attributes, etc.</p> <p>For more on this topic, see “How to Topics: Backup Mirrors and File Actions” and “About: Copy and move backup files”.</p>
PathOrder	Int	<p>If a backup goes to multiple drives, or is copied to multiple drives, then PathOrder is used to determine the order in which the different drives are used.</p> <p>IMPORTANT: Like all ranking fields in Minion, PathOrder is a weighted measure. Higher numbers have a greater “weight” - they have a higher priority - and will be used earlier than lower numbers.</p>
IsActive	bit	<p>The current row is valid (active), and should be used in the Minion Backup process.</p>
AzureCredential	Varchar	<p>The name of the credential used to back up to a Microsoft Azure Blob.</p> <p>When you take a backup to a Microsoft Azure Blob (with TO URL='...'), you must set up a credential under security so you can access that</p>

		blob. You have to pass that into the backup statement (WITH CREDENTIAL='...'). See https://msdn.microsoft.com/en-us/jj720558
Comment	Varchar	For your reference only. You can label each row with a short description and/or purpose.

Discussion:

The Minion.BackupSettingsPath table comes with one default row: DBName='MinionDefault' and isMirror=0. If all of your backups are going to same location, you only need to update this row with your backup location.

You can also insert additional rows to configure the backup file target for an individual database, to override the default backup settings for that database.

You can also insert a row with BackupType='MOVE', to move a backup file after the backup operations are complete; and/or one or more rows with BackupType='COPY' to copy a backup file. Both MOVE and COPY operations are performed at a time designated by the FileActionTime field in the Minion.BackupSettings table. For example, if FileActionTime is set to 'AfterBackup', then a MOVE or COPY specified here in Minion.BackupSettingsPath will happen immediately after that backup (instead of at the end of the entire backup operation).

To backup a server certificate or database certificate, you must insert a row with BackupType = 'ServerCert'. Server certificate backups don't make use of the DBName field, so you can set it to 'MinionDefault', to signify that it applies universally. To backup a database certificate, you must insert an individual row for each — either DBName = 'MinionDefault' and BackupType = 'DatabaseCert', or BackupType='DatabaseCert' for a specific database.

Minion Backup will not back up certificates without an explicit BackupType='ServerCert' / 'DatabaseCert' row(s). You can have multiple certificate backup path rows for the same database (or for the server) going to multiple locations, all with isActive = 1. This is because certificates are so important to the restoration of a database, that Minion Backup allows you to back up the certificates to multiple locations. If you have five rows for DB2 database certificate backups, and all are set to isActive = 1, then all five of them are valid and will be executed. For more information, see the "[How to: Configure certificate backups](#)" section.

Minion.BackupSettingsServer

This table contains server-level backup settings. Specifically, each row represents a backup scenario as defined by the database type, backup type, day, begin and end time, and maximum number of backups per timeframe. The backup job (*MinionBackup-AUTO*) runs regularly in conjunction with this table to provide a wide range of backup options, all without introducing additional jobs.

In addition, you can enable settings synchronization, and/or log synchronization, for any or all of the backup scenarios. (So for example, Minion Backup can synchronize settings and logs with the weekly full backups.)

Minion.BackupSettingsServer ships with a full set of schedules in place.

Name	Type	Description
ID	Int	Primary key row identifier.
DBType	varchar	Database type. Valid values: User System
BackupType	varchar	Backup type. Valid inputs: Full Diff Log
Day	Varchar	The day or days to which the settings apply. See the discussion below for information about Day hierarchy and precedence. Valid inputs: Daily Weekday Weekend [an individual day, e.g., Sunday] FirstOfMonth LastOfMonth FirstOfYear LastOfYear
ReadOnly	tinyint	Backup readonly option; this decides whether or not to include ReadOnly databases in the backup, or to perform backups on <i>only</i> ReadOnly databases. A value of 1 includes ReadOnly databases; 2 excludes ReadOnly databases; and 3 only includes ReadOnly databases. Valid values: 1 2 3
BeginTime	varchar	The start time at which this schedule applies.

		IMPORTANT: Must be in the format <i>hh:mm:ss</i> , or <i>hh:mm:ss:mmm</i> (where <i>mmm</i> is milliseconds), on a 24 hour clock. This means that both '00:00:00' and '08:15:00:000' are valid times, but '8:15:00:000' is not (because single digit hours must have a leading 0).
EndTime	varchar	The end time at which this schedule applies. IMPORTANT: Must be in the format <i>hh:mm:ss</i> , or <i>hh:mm:ss:mmm</i> (where <i>mmm</i> is milliseconds), on a 24 hour clock. This means that both '00:00:00' and '08:15:00:000' are valid times, but '8:15:00:000' is not (because single digit hours must have a leading 0).
MaxForTimeframe	int	Maximum number of iterations within the specified timeframe (BeginTime to EndTime). For more information, see " Table based scheduling " in the " Quick Start " section.
CurrentNumBackups	int	Count of backup attempts for the particular DBType, BackupType, and Day, for the current timeframe (BeginTime to EndTime)
NumConcurrentBackups	tinyint	For future use.
LastRunDateTime	datetime	The last time a backup ran that applied to this particular scenario (DBType, BackupType, Day, and timeframe).
Include	varchar	The value to pass into the @Include parameter of the Minion.BackupMaster job; in other words, the databases to include in this attempt. This may be left NULL (meaning "all databases").
Exclude	varchar	The value to pass into the @Exclude parameter of the Minion.BackupMaster job; in other words, the databases to exclude from this attempt. This may be left NULL (meaning "no exclusions").
SyncSettings	bit	Whether or not to perform a synchronization of settings tables during this particular run. For more information, see " How to: Synchronize backup settings and logs among instances ".
SyncLogs	bit	Whether or not to perform a synchronization of log tables during this particular run. For more information, see " How to: Synchronize backup settings and logs among instances ".
BatchPreCode	varchar	Precode to run before the entire backup operation.

BatchPostCode	varchar	Precode to run after the entire backup operation.
IsActive	Bit	Whether the current row is valid (active), and should be used in the Minion Backup process.
Comment	varchar	For your reference only. You can label each row with a short description and/or purpose.

Discussion: Hierarchy and Precedence

There is an order of precedence to these settings, from least frequent (First/LastOfYear) to most frequent (daily); the least frequent setting, when it applies, takes precedence over all others. For example, if today is the first of the year, and there is a FirstOfYear setting, that's the one it runs.

The full list, from most frequent, to least frequent (and therefore of highest precedence), is:

1. Daily
2. Weekday / Weekend
3. Monday / Tuesday / Wednesday / Thursday / Friday / Saturday / Sunday
4. FirstOfMonth / LastOfMonth
5. FirstOfYear / LastOfYear

Note that the least frequent "Day" settings – FirstOfYear, LastOfYear, FirstOfMonth, LastOfMonth – only apply to user databases, not to system databases. System databases may have "Day" set to a day of the week (e.g., Tuesday), Daily, or NULL (which is equivalent to "Daily").

Discussion: Overlapping Schedules, and MaxForTimeframe

The Minion.BackupSettingsServer table allows you to have backup schedule settings that overlap. For example, we could perform a differential backup at the top of every hour, and then log backups every 5 minutes. For this scenario, we would:

- Insert 24 rows (one per hour) for the differential backup, each with a MaxForTimeframe value of 1.
- Insert one row for log backups, with a MaxForTimeframe value of 288 (or more, as there are only 288 5-minute increments in a day).
- Set the backup job MinionBackup-AUTO to run every 5 minutes.

The sequence of job executions then goes like this:

1. At 8:00am, the MinionBackup-AUTO job will run.
2. Minion Backup determines that a differential backup is slated for that hour.
3. MB will execute the differential backup, which takes precedence over the log backup. The log backup is *not* executed during this run.
4. MB also increments the differential CurrentNumBackups for that timeframe.
5. At 8:05, the MinionBackup-AUTO job will run again.

6. Minion Backup determines that the differential backup slated for that hour is already complete. (The differential is limited to one per hour via the MaxForTimeframe field.)
7. MB executes the log backup, and increments the differential CurrentNumbackups.

And, so on.

Important: The MaxForTimeframe field may limit you when running manual backups. For example, if only one full backup is slated for Saturday, and it has already run, then CurrentNumBackups will be 1. As the daily MaxForTimeframe value is 1, executing Minion.BackupMaster will fail, because the max has been reached. [] even a manual run won't let you run that backup. You would have to either reset the count, or change MaxForTimeframe to 2 (and then change it back after the manual run).

Discussion: Sample row for missing backups

Remember that you can run Minion.BackupMaster with Include='Missing' (either in the parameter, or in Minion.BackupSettingsServer, if you're using table based scheduling) to check for incomplete backups from the last run, for a given database type and backup type (e.g., 'User', 'Diff')

The Minion.BackupSettingsServer includes a sample row – the Include='Missing' row, which is inactive by default –to check for missing differential backups. The row is scheduled to run once at 5:00am (but it won't, unless you set isActive = 1). This is an example that you could enable, to give your routine an automatic check for missing backups.

Example 1: Weekly full, daily differential, hourly log backups

We could use this table to define the following backup time scenarios:

- Full system backups on Sunday, one time between 6pm and 7pm.
- Full user backups on Sunday, one time between 8pm and 9pm.
- Differential backups on every other day (Monday-Saturday), one time each between 8pm and 9pm.
- Log backups hourly (except when differential or full backups are running).

To do this, we would set the MinionBackup-AUTO backup job to run once hourly, and define the following rows. (Note that some of the table columns are omitted, for presentation purposes.)

ID	DBType	BackupType	Day	ReadOnly	BeginTime	EndTime	MaxForTimeframe
5	System	Full	Sunday	1	18:00:00	19:00:00	1
6	User	Full	Sunday	1	20:00:00	21:00:00	1
7	User	Diff	Weekday	1	20:00:00	21:00:00	1
8	User	Diff	Saturday	1	20:00:00	21:00:00	1
9	User	Log	Sunday	1	00:00:00	23:59:59	24

We do not have to specifically time the log backups to avoid the 8pm differential and full backup windows; because both differential and full backups take precedence over log backups. So when the 8pm job begins, it

will see the differential or full backup slated, and discard the log backup for that hour. In other words, the job run history would look like this:

- Sunday 7pm – user log backup, system full backup
- Sunday 8pm – user full backup
- Sunday 9pm – user log backup
- *Continuing hourly log backups...*
- Monday 7pm – user log backup
- Monday 8pm – user diff backups
- *Etc...*

Example 2: Daily full, differential every 4 hours, log backups every 15 minutes

We could use this table to define the following backup time scenarios:

- Full system backups daily, one time between 9pm and 9:30pm.
- Full user backups daily, one time between 10pm and 10:30pm.
- Differential backups every 4 hours (except when full backups are running), starting at 2:00am.
- Log backups every 15 minutes (except when differential or full backups are running).

To do this, we would set the MinionBackup-AUTO backup job to run every 15 minutes, and define the following rows. (Note that some of the table columns are omitted, for presentation purposes.)

ID	DBType	BackupType	Day	ReadOnly	BeginTime	EndTime	MaxForTimeframe
5	System	Full	Daily	1	21:00:00	21:30:00	1
6	User	Full	Daily	1	22:00:00	22:30:00	1
7	User	Diff	Daily	1	02:00:00	02:30:00	1
8	User	Diff	Daily	1	06:00:00	06:30:00	1
9	User	Diff	Daily	1	10:00:00	10:30:00	1
10	User	Diff	Daily	1	14:00:00	14:30:00	1
11	User	Diff	Daily	1	18:00:00	18:30:00	1
12	User	Log	Daily	1	00:00:00	23:59:59	96

In short, we need one row each for:

- full daily system backups
- full daily user backups
- full log backups (these run every 15 minutes)

And additionally, one row per each differential backup timeframe (2am, 6am, 10am, 2pm, and 6pm). We don't take a differential at 10pm, of course, because that is when the full backup will run.

Note: The 10pm user log backups will be replaced by the 10pm user full backups.

Minion.BackupTuningThresholds

This table holds the thresholds used to determine the tuning of a backup.

For more information, see the sections “[About: Dynamic Backup Tuning Thresholds](#)” and “[How to: Set up dynamic backup tuning thresholds](#)”.

Name	Type	Description
DBName	sysname	Database name.
BackupType	Varchar	Backup type. Valid inputs: ALL Full Diff Log Note that ALL encompasses full, differential, and log backups.
SpaceType	varchar	The way in Minion Backup determines the size of the database (e.g., data only, data and index, etc.) Note that this column is ignored for log backups, but you should put “Log” here anyway for rows where BackupType=Log, because it’s descriptive. Valid inputs: DataAndIndex Data File Log
ThresholdMeasure	char	The measure for our threshold value. Valid inputs: GB
ThresholdValue	bigint	The correlating value to ThresholdMeasure. So, if ThresholdMeasure is GB, then ThresholdValue is the value – the number of gigabytes.
NumberOfFiles	tinyint	The number of files to use for the backup.
Buffercount	Smallint	From MSDN.Microsoft.com : “Specifies the total number of I/O buffers to be used for the backup operation. You can specify any positive integer; however, large numbers of buffers might cause "out of memory" errors because of inadequate virtual address space in the Sqlservr.exe process.”

MaxTransferSize	bigint	<p>Max transfer size, as specified in bytes. This must be a multiple of 64KB.</p> <p>Note that a value of 0 will allow Minion Backup to use the SQL Server default value, typically 1MB.</p> <p>From MSDN.Microsoft.com: “Specifies the largest unit of transfer in bytes to be used between SQL Server and the backup media. The possible values are multiples of 65536 bytes (64 KB) ranging up to 4194304 bytes (4 MB).”</p>
Compression	bit	<p>From MSDN.Microsoft.com: “In SQL Server 2008 Enterprise and later versions only, specifies whether backup compression is performed on this backup, overriding the server-level default.”</p>
BlockSize	bigint	<p>From MSDN.Microsoft.com: “Specifies the physical block size, in bytes. The supported sizes are 512, 1024, 2048, 4096, 8192, 16384, 32768, and 65536 (64 KB) bytes. The default is 65536 for tape devices and 512 otherwise. Typically, this option is unnecessary because BACKUP automatically selects a block size that is appropriate to the device. Explicitly stating a block size overrides the automatic selection of block size.”</p>
BeginTime	varchar	<p>The start time at which this threshold applies.</p> <p>IMPORTANT: Must be in the format <i>hh:mm:ss</i>, or <i>hh:mm:ss:mmm</i> (where <i>mmm</i> is milliseconds), on a 24 hour clock. This means that both '00:00:00' and '08:15:00:000' are valid times, but '8:15:00:000' is not (because single digit hours must have a leading 0).</p>
EndTime	Varchar	<p>The end time at which this threshold applies.</p> <p>IMPORTANT: Must be in the format <i>hh:mm:ss</i>, or <i>hh:mm:ss:mmm</i> (where <i>mmm</i> is milliseconds), on a 24 hour clock. This means that both '00:00:00' and '08:15:00:000' are valid times, but '8:15:00:000' is not (because single digit hours must have a leading 0).</p>
DayOfWeek	Varchar	<p>The day or days to which the settings apply.</p> <p>Valid inputs: Weekday Weekend [an individual day, e.g., Sunday]</p>

IsActive	Bit	Whether the current row is valid (active), and should be used in the Minion Backup process.
Comment	Varchar	For your reference only. You can label each row with a short description and/or purpose.

Minion.DBMaintRegexLookup

Allows you to exclude databases from index maintenance (or all maintenance), based off of regular expressions.

Note that this procedure is shared between Minion modules.

Name	Type	Description
Action	varchar	Action to perform with this regular expression. Valid inputs: INCLUDE EXCLUDE
MaintType	varchar	Maintenance type to which this applies. Valid inputs: All Reindex Backup CheckDB
Regex	nvarchar	Regular expression to match a database name, or set of database names.
Comments	varchar	For your reference only. You can label each row with a short description and/or purpose.

Discussion:

Note that you can create more than one regular expression in Minion.DBMaintRegexLookup. For example:

- **To use Regex to include only DB3, DB4, and DB5:** insert a row like the example above, where Regex = 'DB[3-5](?!\d)'.
- **To use Regex to include any database beginning with the word "Market" followed by a number:** insert a row where Regex='Market[0-9]'.
- **With these two rows,** a backup operation with @Include='Regex' will backup *both* the DB3-DB5 databases, and the databases Marketing4 and Marketing308 (and similar others, if they exist).

For more information, see "[How To: Include databases in backups](#)" and "[How To: Exclude databases from backups](#)".

Minion.SyncServer

Configure the synchronization server information per database here in Minion.SyncServer.

For more information, see the sections [“How to: Synchronize backup settings and logs among instances”](#), [“Minion.SyncCmds”](#), and [“Minion.SyncErrorCmds”](#).

Name	Type	Description
ID	int	Primary key row identifier.
DBName	sysname	Database type. Valid values: User System
SyncServerName	varchar	Name of the target server, or of the target category, where you want to ship the table entries to. <ul style="list-style-type: none"> • Use “AGReplica” if the database is in an Availability Group. Minion Backup will automatically ship to all the replicas for that AG. • Use “MirrorPartner” if the databases is mirrored, and you want to sync to the mirroring partner. • Use “LogShippingPartner” in a log shipping scenario. • Or use the specific server name. <p>If you have either a server that isn’t one of those three, OR an AG replica where you only want to send to 1 or 2 replicas, you can enter in server names manually.</p> <p>Single server: “servername\instancename”, e.g. “Server1”, “Server2\SQL”.</p> <p>If you have multiple servers, you don’t need multiple rows; just use pipes: “Server1 Server2\SQL Server3”. One example of where this would be useful: if you routinely do restores to a development server from a production server, you can sync the logs from the production server to the development server.</p>
SyncDBName	sysname	Your management database, where the Minion objects reside. (This is either ‘master’, or your custom management database.)
Port	int	The port to be used for the connection to the target SQL Server.

Log Tables Detail

Minion.BackupDebug

This table holds high level debugging data from backup runs where debugging was enabled. Both the Minion.BackupMaster and the Minion.BackupDB stored procedures allow you to enable debugging.

Note: The data in Minion.BackupDebug and Minion.BackupDebugLogDetails is useful to Minion support. Contact us through www.MinionWare.net for help with your backup scenarios and debugging.

Minion.BackupDebugLogDetails

This table holds detailed debugging data from backup runs where debugging was enabled. Note: The data in Minion.BackupDebug and Minion.BackupDebugLogDetails is useful to Minion support. Contact us through www.MinionWare.net for help with your backup scenarios and debugging.

Minion.BackupFileListOnly

This table holds log of RESTORE FILELISTONLY output for each backup taken.

For most of the following columns, you can get more information from the MSDN article “RESTORE FILELISTONLY” at <https://msdn.microsoft.com/en-us/library/ms173778.aspx>.

Name	Type	Description
ID	Int	Primary key row identifier.
ExecutionDateTime	datetime	Date and time the file action took place.
DBName	sysname	Database name.
LogicalName	nvarchar	Database file logical name.
PhysicalName	nvarchar	Database file physical name, with path.
Type	char	File type.
FileGroupName	nvarchar	Filegroup name.
Size	numeric	File size in bytes.
MaxSize	numeric	Maximum allowed size in bytes.
FileID	bigint	File identifier.
CreateLSN	numeric	Log sequence number at which the file was created.
DropLSN	numeric	Log sequence number at which the file was dropped.
UniqueID	uniqueidentifier	File GUID.
ReadOnlyLSN	numeric	See MSDN article “RESTORE FILELISTONLY”.
ReadWriteLSN	numeric	See MSDN article “RESTORE FILELISTONLY”.
BackupSizeInBytes	bigint	Size of the backup for this file, in bytes.
SourceBlockSize	Int	See MSDN article “RESTORE FILELISTONLY”.
FileGroupID	int	See MSDN article “RESTORE FILELISTONLY”.
LogGroupGUID	uniqueidentifier	See MSDN article “RESTORE FILELISTONLY”.
DifferentialBaseLSN	numeric	See MSDN article “RESTORE FILELISTONLY”.
DifferentialBaseGUID	uniqueidentifier	See MSDN article “RESTORE FILELISTONLY”.
IsReadOnly	bit	Whether the file is read only.
IsPresent	Bit	See MSDN article “RESTORE FILELISTONLY”.

TDEThumbprint	varbinary	The thumbprint of the Database Encryption Key.
---------------	-----------	--

Minion.BackupFiles

A log of all backup files (whether they originate from a backup, a copy, or a move). A backup that is striped to 10 files will have 10 rows in this table. A backup that has one file, but is then copied to one other location, will have two rows in this table.

Note: With dynamic backup tuning, a backup could have 3 files one day, 10 files the next, 5 the next, and so on.

Many of the fields in this table are taken directly from BACKUP HEADERONLY. Refer to the BACKUP HEADERONLY article on MSDN: <https://msdn.microsoft.com/en-us/library/ms178536.aspx>

Name	Type	Description
ID	bigint	Primary key row identifier.
ExecutionDateTime	Datetime	Date and time the entire backup operation took place.
Op	varchar	The operation that was performed. For example: Backup, Copy, or Move.
Status	varchar	Current status of the file operation.
DBName	sysname	Database name.
ServerLabel	varchar	The user-customized label for the server name. For more information, see the ServerLabel column in Minion.BackupSettingsPath.
NETBIOSName	varchar	NetBIOS name.
BackupType	varchar	Specifies full, log, or differential backups. Example values: Full Log Diff Private Key Certificate
BackupLocType	Varchar	Backup location type. Example values: Local NAS URL Note: URL is the most important of these, and is used by the Minion Backup process. The remaining inputs are user defined, as they're just information for you.

BackupDrive	varchar	Backup drive. This is only the drive letter of the backup destination. IMPORTANT: If this is drive, this must end with colon-slash (for example, 'M:'). If this is URL, use the base path (for example, '\\server2\')
BackupPath	varchar	Backup path. This is only the path (for example, 'SQLBackups\') of the backup destination.
FullPath	varchar	The full path without filename. For example: "C:\SQLBackups\Server1\DB1".
FullFileName	varchar	The full path (drive, path, and file name) of the backup file. For example: "C:\SQLBackups\Server1\DB1\1of1LogDB120150514085245.TRN"
FileName	varchar	Base file name, without extension. For example, "1of1LogDB120150514085245".
DateLogic	varchar	The date and time, in YYYYMMDDHHMMSS format. For example, 20150514085245. This is used in generating the backup filename.
Extension	varchar	The file extension. For example, ".TRN".
RetHrs	int	Number of hours to retain the backup files.
IsMirror	bit	Is a backup mirror location.
ToBeDeleted	datetime	Date that the file is set to be deleted.
DeleteDateTime	datetime	Date that the file was deleted.
IsDeleted	bit	Whether the file has been deleted.
IsArchive	bit	Whether the file is marked as "Archived", which protects the file from being deleted at any time.
BackupSizeInMB	numeric	The size of the entire backup, in MB.
BackupName	varchar	See the MSDN article "RESTORE HEADERONLY".
BackupDescription	varchar	See the MSDN article "RESTORE HEADERONLY".
ExpirationDate	datetime	See the MSDN article "RESTORE HEADERONLY".
Compressed	bit	See the MSDN article "RESTORE HEADERONLY".
POSITION	tinyint	See the MSDN article "RESTORE HEADERONLY".
DeviceType	tinyint	See the MSDN article "RESTORE HEADERONLY".
UserName	varchar	See the MSDN article "RESTORE HEADERONLY".
DatabaseName	Sysname	See the MSDN article "RESTORE HEADERONLY".
DatabaseVersion	int	See the MSDN article "RESTORE HEADERONLY".
DatabaseCreationDate	datetime	See the MSDN article "RESTORE HEADERONLY".
BackupSizeInBytes	bigint	See the MSDN article "RESTORE HEADERONLY".
FirstLSN	varchar	See the MSDN article "RESTORE HEADERONLY".
LastLSN	varchar	See the MSDN article "RESTORE HEADERONLY".
CheckpointLSN	varchar	See the MSDN article "RESTORE HEADERONLY".
DatabaseBackupLSN	varchar	See the MSDN article "RESTORE HEADERONLY".
BackupStartDate	datetime	See the MSDN article "RESTORE HEADERONLY".
BackupFinishDate	datetime	See the MSDN article "RESTORE HEADERONLY".
SortOrder	int	See the MSDN article "RESTORE HEADERONLY".
CODEPAGE	int	See the MSDN article "RESTORE HEADERONLY".
UnicodeLocaleId	int	See the MSDN article "RESTORE HEADERONLY".
UnicodeComparisonStyle	int	See the MSDN article "RESTORE HEADERONLY".

CompatibilityLevel	int	See the MSDN article "RESTORE HEADERONLY".
SoftwareVendorId	int	See the MSDN article "RESTORE HEADERONLY".
SoftwareVersionMajor	int	See the MSDN article "RESTORE HEADERONLY".
SoftwareVersionMinor	int	See the MSDN article "RESTORE HEADERONLY".
SoftwareVersionBuild	int	See the MSDN article "RESTORE HEADERONLY".
MachineName	varchar	See the MSDN article "RESTORE HEADERONLY".
Flags	int	See the MSDN article "RESTORE HEADERONLY".
BindingID	varchar	See the MSDN article "RESTORE HEADERONLY".
RecoveryForkID	varchar	See the MSDN article "RESTORE HEADERONLY".
COLLATION	varchar	See the MSDN article "RESTORE HEADERONLY".
FamilyGUID	varchar	See the MSDN article "RESTORE HEADERONLY".
HasBulkLoggedData	bit	See the MSDN article "RESTORE HEADERONLY".
IsSnapshot	bit	See the MSDN article "RESTORE HEADERONLY".
IsReadOnly	bit	See the MSDN article "RESTORE HEADERONLY".
IsSingleUser	bit	See the MSDN article "RESTORE HEADERONLY".
HasBackupChecksums	bit	See the MSDN article "RESTORE HEADERONLY".
IsDamaged	bit	See the MSDN article "RESTORE HEADERONLY".
BeginsLogChain	bit	See the MSDN article "RESTORE HEADERONLY".
HasIncompleteMeatdata	bit	See the MSDN article "RESTORE HEADERONLY".
IsForceOffline	bit	See the MSDN article "RESTORE HEADERONLY".
IsCopyOnly	bit	See the MSDN article "RESTORE HEADERONLY".
FirstRecoveryForkID	varchar	See the MSDN article "RESTORE HEADERONLY".
ForkPointLSN	varchar	See the MSDN article "RESTORE HEADERONLY".
RecoveryModel	varchar	See the MSDN article "RESTORE HEADERONLY".
DifferentialBaseLSN	varchar	See the MSDN article "RESTORE HEADERONLY".
DifferentialBaseGUID	varchar	See the MSDN article "RESTORE HEADERONLY".
BackupTypeDescription	varchar	See the MSDN article "RESTORE HEADERONLY".
BackupSetGUID	varchar	See the MSDN article "RESTORE HEADERONLY".
CompressedBackupSize	bigint	See the MSDN article "RESTORE HEADERONLY".
CONTAINMENT	tinyint	See the MSDN article "RESTORE HEADERONLY".

Minion.BackupHeaderOnlyWork

This table is for internal use only. Do not modify in any way.

Minion.BackupLog

Contains records of backup operations. It contains one time-stamped row for each run of Minion.BackupMaster, which may encompass several database backup operations. This table stores status information for the overall backup operation. This information can help with troubleshooting, or just information gathering when you want to see what has happened between one backup run to the next.

Name	Type	Description
ID	bigint	Primary key row identifier.
ExecutionDateTime	datetime	Date and time the entire backup operation took place.
STATUS	varchar	Current status of the backup operation. If Live Insight is being used the status updates will

		<p>appear here. When finished, this column will typically either read 'Complete' or 'Complete with warnings'.</p> <p>If, for example, the backup process was halted midway through the operation, the Status would reflect the step in progress at the time the operation stopped.</p>
DBType	varchar	<p>Database type.</p> <p>Valid values: System User</p>
BackupType	varchar	<p>Backup type.</p> <p>Valid values: Full Diff Log</p>
StmtOnly	bit	Only generated backup statements, instead of running them.
NumDBsOnServer	int	Number of databases on server.
NumDBsProcessed	int	Number of databases processed in this backup operation.
TotalBackupSizeInMB	float	Total size of all backup files, in MB.
ReadOnly	tinyint	<p>Backup readonly option; this decides whether or not to include ReadOnly databases in the backup, or to perform backups on <i>only</i> ReadOnly databases.</p> <p>A value of 1 includes ReadOnly databases; 2 excludes ReadOnly databases; and 3 only includes ReadOnly databases.</p> <p>Valid values: 1 2 3</p>
ExecutionEndDateTime	datetime	Date and time the entire backup operation completed.
ExecutionRunTimeInSecs	float	The duration, in seconds, of the entire backup operation.
BatchPreCode	varchar	Precode set to run before the entire backup operation. This code is set in the Minion.SettingsServer table.
BatchPostCode	varchar	Precode set to run after the entire backup operation. This code is set in the Minion.SettingsServer table.
BatchPreCodeStartDateTime	datetime	Start date of the batch precode.

BatchPreCodeEndDateTime	datetime	End date of the batch precode.
BatchPreCodeTimeInSecs	int	Batch precode time to run, in seconds.
BatchPostCodeStartDateTime	datetime	Start date of the batch postcode.
BatchPostCodeEndDateTime	datetime	End date of the batch postcode.
BatchPostCodeTimeInSecs	int	Batch precode time to run, in seconds.
IncludeDBs	varchar	A comma-delimited list of database names, and/or wildcard strings, to include in the backup operation. When this is 'All' or 'null', the operation processed all (non-excluded) databases.
ExcludeDBs	varchar	A comma-delimited list of database names, and/or wildcard strings, to exclude from the backup operation. When this is 'null', the operation excluded no databases (except those excluded by configuration in Minion.BackupSettings).
RegexDBsIncluded	varchar	A list of databases included in the backup operation via the Minion Backup regular expressions feature.
RegexDBsExcluded	varchar	A list of databases excluded from the backup operation via the Minion Backup regular expressions feature.

Minion.BackupLogDetails

Contains records of individual backup operations. It contains one time-stamped row for each individual database backup operation. This table stores the parameters and settings that were used during the operation, as well as status information. This information can help with troubleshooting, or just information gathering when you want to see what has happened between one backup run to the next.

Note: Several of the columns in this table are from the output of [Trace Flag 3213](#); you can read more about this trace flag at <http://blogs.msdn.com/b/psssql/archive/2008/01/28/how-it-works-sql-server-backup-buffer-exchange-a-vdi-focus.aspx>

Name	Type	Description
ID	bigint	Primary key row identifier.
ExecutionDateTime	datetime	Date and time the entire backup operation took place. If the job were started through BackupMaster then all databases in that run have the same ExecutionDateTime. If the job was run manually from Minion.BackupDB, then this value will only be for this database. It will still have a matching row in the Minion.BackupLog table.
STATUS	varchar	Current status of the backup operation. If Live Insight is being used the status updates

		will appear here. When finished, this column will typically either read 'Complete' or 'Complete with warnings'.
PctComplete	tinyint	Backup percent complete (e.g., 50% complete).
DBName	sysname	Database name.
ServerLabel	varchar	A user-customized label for the server name. It can be the name of the server, server\instance, or a label for a server. For more information, see the ServerLabel column in Minion.BackupSettingsPath.
NETBIOSName	varchar	The name of the server from which the backup is taken. If the instance is on a cluster, this will be the name of the cluster node SQL Server was running on. If it's part of an Availability Group, the NETBIOSName will be the physical name of the Availability Group replica.
IsClustered	bit	Flag: is clustered.
IsInAG	bit	Flag: is in an Availability Group.
IsPrimaryReplica	bit	Flag: is the primary replica.
DBType	varchar	Database type. Valid values: User System
BackupType	varchar	Backup type. Valid values: Full Diff Log
BackupStartDateTime	datetime	Date and time of backup start.
BackupEndDateTime	datetime	Date and time of backup end.
BackupTimeInSecs	float	Backup time, measured in seconds.
MBPerSec	float	Backup rate, in megabytes per second.
BackupCmd	varchar	The T-SQL command used to back up the database.
SizeInMB	float	Backup file size, in megabytes.
StmtOnly	bit	Flag: only generate statement.
READONLY	tinyint	Backup readonly option; this decides whether or not to include ReadOnly databases in the backup, or to perform backups on <i>only</i> ReadOnly databases.

		<p>A value of 1 includes ReadOnly databases; 2 excludes ReadOnly databases; and 3 only includes ReadOnly databases.</p> <p>Valid values: 1 2 3</p>
BackupGroupOrder	int	<p>Group to which this table belongs. Used solely for determining the order in which tables should be backed up.</p> <p>Most of the time this will be 0. However, if you choose to take advantage of this feature a row in Minion.BackupSettings will get you there. This is a weighted list so higher numbers are more important and will be processed first.</p> <p>For more information, see “How To: Back up databases in a specific order”.</p>
BackupGroupDBOrder	int	<p>Group to which this database belongs. Used solely for determining the order in which databases should be backed up.</p> <p>By default, all databases have a value of 0, which means they’ll be processed in the order they’re queried from sysobjects.</p> <p>Higher numbers have a greater “weight” (they have a higher priority), and will be backed up earlier than lower numbers. The range of GroupDBOrder weight numbers is 0-255.</p> <p>For more information, see “How To: Backup databases in a specific order”.</p>
NumberOfFiles	tinyint	<p>Number of backup files.</p> <p>Note that this is not at all related to the number of files in the database itself.</p>
Buffercount	int	<p>Total number of I/O buffers to be used for the backup operation. From the output of Trace Flag 3213.</p>
MaxTransferSize	bigint	<p>The largest unit of transfer (in bytes) to be used between SQL Server and the backup media. From the output of Trace Flag 3213.</p>
MemoryLimitInMB	bigint	<p>How much memory the system has available for backups. From the output of Trace Flag 3213.</p>

TotalBufferSpaceInMB	bigint	How much memory used to process the backup. From the output of Trace Flag 3213 .
FileSystemIOAlignInKB	int	The disk block size. From the output of Trace Flag 3213 .
SetsOfBuffers	tinyint	From the output of Trace Flag 3213 .
Verify	varchar	Specifies when the RESTORE VERIFYONLY operation is to happen.
Compression	bit	Flag: Whether backup compression is performed on this backup.
FileAction	varchar	Action to take with the backup file(s) (MOVE, COPY, or NULL).
FileActionTime	varchar	The time at which to perform the COPY or MOVE FileAction. Example values: AfterBackup AfterBatch
FileActionBeginDateTime	datetime	Date and time of the file action start.
FileActionEndDateTime	datetime	Date and time of the file action end.
FileActionTimeInSecs	int	File action time, measured in seconds.
UnCompressedBackupSizeMB	int	Size of the uncompressed backup, in megabytes.
CompressedBackupSizeMB	int	Size of the compressed backup, in megabytes.
CompressionRatio	float	Backup compression ratio. As noted in the MSDN Backup Compression article, “a 3:1 compression ratio indicates that you are saving about 66% on disk space”.
COMPRESSIONPct	numeric	Backup compression ratio, in percent. As noted in the MSDN Backup Compression article, “a 3:1 compression ratio indicates that you are saving about 66% on disk space”.
BackupRetHrs	tinyint	Number of hours to retain the backup files.
BackupLogging	varchar	Whether log data is only stored on the local (client) server, or on both the local server and the central Minion (repository) server. Example values: Local Repo
BackupLoggingRetDays	smallint	Number of days to retain a history of backups (in Minion Backup log tables). Minion Backup does not modify or delete backup information from the MSDB database.
DelFileBefore	bit	Whether backup files are to be deleted before or after the current backup.

DBPreCode	nvarchar	Code that ran before the backup operation begins for that database.
DBPostCode	nvarchar	Code that ran after the backup operation completed for that database.
DBPreCodeStartDateTime	datetime	The date and time that the database precode began.
DBPreCodeEndDateTime	datetime	The date and time that the database precode ended.
DBPreCodeTimeInSecs	int	The duration of the database precode run.
DBPostCodeStartDateTime	datetime	The date and time that the database postcode began.
DBPostCodeEndDateTime	datetime	The date and time that the database postcode ended.
DBPostCodeTimeInSecs	Int	The duration of the database postcode run.
IncludeDBs	varchar	Databases included in the backup batch.
ExcludeDBs	varchar	Databases excluded from the backup batch.
RegexDBsExcluded	varchar	Databases excluded from the backup batch via regular expressions.
Verified	bit	Specifies whether the RESTORE VERIFYONLY operation was performed.
VerifyStartDateTime	datetime	The date and time that RESTORE VERIFYONLY began.
VerifyEndDateTime	datetime	The date and time that RESTORE VERIFYONLY began.
VerifyTimeInSecs	Int	The duration of the RESTORE VERIFYONLY run.
IsInit	bit	Flag: Overwrite the existing backup set.
IsFormat	bit	Flag: Overwrite the existing media header. Note that Format=1 is equivalent to Format=1 AND Init=1; therefore, FORMAT=1 would have overridden the Init setting.
IsChecksum	bit	Flag: Verify each page for checksum and torn page (if enabled and available) and generate a checksum for the entire backup.
Descr	varchar	
IsCopyOnly	bit	Flag: Perform a copy-only backup.
IsSkip	bit	Flag: Skip the check of the backup set's expiration before overwriting.
BackupName	Varchar	Backup name.
BackupErrorMgmt	varchar	Rollup of the two BACKUP flags – STOP_ON_ERROR and CONTINUE_AFTER_ERROR.
MediaName	varchar	The backup set's media name.
MediaDescription	varchar	Description of the media set.
ExpireDateInHrs	int	Number of hours until the backup set for this backup can be overwritten.

		If both ExpireDateInHrs and RetainDays are both used, RetainDays takes precedence.
RetainDays	smallint	The number of days that must elapse before this backup media set can be overwritten.
MirrorBackup	bit	Flag: Mirror backup.
DynamicTuning	bit	Flag: Enable dynamic tuning.
ShrinkLogOnLogBackup	Bit	Flag: Turn on log shrink after log backups.
ShrinkLogThresholdInMB	int	How big (in MB) the log file is before Minion Backup will shrink it.
ShrinkLogSizeInMB	int	The size (in MB) the log file shrink should target. In other words, how big you would like the log file to be after a file shrink. This setting applies for EACH log file, not for all log files totaled.
PreBackupLogSizeInMB	float	Log size in MB before the backup.
PreBackupLogUsedPct	float	Log percent used before the backup.
PostBackupLogSizeInMB	float	Log size in MB after the backup.
PostBackupLogUsedPct	int	Log percent used after the backup.
PreBackupLogReuseWait	varchar	Log reuse wait description, before the backup.
PostBackupLogReuseWait	varchar	Log reuse wait description, after the backup.
VLFs	bigint	The number of Virtual Log Files.
FileList	varchar	A comma delimited list of backup files, in the format "DISK = '<full file path>', DISK = '<full file path>'".
IsTDE	bit	Flag: Is a TDE database.
BackupCert	bit	Flag: Certificate backups enabled.
CertPword	varbinary	Certificate password. This is the password used to protect the certificate backup.
IsEncryptedBackup	bit	Flag: Is an encrypted backup.
BackupEncryptionCertName	nchar	Backup encryption certificate name.
BackupEncryptionAlgorithm	varchar	Backup encryption certificate algorithm.
BackupEncryptionCertThumbPrint	varbinary	Backup encryption certificate thumbprint, a globally unique hash of the certificate.
DeleteFilesStartDateTime	datetime	The date and time that the file deletion began.
DeleteFilesEndDateTime	datetime	The date and time that the file deletion completed.
DeleteFilesTimeInSecs	int	The duration of the file deletion run.
Warnings	varchar	Warnings.

Minion.SyncCmds

This table holds the commands used to synchronize settings and log tables to target servers (which are configured in the Minion.SyncServer table). Minion.SyncCmds is both a log table and a work table: the synchronization process uses Minion.SyncCmds to push the synchronization commands to target servers, and it is also a log of those commands (complete and incomplete).

At the end of a backup, Minion Backup writes logged data to this table as INSERT commands. So, everything MB wrote to the log tables is automatically entered into this table as a command, to be used on the target instances. The same thing happens with changes to settings: when you configure Minion Backup to synchronize settings to a server, it writes those settings as commands in this table, to be run on the target servers.

For more information, see the sections [“How to: Synchronize backup settings and logs among instances”](#), [“Minion.SyncServer”](#), and [“Minion.SyncErrorCmds”](#).

Note: This table is used by Minion Backup, as well as (if installed) Minion Reindex, and other Minion modules.

Name	Type	Description
ID	Int	Primary key row identifier.
ExecutionDateTime	Datetime	Date and time the command took place.
Status	varchar	Current status of the sync for this command. Example values: In queue Complete
ObjectName	Sysname	The name of the table being synced (without the schema name attached). Example values: BackupSyncCmds BackupLogDetails BackupFiles
Op	varchar	Operation being performed on table. INSERT UPDATE DELETE TRUNCATE
Cmd	Nvarchar	The synchronization command to be pushed to one or more sync partners.
Pushed	bit	Whether it was successfully pushed to all servers.
Attempts	bigint	How many times it has attempted to send.
ErroredServers	varchar	Comma delimited list of servers to which this command failed to push. (The Data Waiter will retry these commands, and update the lists, automatically.)

Minion.SyncErrorCmds

This table holds synchronization commands that have failed, to be retried again later.

For more information, see the sections [“How to: Synchronize backup settings and logs among instances”](#), [“Minion.SyncServer”](#), and [“Minion.SyncCmds”](#).

Note: This table has the potential to very large, if a replica is down for a long time, or if many replicas are down. In that case, it might be wise to turn off synchronization for that particular server, and if necessary, clear that server’s records from Minion.SyncErrorCmds and reinitialize it as a new partner.

Name	Type	Description
ID	bigint	Primary key row identifier.
SyncServerName	varchar	Name of the synchronization target server.
SyncDBName	varchar	The target database name of the synchronization target server.
Port	varchar	Port number of the synchronization target server.
SyncCmdID	bigint	Command identity number, from the Minion.SyncCmds table.
STATUS	varchar	Status of the last synchronization attempt. Values include “Initial attempt failed”, and “Fatal error on [servername]”.
LastAttemptDateTime	datetime	Date last attempted to synchronize the command to the target server.

Discussion:

The synchronization logging process, along with Minion.SyncErrorCmds, makes it easy to bring a target server (subscriber) up to date if it has been unavailable for a time. For example, if a target server is shut down for a day, once it restarts, MB can easily replay those commands starting from the time the server went down.

Let us take the case where YourServer is set to synchronize with its three Availability Group replicas, and one of those replicas is down. The sync commands that fail to run against the downed replica will be logged here for as long as the replica is down. When that replica comes back online, Minion Backup will run through all the saved commands, bringing the replica’s tables back in sync with the primary tables. (Note that the other replicas will have been kept up to date this entire time.)

Minion.Work

This table is for internal use only. Do not modify in any way.

Overview of Procedures

Two separate procedures **execute backup operations** for Minion Backup: one procedure runs per database, and the other is a “Master” procedure that performs run time logic and calls the DB procedure as appropriate.

In addition, Minion Backup comes with a **Help procedure** to provide information about the system itself.

Backup procedures:

- **Minion.BackupMaster** – This procedure makes all the decisions on which databases to back up, and what order they should be in.
- **Minion.BackupDB** – This procedure is called by Minion.BackupMaster to perform backup for a single database.
- **Minion.HELP** – Display help on Minion Backup objects and concepts.

Procedures Detail

Minion.BackupDB

The Minion.Backup DB stored procedure performs backups for a single database. Minion.Backup DB is the procedure that creates and runs the actual backup statements for databases which meet the criteria stored in the settings table (Minion.BackupSettings).

IMPORTANT: We HIGHLY recommend using Minion.BackupMaster for all of your backup operations, even when backing up a single database. Do not call Minion.BackupDB to perform backups.

The Minion.Backup Master procedure makes all the decisions on which databases to back up, and what order they should be in. It's certainly possible to call Minion.BackupDB manually, to back up an individual database, but we instead recommend using the Minion.BackupMaster procedure (and just include the single database using the @Include parameter). First, it unifies your code, and therefore minimizes your effort. By calling the same procedure every time you reduce your learning curve and cut down on mistakes. Second, future functionality may move to the Minion.BackupMaster procedure; if you get used to using Minion.Backup Master now, then things will always work as intended.

Name	Type	Description
@DBName	SYSNAME	Database name.
@BackupType	VARCHAR	Backup type. Valid inputs: Full Log Diff
@StmtOnly	BIT	Generate back up statements without running the statements.
@ExecutionDateTime	DATETIME	Date and time the backup took place. If this SP was called by Minion.BackupMaster, @ExecutionDateTime will be passed in, so this backup is included as part of the entire (multi-database) backup operation.
@Debug	bit	Enable logging of special data to the debug tables. For more information, see " Minion.BackupDebug " and " Minion.BackupDebugLogDetails ".

Minion.BackupFileAction

This stored procedure is called by the backup routine to perform the backup file action – MOVE or COPY – you specified **in the table**. Minion.BackupFileAction will MOVE or COPY any number of files to any number of locations.

Name	Type	Description
@DBName	Sysname	Database name.
@DateLogic	Varchar	The date and time, in YYYYMMDDHHMMSS format. Used to select the correct records from Minion.BackupFiles.
@BackupType	Varchar	Backup type. Valid inputs: Full Log Diff
@ManualRun	Bit	Determines whether or not to log the backup action.

Note: This procedure only takes the first “move” command, because the file won't be there anymore if you try to move it twice. But, you can have as many copies as you like.

Warning: You should be careful as this can run for a very long time and could increase the time of your backups if you run this inline.

Minion.BackupFilesDelete

This stored procedure is responsible for deleting backup files from disk, which have aged out according to the RetHrs column in the Minion.BackupFiles table. It is called from Minion.BackupDB, and can be run either before or after the backup. Minion.BackupFilesDelete can also be run manually, with a custom retention hours setting.

Note: This routine will never delete a file where IsArchive = 1 in Minion.BackupFiles. Archive files are saved indefinitely.

For more information, see [“About: Backup file retention”](#).

Name	Type	Description
@DBName	varchar	Database name. The value 'All' will delete files for all databases on the instance. Valid options: <database name> All

@RetHrs	Int	Delete files older than the number of hours specified here. NULL will cause the SP to use the retention hours (RetHrs) field in the Minion.BackupFiles table.
@Delete	Bit	Delete files. Defaults to 1. @Delete=0 will return a list of the files that will be deleted, and the amount of space that would be freed.
@EvalDateTime	Datetime	Evaluate the file age against this date and time. Defaults to NULL, which evaluates the file dates against the current time. Passing in your own value causes the delete process to compute file age against this hypothetical date, instead of the current date. This lets you delete files, or see what files WOULD be deleted, as if it were a different datetime. Combined with @Delete = 0, and you can see what files will be deleted on which day, and how much disk space you would save. WARNING: If you set @EvalDateTime to a far enough date in the future (say, a year) and pass in @Delete=1, you will delete ALL of your backup files.

Discussion:

Minion.BackupFilesDelete is useful in a number of ways. Of course, it is run with every backup operation, to keep outdated backup files cleared out.

The interesting part of this stored procedure is the functionality the parameters give you:

- @DBName = 'All' will let you delete files for all databases on the server, based off of the other parameters. This is a great breakthrough when you need to clean up all of the databases' backup files. One example of when you would need this, is if permissions to the SQL account were removed from the NAS, and the files hadn't been deleting. You have 500 databases on the server, and they all need to be cleaned up. @DBName='All' would take care of it.
- @Delete = 0 will only *report on* what would be deleted, with the current parameter settings. (@Delete=0 is similar to PowerShell's -WhatIf parameter.)

- @RetHrs = NULL uses the RetHrs setting in the Minion.BackupSettings table. Pass in your own value instead, and the procedure will use that instead. This allows you to do custom cleanups.
- @EvalDateTime = NULL evaluates the file dates against the current time. Passing in your own value will evaluate the file dates against that time. This is very useful, as it lets you delete files as if it were a different datetime. Combined this with @Delete = 0, and you can see what files will be deleted on which day.

Below are three examples of how you can use this procedure:

- Delete files for a single database.
- Manually delete backup files, using a custom retention period.
- Check to see what databases *would* be deleted, for a custom retention period and date.

Example execution:

```
-- Delete files for a single database.
EXEC [Minion].[BackupFilesDelete]
    @DBName = 'DB1',
    @RetHrs = NULL, -- Use the configured retention period.
    @Delete = 1,
    @EvalDateTime = NULL;
```

Example execution:

```
-- Delete files for all databases, using a custom retention period.
EXEC [Minion].[BackupFilesDelete]
    @DBName = 'All',
    @RetHrs = 24, -- Pass in specific hrs to do a custom delete.
    @Delete = 1,
    @EvalDateTime = NULL;
```

Example execution:

```
-- Play "what if"; check to see what databases would be deleted.
EXEC [Minion].[BackupFilesDelete]
    @DBName = 'All',
    @RetHrs = NULL,
    @Delete = 0, -- 0: report files that will be deleted.
    @EvalDateTime = '6/1/2015 06:00:00'; -- The SP will pretend this is the current date.
```



Minion Enterprise Hint

We are planning a Minion Enterprise tool that will centrally delete backup files for all servers!

See www.MinionWare.net for more information, or email us today at Support@MidnightDBA.com for a demo!

Minion.BackupMaster

The Minion.Backup Master is the central procedure of Minion Backup. It uses the parameter and/or table data to make all the decisions on which databases to back up, and what order they should be in. This stored procedure calls the Minion.Backup DB stored procedure once per each database specified in the parameters; or, if @Include = "All" is specified, per each eligible database in sys.databases.

In addition, Minion.BackupMaster performs extensive logging, runs configured pre- and postcode, enables and disables the status monitor job (which updates log files for Live Insight, providing percent complete for each backup), determines AG backup location, performs file actions (such as copy and move), and runs the Data Waiter feature to synchronize log and settings data across instances.

In short, Minion.BackupMaster decides on, runs, or causes to run every feature in Minion Backup.

Name	Type	Description
@DBType	Varchar	The type of database. Valid inputs: System User
@BackupType	Varchar	Specifies full, log, or differential backups. Valid inputs: Full Log Diff
@StmtOnly	Bit	Allows you to generate backup statements only, instead of running them. This is a good option if you ever need to run backup statements manually.
@Include	Varchar	Use @Include to run backups on a specific list of databases, or databases that match a LIKE expression. Alternately, set @Include='All' or @Include=NULL to run maintenance on all databases.

		<p>If, during the last backup run, there were backups that failed, and you need to back them up now, just call this procedure with <code>@Include = 'Missing'</code>. The SP will search the log for the backups that failed in the previous batch (for a given BackupType and DBType), and back them up now. Note that the BackupType and DBType must match the errored out backups.</p> <p>Valid inputs: NULL Regex Missing <i><comma-separated list of DBs including wildcard searches containing '%></i></p> <p>For more information, see “How to: Include databases in backups”.</p>
@Exclude	Varchar	<p>Use @Exclude to skip backups for a specific list of databases, or databases that match a LIKE expression.</p> <p>Examples of valid inputs include: DBname DBName1, DBName2, etc. DBName%, YourDatabase, Archive%</p> <p>For more information, see “How To: Exclude databases from backups”.</p>
@ReadOnly	Tinyint	<p>Use @ReadOnly to (1) include ReadOnly databases, (2) exclude ReadOnly databases, or (3) only include ReadOnly databases.</p>
@Debug	bit	<p>Enable logging of special data to the debug tables.</p> <p>For more information, see “Minion.BackupDebug” and “Minion.BackupDebugLogDetails”.</p>

Minion.BackupRestoreDB

This procedure will generate restore statements based on existing backup files.

For full or differential backups, the procedure will generate a “restore database” statement based on the most recent backup of that backup type. For log backups, the procedure will generate a list of “restore log” statements, starting with the first log backup taken after the most recent full backup; and ending with the

most recent log backup. In other words, @BackupType='Log' will generate statements to roll through all recent log backups.

Name	Type	Description
@DBName	sysname	Database name
@BackupType	varchar	Backup type. Valid inputs: Full Diff Log
@BackupLoc	varchar	Backup location (by category). You can restore from the primary backup location, from a copy location, a mirror location, or a move location. Note: "Backup" and "Primary" both mean the primary backup location. Valid inputs: Backup Primary Mirror Copy Move
@StmtOnly	bit	Generate log statements only. Currently, @StmtOnly = 1 is the only valid input.

Example: Generate restore statement, from the mirror location, for the most recent DB1 full backup

```
EXEC [Minion].BackupRestoreDB
    @DBName = 'DB1',
    @BackupType = 'Log',
    @BackupLoc = 'Mirror',
    @StmtOnly = 1;
```

Example: Generate log restore statements for all log backups since the most recent DB1 full backup

```
EXEC [Minion].BackupRestoreDB
    @DBName = 'DB1',
    @BackupType = 'Log',
    @BackupLoc = 'Primary',
    @StmtOnly = 1;
```

Minion.BackupSyncLogs

This is a key "Data Waiter" procedure. It prepares log data to be pushed across to target servers.

The master backup procedure `Minion.BackupMaster` calls `Minion.BackupSyncLogs`, which loads log data to the `Minion.SyncCmds` table as insert and delete statements.

For more information, see [“How to: Synchronize backup settings and logs among instances”](#) and [“About: Synchronizing settings and log data with the Data Waiter”](#).

Name	Type	Description
@ExecutionDateTime	Datetime	The date of the backup batch to synchronize.

Minion.BackupSyncSettings

This is a key “Data Waiter” procedure. It prepares settings data to be pushed across to target servers.

The master backup procedure `Minion.BackupMaster` calls `Minion.BackupSyncSettings`, which loads a `TRUNCATE TABLE` statement to the `Minion.SyncCmds` table; then loads settings data to the table as insert statements.

Note: We chose to truncate and fully reinitialize settings data on sync partners; and to just push `INSERT/UPDATE/DELETE` statements for log data changes to sync partners; because settings tables tend to be far smaller tables than log tables, and it makes sense to get the full current “snapshot” of settings from the primary server.

For more information, see [“How to: Synchronize backup settings and logs among instances”](#) and [“About: Synchronizing settings and log data with the Data Waiter”](#).

Name	Type	Description
@ExecutionDateTime	Datetime	The date of the backup batch to synchronize.

Minion.BackupStatusMonitor

Updates `Minion.BackupLogDetails` with the percent complete of running backups.

The `Minion.BackupMaster` stored procedure starts the “`MinionBackupStatusMonitor`” job, which calls `Minion.BackupStatusMonitor`, at the beginning of a backup batch; and stops the job when the backup batch is complete.

Name	Type	Description
@IntervallnSecs	varchar	The amount of time to wait before updating the table again (in the format ' <code>h:m:ss</code> '). Default value = ' <code>0:00:05</code> ' (5 seconds).

Minion.BackupStmtGet

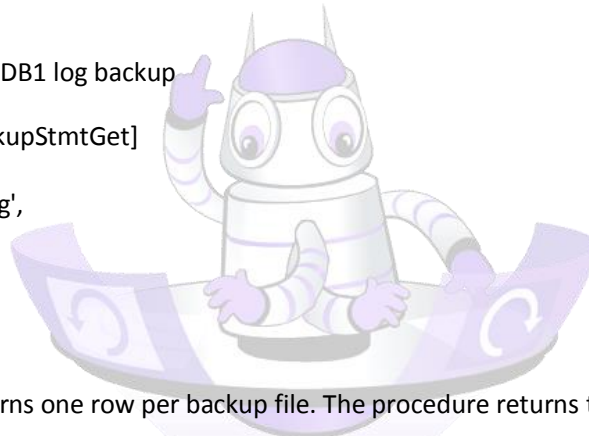
This stored procedure builds and returns a backup statement, along with associated data. The `Minion.BackupDB` procedure calls it to generate backup statements.

You can also use `Minion.BackupStmtGet` to determine which backup options and settings will be used for a given backup. This is particularly helpful for testing your settings and backup tuning thresholds.

Name	Type	Description
@DBName	Sysname	Database name.
@BackupType	Varchar	Specifies full, log, or differential backups. Valid inputs: Full Diff Log
@DBSize	Decimal	Database size. This parameter makes it possible to test the settings of the database at various hypothetical sizes. See discussion below.

Example: Get statement for DB1 log backup

```
EXEC [Minion].[BackupStmtGet]
@DBName = 'DB1',
@BackupType = 'Log',
@DBSize = NULL;
```



Discussion: The Result Set

`Minion.BackupStmtGet` returns one row per backup file. The procedure returns the backup command, as well as a long list of related items (such as server name, backup path, path order, compression, etc.).

Discussion: The DBSize Parameter

The `DBSize` parameter is especially cool. When you run `Minion.BackupStmtGet` with a specific `@DBSize`, the procedure generates the backup statement for the database as if the database were currently that size. Of course, with normal, untuned backups this would have no impact; but when you use backup tuning thresholds, the size of the database determines which settings will be used.

Let's say your database is 50GB, but you want to know if you've configured the dynamic settings correctly for it when it reaches 100GB. You can use the `@DBSize` parameter to test the settings like this:

```
EXEC [Minion].[BackupStmtGet]
@DBName = 'AdventureWorks',
@BackupType = 'Log',
@DBSize = 100;
```

This procedure will not run the backup, delete any files, or do any other action; it only generates the backup statements and returns them, along with backup files and other information. Feel free to use this as much as you like to help you make sure your configuration is what you expect.

Minion.CloneSettings

This procedure allows you to generate an insert statement for a table, based on a particular row in that table.

We made this procedure flexible: you can enter in the name of any Minion table, and a row ID, and it will generate the insert statement for you.

WARNING: This generates a clone of an existing row as an INSERT statement. Before you run that insert, be sure to change key identifying information - e.g., the DBName - before you run the INSERT statement; you would not want to insert a completely identical row.

Name	Type	Description
@TableName	Varchar	The name of the table to generate an insert statement for. Note: This can be in the format "Minion.BackupSettings" or just "BackupSettings".
@ID	Int	The ID number of the row you'd like to clone. See the discussion below.
@WithTrans	Bit	Include "BEGIN TRANSACTION" and "ROLLBACK TRANSACTION" clauses around the insert statement, for safety.

Discussion:

Because of the way we have written Minion Backup, you may often need to insert a row that is nearly identical to an existing row. If you want to change just one setting, you still have to fill out 40 columns. For example, you may wish to insert a row to Minion.BackupSettings that is only different from the MinionDefault row in two respects (e.g., DBName and Verify).

We created Minion.CloneSettings to easily duplicate any existing row in any table. This "helper" procedure lets you pass in the name of the table you would like to insert to, and the ID of the row you want to model the new row off of. The procedure it returns an insert statement so you can change the one or two values you want.

Discussion: Identity columns

If the table in question has an IDENTITY column, regardless of that column's name, Minion.CloneSettings will be able to use it to select your chosen row. For example, let's say that the IDENTITY column of Table1 is *ObjectID*, and that you call Minion.CloneSettings with @ID = 2. The procedure will identify that column and return an INSERT statement that contains the values from the row where ObjectID = 2.

Minion.HELP

Use this stored procedure to get help on any Minion Backup object without leaving Management Studio.

Name	Type	Description
------	------	-------------

@Module	Varchar	The name of the module to retrieve help for. Valid inputs include: NULL Reindex
@Name	varchar	The name of the topic for which you would like help. If you run Minion.HELP by itself, or with a @Module specified, it will return a list of available topics.

Examples:

For introductory help, run:

```
EXEC Minion.HELP;
```

For introductory help on Minion Backup, run:

```
EXEC Minion.HELP 'Backup';
```

For help on a particular topic – in this case, the Top 10 Features – run:

```
EXEC Minion.HELP 'Backup', 'Top 10 Features';
```

Minion.SyncPush

This is a key “Data Waiter” procedure. It pushes log and settings data to Minion Backup tables on other SQL Server instances, which are configured as synchronization partners.

Minion.SyncPush is meant to be run as an automated process most of the time. The automated Data Waiter process pulls sync server name (target server), sync database name (the target database), and port from the Minion.SyncServer table.

Adding or repairing a sync partner is a manual process. In that case, you would supply all the parameters to Minion.SyncPush, including @Process='All', to push all existing records to the target server.

For more information, see [“How to: Synchronize backup settings and logs among instances”](#) and [“About: Synchronizing settings and log data with the Data Waiter”](#).

Name	Type	Description
@Tables	Varchar	The category of table that you want to sync: log tables, settings tables, or both. Note: NULL is equivalent to All.

		Valid inputs: NULL All Logs Settings
@SyncServerName	Varchar	This is the name of the target server you want to push the data to. Note that this parameter accepts a single server name, not a delimited list.
@SyncDBName	Varchar	This is the name of the database on the new server that holds the Minion tables.
@Port	Varchar	The port to be used for the connection to the new SQL Server.
@Process	Varchar	Which records to you want to process: just the new ones, or all of them. Most of the time, you will want to run with “New”. “All” is used for bringing on new servers when you want to push all the records in the table to that server. Valid inputs: All New
@Module		Valid inputs: Backup

Overview of Jobs

When you install Minion Backup, it creates two new jobs:

- *MinionBackup-Auto* – Runs every half hour. This job consults the **Minion.BackupSettingsServer** table to determine what, if any, backups are slated to run at that time. By default, the **Minion.BackupSettingsServer** table is configured with Saturday full backups, daily weekday differential backups, and log backups every half hour.
- *MinionBackup-StatusMonitor* – Monitor job that updates the log tables with “backup percentage complete” data. By default, this job runs continuously, updating every 10 seconds, while a Minion Backup operation is running.

“About” Topics

The “About” topics document more detailed information about various segments of Minion Backup.

About: Backup Schedules

Minion Backup offers you a choice of scheduling options:

- You can use the Minion.BackupSettingsServer table to configure flexible backup scheduling scenarios;
- Or, you can use the traditional approach of one job per backup schedule;
- Or, you can use a hybrid approach that employs a bit of both options.

For more information, see “[Changing Schedules](#)” in the Quick Start section, and “[How to: Change backup schedules](#)”.

Table based scheduling

When Minion Backup is installed, it uses a single backup job to run the stored procedure Minion.BackupMaster with no parameters, every 30 minutes. When the Minion.BackupMaster procedure runs without parameters, it uses the Minion.BackupSettingsServer table to determine its runtime parameters (including the schedule of backup jobs per backup type). This is how MB operates by default, to allow for the most flexible backup scheduling with as few jobs as possible.

Table based scheduling presents multiple advantages:

- **A single backup job** – Multiple backup jobs are, to put it simply, a pain. They’re a pain to update and slow to manage, as compared with using update and insert statements on a table.
- **Fast, repeatable configuration** – Keeping your backup schedules in a table saves loads of time, because you can enable and disable schedules, change frequency and time range, etc. all with an update statements. This also makes standardization easier: write one script to alter your backup schedules, and run it across all Minion Backup instances (instead of changing dozens or hundreds of jobs).
- **Mass updates across instances** – With a simple PowerShell script, you can take that same script and run it across *hundreds* of SQL Server instances, standardizing your entire enterprise all at once.
- **Transparent scheduling** – Multiple backup jobs tend to obscure the backup scenario, because each piece of the configuration is displayed in separate windows. Table based scheduling allows you to see all aspects of the backup schedule in one place, easily and clearly.
- **Boundless flexibility** – Table based scheduling provides an amazing degree of flexibility that would be very troublesome to implement with multiple jobs. With one job, you can schedule all of the following:
 - System full backups three days a week.
 - User full backups on weekend days and Wednesday.
 - DB1 log backups between 7am and 5pm on weekdays.
 - All other user log backups between 1am and 11pm on all days.
 - Differential backups for DB2 at 2am and 2pm.
 - Read only backups on the first of every month.

- ...and each of these can also use dynamic backup tuning, which can also be slated for different file sizes, applicable at different times and days of the week and year.
- ...and each of these can also stripe across multiple files, to multiple locations, and/or copy to secondary locations, and/or mirror to a secondary location.

Parameter Based Scheduling

Other SQL Server native backup solutions traditionally use one backup job per schedule. Typically and at a minimum, that means one job for system database full backups, one job for user database full backups, and one job for log backups.

Note: Whether you use table based or parameter based scheduling, we *highly* recommend always using the Minion.BackupMaster stored procedure to run backups. While it is possible to use Minion.BackupDB to execute backups, doing so will bypass much of the configuration and logging benefits that Minion Backup was designed to provide.

About: Backup file retention

The backup file deletion cycle is this:

1. Backup file retention settings are configured in the Minion.BackupSettingsPath table.
2. Each time Minion Backup takes a backup, it logs one row per backup file in the Minion.BackupFiles table. These rows include, among other data, the RetHrs (retention in hours) field for that file.
3. The procedure Minion.BackupFilesDelete runs with every backup operation; it checks the Minion.BackupFiles table to see which files should be deleted. And, of course, it deletes them.

IMPORTANT: As the RetHrs field in Minion.BackupSettingsPath is just the configuration value, not the *configured* retention value. In other words, **updating the RetHrs field in Minion.BackupSettingsPath has *no* effect on the existing backup files' retention settings;** that field only sets the retention for future backup files.

If you reduce the RetHrs value in Minion.BackupSettingsPath, and would like it to also apply to the existing backup files (regardless of their current retention settings), you have two options:

- Use Minion.BackupFilesDelete with a custom retention, or
- Update the Minion.BackupFiles log table.

Minion.BackupFilesDelete procedure: You can call the Minion.BackupFilesDelete stored procedure for your specified database – or, for @DBName='All' – and pass in a specific retention hours using the @RetHrs parameter. For example, to delete all YourDatabase backup files – full, diff, and log – older than 24 hours, run the following:

```
EXEC [Minion].[BackupFilesDelete]
    @DBName = 'YourDatabase',
    @RetHrs = 24 ,
    @Delete = 1 ;
```

Minion.BackupFiles table: Update RetHrs in the Minion.BackupFiles table manually for that database. For example:

```
UPDATE Minion.BackupFiles
SET RetHrs = 24
WHERE DBName = 'YourDatabase';
```

Then, you can either call Minion.BackupFilesDelete manually, or wait for it to run as scheduled.

About: Synchronizing settings and log data with the Data Waiter

Minion Backup provides a “Data Waiter” feature, which syncs backup settings and backup logs between instances of SQL Server. This is especially useful in failover situations – for example, Availability Groups, replication scenarios, or mirrored partners – so that all the latest backup settings and logs are available, regardless of which node is the primary at any given time.

Note: This feature is informally known as the *Data Waiter*, because it goes around and gives data to all of your destination tables. (*Get it??*)

For detailed instructions on configuring the Data Waiter, see [“How to: Synchronize backup settings and logs among instances”](#).

IMPORTANT: When you enable log sync or settings sync for a schedule, it becomes possible for the Data Waiter to cause the backup job to run very long, if there are synchronizing commands that fail (for example, due to a downed sync partner). Consider setting the timeout to a lower value in Minion.SyncServer, to limit the amount of time that the Data Waiter will wait.

Moving Parts

A complete Data Waiter scenario has several moving parts on the primary instance:

- **The Minion.SyncServer table** allows you to configure synchronization partners (i.e., server to which you would like the primary instance to share data).
- **The fields “SyncLogs” and “SyncSettings” in the Minion.BackupSettingsServer table** allow you to enable log and/or settings synchronization for one or more schedules. So, if you enable SyncSettings on a weekly schedule, your settings will be synchronized weekly; enable log settings on a log backup schedule that runs hourly, and the log settings will synchronize hourly.
- **The Minion.BackupSyncLogs procedure** loads INSERT/UPDATE/DELETE statements, designed to bring log data up to date, to the Minion.SyncServer table.
- **The Minion.BackupSyncSettings procedure** loads a snapshot of the settings data (TRUNCATE / INSERT) to the Minion.SyncServer table.
- **The Minion.SyncCmds table** holds the synchronization commands that are to be pushed to sync partners.

- **The Minion.SyncPush procedure** pushes data to sync partners. We use this to initialize the sync partner in the beginning; and Minion Backup uses it to keep sync partners up to date.
- **The Minion.SyncErrorCmds table** holds synchronization commands that failed to push to sync partners. In tandem with the Minion.SyncCmds “ErroredServers” field, Minion.SyncErrorCmds allows the Data Waiter to retry only those statements that failed, and only on those sync partners where they failed.

When enabled and set up, the Data Waiter synchronizes the following tables among configured instances:

- **all settings tables, except** the Minion.SyncServer table (because that table’s data is only applicable on the current instance).
- **all log tables, except:**
 - Minion.BackupDebug
 - Minion.BackupDebugLogDetails
 - Minion.BackupHeaderOnlyWork
 - Minion.SyncCmds
 - Minion.SyncErrorCmds
 - Minion.Work

Use Cases

There are many situations where the Data Waiter feature will be very useful. The primary use case is in any HA/DR scenario where it is possible to “fail over” to another instance. A few of these use cases:

- Four instances that host Availability Group replicas, where a secondary replica may become primary.
- A database mirrored across two instances.
- Several databases that are log shipped to a warm standby server.
- A set of databases replicated to several subscriber servers.
- A HA scenario using third party software, which involves multiple instances.

In each of these cases, the Data Waiter provides an additional layer of transparency to the failover process. After failover, you do not have to reconfigure the backup settings, nor to make sure that old backup files are deleted (so long as the backups are going to UNC).

IMPORTANT: We highly recommend backing up to UNC paths, instead of to locally defined drives. If you have backups going to UNC, and your HA/DR scenario fails over to another server, that server can continue backing up to (and deleting old files from) that same location. Conversely, if Minion Backup is configured to back up locally, it will not be able to delete files from the previous location.

After a failover, you *should* configure the new primary server’s Minion.SyncServer table to point to the other sync partner(s) in the Data Waiter scenario. This is very like a log shipping “failover”, where – once you have failed over to the secondary node – you need to set up log shipping in the other direction.

Failure Handling

In a Data Waiter scenario, if a synchronization partner becomes unavailable over the short term, Minion Backup will track those entries that failed to synchronize. Each time Minion.SyncPush runs, it will attempt to push the failed entries to the downed server. So when the instance becomes available again, the Data Waiter will roll through the changes to bring the sync partner back up to date.

IMPORTANT: Settings and log data that fail to sync through the Data Waiter, do not obstruct the system in any way (though it may somewhat slow the Data Waiter process over time). For example, the Data Waiter may fail to push a command to Server1, but it will still push that command (and future ones) to Server2. The Data Waiter simply tracks the commands that did not sync to Server1 and continues to retry them against that instance, either until they succeed, or until they become outdated and are archived.

Let's take a look at different failed sync scenarios:

- Commands that fail to sync to all sync partners will have Pushed = 0, and ErroredServers = <a comma-delimited list of all sync partners to which the push failed> in Minion.SyncCmds.
- Commands that fail to sync to some, but not all, sync partners will have Pushed = 1, and ErroredServers = <a comma-delimited list of all sync partners to which the push failed> in Minion.SyncCmds.
- Any command that failed to synchronize to one or more partners will have an entry in Minion.SyncErrorCmds.

If a synchronization partner becomes unavailable over a long period of time, we advise that you disable the Data Waiter for that instance, and reinitialize it as if it were a new sync partner when it again becomes available. The reason for this is, after even a week or two passes, it is more efficient to set up the partner again, instead of rolling through all the changes that have accumulated.

Enabling Data Waiter while using parameter based scheduling

Minion Backup uses table based scheduling by default, which retrieves schedule and other server-level settings from the Minion.BackupSettingsServer table. In fact, the Data Waiter settings and log synchronization options are enabled in Minion.BackupSettingsServer.

If you choose to use parameter based scheduling instead of table based, then the Data Waiter will not run automatically. You must instead set up synchronization as you normally would, and then create a job to run the Data Waiter stored procedures. Check www.MinionWare.net for additional instructions.

For more information on the Data Waiter process, see "[How to: Synchronize backup settings and logs among instances](#)".

About: Dynamic Backup Tuning Thresholds

In SQL Server, we can adjust high level settings to improve server performance. Similarly, we can *adjust settings in individual backup statements* to improve the performance of backups themselves. A backup tuning

primer is well beyond the scope of this document; to learn about backup tuning, please see the recording of our Backup Tuning class at <http://bit.ly/1O6Rsh3> (download demo code at <http://bit.ly/1Os6yzz>).

Introduction

Once you are familiar with the backup tuning process, you can perform an analysis, and then set up specific thresholds in the `Minion.BackupTuningThresholds` table. It is a “Thresholds” table, because you cannot tune a backup once and disregard database growth; backup tuning settings must change as a database grows. So, Minion Backup allows you to configure a different collection of backup tuning settings for different sized databases (thereby, defining backup tuning thresholds). **As your database grows and shrinks, Minion Backup will use the settings you’ve defined for those sizes**, so that backups always stay at peak performance.

Note: You can get more specific information about the `Minion.BackupTuningThresholds` table in the “[Minion.BackupTuningThresholds](#)” section.

As a small example, here is a limited rowset for `Minion.BackupTuningThresholds`, which shows different backup tuning settings for a single database at various sizes, and for two different backup types:

DBName	Backup Type	Space Type	Threshold Measure	Threshold Value	NumberOf Files	Buffer count	MaxTransferSize
DB1	Full	DataAnd Index	GB	0	2	30	1048576
DB1	Full	DataAnd Index	GB	50	5	50	2097152
DB1	Diff	DataAnd Index	GB	0	2	30	1048576
DB1	Log	Log	GB	0	1	15	1048576

This sample data shows two threshold levels for DB1 full backups: one for databases larger than 50GB, and one for databases above 0GB. Note that the threshold value is a “floor” threshold: so, if DB1 is 25GB, it will use the 0GB threshold settings; if it is 60GB, it will use the 0GB threshold settings. The sample data also shows just one threshold level each for DB1 log backups and DB1 differential backups.

Of course, we could add additional rows for each type, for different size thresholds. This is what puts the “dynamic” in “dynamic backup tuning”; Minion Backup will automatically change to the new group of settings when your database passes the defined threshold.

Enabled by Default

Default backup tuning settings are in effect the moment that Minion Backup is installed: the system comes installed with a default “MinionDefault” row in `Minion.BackupTuningThresholds`. These backup tuning settings are used for **any database which does not have a specific set of thresholds defined for it**; as well as for **any database that has dynamic tuning disabled in `Minion.BackupSettings`**.

While this last point may seem inconsistent – after all, why should a database refer to the “MinionDefault” row in this table if dynamic tuning is disabled? – in fact, it makes perfect sense:

- First, the default backup tuning settings cannot truly be said to be “dynamic”, as the dynamic aspect of backup tuning comes from having different settings for a database come into effect automatically as the database grows. The MinionDefault row in this table has a threshold size of 0GB, and so applies to databases of all sizes.
- Second, most of the settings in the MinionDefault row are “passive”: NumberOfFiles is 1, which is the case for any backup where number of files is not specified. And Buffercount, MaxTransferSize, and BlockSize are zero, meaning SQL Server is free to choose the appropriate value for these settings at the time the backup runs.

Essential Guidelines

There are three essential guidelines for setting dynamic backup tuning thresholds in Minion Backup:

- **Any group of tuning thresholds – whether it is the MinionDefault group of settings, or a database-specific group of settings – must have one row with a “floor” setting of zero.**
- **Once you have defined a single database-specific row, all backup types for that database must be represented in one or more rows.** (Note that each backup type must also, therefore, have a “floor” threshold of zero represented.) For more information about this rule, see [“The Configuration Settings Hierarchy Rule”](#) in the [“Architecture Overview”](#) section.
- **However, if there is a hole in your backup tuning threshold settings, the MinionDefault row acts as a failsafe.** It is best to define your backup tuning settings thoughtfully and with foresight; but the failsafe is there, just in case of oversights. (This failsafe is the exception to The Configuration Settings Hierarchy Rule; no other table can rely on the MinionDefault row in this way.)

Important Backup Tuning Concepts

Here is a quick review of important backup tuning threshold concepts in Minion Backup:

- **Tune your own:** The settings we use for these examples are just that: examples. They are not recommendations, and have no bearing on your particular environment. We DO NOT recommend using the example number in this document, without proper analysis of your particular system.
- **Default Settings:** Minion Backup is installed with a default backup tuning threshold setting, defined by the row DBName='MinionDefault', BackupType='All', and ThresholdValue=0. These settings are in effect for any database with DynamicTuning enabled in the Minion.BackupSettings.
- **Space Types:** You have the option of basing our tuning thresholds on data size only, on data and index size, or on file size. File size includes any unused space in the file; “data and index” does not.
- **Available Data:** Minion Backup is a huge help to your analysis, because it gathers and records the backup settings for EVERY backup (including Buffercount, MaxTransferSize, etc.) in Minion.BackupLogDetails, whether or not it was a tuned backup.

- **Floor Thresholds:** The thresholds in Minion.BackupTuningThresholds represent the *LOWER* threshold (the “floor”). Therefore, you must be sure to enter a threshold for file size 0.
- **Settings Precedence:** Minion Backup has a hierarchy of settings, where the most specific setting takes precedence. See the “Backup Tuning Threshold Precedence” section below.

Backup Tuning Threshold Precedence

Minion Backup has a hierarchy of settings, where the most specific setting takes precedence. The precedence for backup tuning threshold settings is as follows:

Precedence Level	DBName	Backuptype
<i>Highest</i>	DB1	Full, or Diff, or Log
<i>High</i>	DB1	All
<i>Low</i>	MinionDefault	Full, or Diff, or Log
<i>Lowest</i>	MinionDefault	All

Note: If you define a database-specific row, we highly recommend that you provide tuning settings for all backup types, for that database. For example, if you insert one row for YourDatabase with backup type Full, you should also insert a row for YourDatabase and backup type All (or two additional rows, one each for differential and log).

Let’s look at an example set of backup tuning threshold settings:

ID	DBName	BackupType	isActive
1	MinionDefault	All	1
2	MinionDefault	Full	1
3	MinionDefault	Log	1
4	DB1	All	1
5	DB1	Full	1

Using these settings, let’s look at which settings will be used when:

- For a DB1 full backup, Minion Backup will use row 5: DBName=DB1, BackupType=Full.
- For a DB1 differential or log backup, Minion Backup will use row 4: DBName=DB1, BackupType=All.
- For a DB2 full backup, Minion Backup will use row 2 (DBName=MinionDefault, BackupType=Full).
- For a DB2 differential backup, Minion Backup will use row 1 (DBName=MinionDefault, BackupType=All).

Note: If you are unsure of what backup tuning settings will be used, you can double check; use the Minion.BackupStmtGet stored procedure, which will build (but not run) the backup statement for you. For more information, see “[Minion.BackupStmtGet](#)”.

Business Aware Dynamic Backup Tuning

What’s more, Minion Backup’s dynamic backup tuning can be made “business aware”, in a sense. For example, configure one set of tuning thresholds for weekday business hours, and another set for after hours

and weekends. Or, perhaps you need a different set of configurations for Monday, because that's the busiest day.

Here is a high-level overview of one way to set up "business aware" backup tuning scenarios:

1. Perform your backup tuning analysis, and determine the settings for two scenarios:
 - a. one low-resource scenario for times when the server is busy (say, weekdays); and
 - b. one high-resource scenario for when the server is largely unused (e.g., on the weekend).
2. Insert rows to `Minion.BackupTuningThresholds` for the low-resource scenario, and set `IsActive=1`.
3. Insert additional rows to `Minion.BackupTuningThresholds` for the high-resource scenario, and set `IsActive=0`.
4. Set up your backup routine with precode that checks the day of the week;
 - a. If the day is Saturday or Sunday, the precode sets `isActive=1` in `Minion.BackupTuningThresholds` for the high-resource scenario, and `isActive=0` for the low-resource scenario.
 - b. Otherwise, the precode enables the low-resource scenario, and disables the high-resource scenario.

Tuning Log Backups

Log backups are interesting, because the size of the *database* doesn't matter for a log file backup. If your database is small, but a process has blown the log up to a huge size, the size of the *data* file has no impact whatsoever on the log backup. You need to perform a backup tuning analysis for log file backups, just like for any other backup type. After all, you wouldn't want to back up a 5MB log file to 10 files!

Any time you have a row in `Minion.BackupTuningThresholds` with `BackupType = 'Log'`, Minion Backup will automatically use the **space used in the log** as the measure for "SpaceType". So for example, if you have a 100GB log file that is 10% used, the space used in the log file is 10GB; Minion Backup uses this measure – the 10GB – to determine when the threshold should change.

Though the value of `SpaceType` does not change anything in regards to log backups, we still recommend you set `SpaceType` equal to "Log" whenever the `BackupType = 'Log'`, because it is a visual reminder of how the threshold is calculated.

This feature is meant to keep a huge log from taking hours to process, while other logs are filling up (because they can't back up yet because of the big one). So, keep a safety net for yourself, and put in a couple tuning options for your logs. If they grow really big, the payoff of tuned log backups is considerable; well-tuned log backups take a fraction of the time they ordinarily would.

Note: The backup tuning thresholds feature does not shrink the log file. To shrink the log file, see the three "ShrinkLog%" columns in the `Minion.BackupSettings` table. These two features – Dynamic Backup Tuning and Shrink Log on Log Backup – work very well together to keep your system running without intervention from you. (You're welcome!) For more information on shrinking the log, see "[How to: Shrink log files after log backup](#)".

FAQ

Why does Minion Backup use xp_cmdshell instead of SQL CLR?

First, it would be a burden to require users to have CLR installed on every single server on the network. Not only that, but the database setting would have to be set to UNTRUSTWORTHY for the things MB needs to do; or else, we would have a far more complex scenario on hand, and that level of complication just for backups is not a good setup.

Using SQL CLR would also put us in the business of having to support different .NET framework versions, which would also complicate things.

Cmdshell is the best choice because it's simple to lock down to only administrators, and it adds no extra "gotchas". There were times when it would have been easier to use CLR, but we simply can't require that everyone enables CLR.

Just be sure to lock down cmdshell. For further reading, here is the link to one of Sean's rants on the topic: <http://www.midnightdba.com/DBARant/?p=1204>

Why should I run Minion.BackupMaster instead of Minion.BackupDB?

We HIGHLY recommend using Minion.BackupMaster for all of your backup operations, even when backing up a single database. To explore the "why", let's look at each of the two procedures briefly.

- The Minion.BackupDB stored procedure creates and runs the actual backup statement for a single database, using the settings stored in the Minion.BackupSettings table.
- The Minion.BackupMaster procedure makes all the decisions on which databases to back up, and what order they should be in. It calls Minion.BackupDB to perform a backup per database, within a single backup batch.

So why run Minion.BackupMaster?

- It unifies your code, and therefore minimizes your effort. By calling the same procedure every time you reduce your learning curve and cut down on mistakes.
- Future functionality may move to the Minion.BackupMaster procedure; if you get used to using Minion.Backup Master now, then things will always work as intended.
- Minion.BackupMaster takes advantage of rich include and exclude functionality, including regular expressions, like expressions, and comma-delimited lists. Even better, when run without parameters, it takes advantage of rich table-based scheduling and all the benefits associated.

- The master SP performs extensive logging, and it enables Live Insight via the status monitor job (which updates each backup percentage complete as it runs).
- Minion.BackupMaster runs configured pre- and postcode, determines AG backup location, performs file actions (such as copy and move), and runs the Data Waiter feature to synchronize log and settings data across instances.

In short, Minion.BackupMaster decides on, runs, or causes to run every feature in Minion Backup. Don't shortcut your features list by running Minion.BackupDB. Use Minion.BackupMaster!

Why must I supply values for all backup types for a database in the settings tables?

Several settings tables – including Minion.BackupSettings, Minion.BackupSettingsPath, and Minion.BackupTuningThresholds – provide a MinionDefault / All row to provide settings for databases that do not have specific settings defined. In this way, there is a base level default that allows Minion Backup to function immediately upon installation.

We made a design decision to “keep in the scope” once any database-specific settings were defined. In other words, once the configuration context is at the database level, it stays at the database level. Therefore, if you define a database-specific row, you must be sure that all backup types are represented for that database.

The reasoning behind this rule is this: It takes a conscious act (inserting a row) to change settings for a specific database. So, we don't want the system to “fall back” on default values, possibly countermanding the intended configuration for that particular database.

For more information, see “[Backup tuning threshold precedence](#)”.

Why isn't my log backup working for this database?

The most likely causes are:

- The database could be in the wrong recovery mode; or
- The database has never had a full backup before; or
- The backups are misconfigured.

Recovery mode: Only databases in full or bulk logged mode allow log backups. Check that your database is in either full or bulk logged mode. For more information on SQL Server recovery models, see <https://msdn.microsoft.com/en-us/library/ms189275.aspx>

Full backups: In SQL Server, a database must have had a full backup before a log backup can be taken. So Minion Backup prevents this: if you try to take a log backup, and the database doesn't have a restore base, then the system will remove the log backup from the list. It will not attempt to take a log backup until there's a full backup in place. Though it may seem logical to perform a full backup instead of a full, we do not do this, because log backups can be taken very frequently; we don't want to make what is usually a quick operation into a very long operation.

Other: If neither of these is the issue try the following:

- Check the Minion.BackupLog and Minion.BackupLogDetails to see if log backups are being attempted and failing, for this database.
- Check Minion.BackupSettings to be sure that either (a) the database in question has rows defined to cover all backup types, or (b) the database has NO database-specific rows defined, and therefore will use the MinionDefault settings.
- Check Minion.BackupSettingsPath to be sure that (a) the database in question has rows defined to cover all backup types, or (b) the database has NO database-specific rows defined, and therefore will use the MinionDefault settings.

And as always, get support from us at www.MinionWare.net if you need it.

Why isn't my log file shrinking after a log backup?

Make sure that you've set all three of the "Shrink" fields in Minion.BackupSettings for the proper database and backup type. (We often find that when a log file won't shrink after log backup, it's because the "Shrink" fields were configured for BackupType='Full' instead of 'All' or 'Log').

Why doesn't Minion Backup offer a "shrink data file after full backup" feature?

This is by design. Shrinking the data file is not recommended.

Why isn't MB using my backup tuning thresholds?

There are a few possibilities:

- Check the log of the latest backups for your database in Minion.BackupLogDetails. Compare the logged backup tuning values that were used (NumberOfFiles, Buffercount, MaxTransferSize, and Compression) against the settings you expect to be used from Minion.BackupTuningThresholds.
- Check if you have disabled dynamic tuning for that database, or for all databases. Check the DynamicTuning column in Minion.BackupSettings.
- Perhaps you have not set a threshold that includes your database at its present size. Check Minion.BackupTuningThresholds to determine that:
 - rows are defined for your database (DBName, BackupType)
 - the rows for your data the appropriate rows are active (IsActive=1),
 - your database is larger than the threshold you're expecting it to use (SpaceType, ThresholdMeasure, ThresholdValue). One common mistake is to omit a "floor" value of zero for a particular database; this causes that database to use the MinionDefault values in Minion.BackupTuningThresholds, instead.

Can I back up to Azure?

Yes, you can back up to Azure. Currently, Minion Backup can't *copy or move* files to or from Microsoft Azure Blobs. However, you can do a primary backup to an Azure Blob.

Minion Backup cannot delete files or create directories on Azure Blobs.

You guys are the MidnightDBAs, and you run MidnightSQL. What's with "MinionWare"?

MidnightDBA is the banner for our free training. MidnightSQL Consulting, LLC is our actual consulting business. And now, we've spun up MinionWare, LLC as our software company. We released our new SQL Server management solution, Minion Enterprise, under the MinionWare banner. And now, all the little Minion guys will live together on www.MinionWare.net.

Minion Reindex, Minion Backup, and other Minion modules are, and will continue to be free. Minion Enterprise is real enterprise software, and we'd love the chance to prove to you that it's worth paying for. Get in touch at www.MinionWare.net and let's do a demo, and get you a free 90 day trial!

About Us

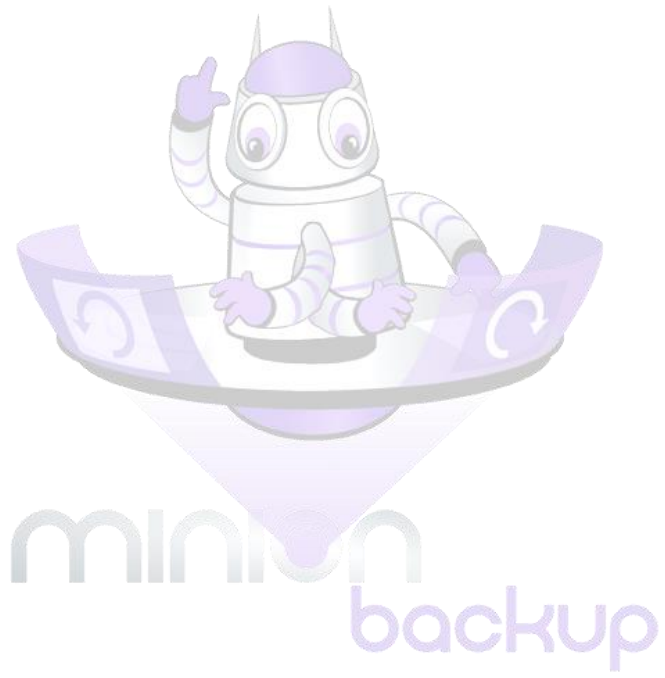
Minion by MidnightDBA is a creation of Jen and Sean McCown, owners of MinionWare, LLC and MidnightSQL Consulting, LLC.

We formed **MinionWare**, LLC to create **Minion Enterprise**: an enterprise management solution for centralized SQL Server management and alerting. This solution allows your database administrator to manage an enterprise of one, hundreds, or even thousands of SQL Servers from one central location. Minion Enterprise provides not just alerting and reporting, but backups, maintenance, configuration, and enforcement. **Go to www.MinionWare.net for details and to request a free 90 day trial.**

In our "**MidnightSQL**" consulting work, we perform a full range of databases services that revolve around SQL Server. We've got over 30 years of experience between us and we've seen and done almost everything there is to do. We have two decades of experience managing large enterprises, and we bring that straight to you. Take a look at www.MidnightSQL.com for more information on what we can do for you and your databases.

Under the "**MidnightDBA**" banner, we make free technology tutorials, blogs, and a live weekly webshow (DBAs@Midnight). We cover various aspects of SQL Server and PowerShell, technology news, and whatever else strikes our fancy. You'll also find recordings of our classes – we speak at user groups and conferences internationally – and of our webshow. Check all of that out at www.MidnightDBA.com

We are both "MidnightDBA" and "MidnightSQL" ...the terms are nearly interchangeable, but we tend to keep all of our free stuff under the MidnightDBA banner, and paid services under MidnightSQL Consulting, LLC. Feel free to call us the MidnightDBAs, those MidnightSQL guys, or just "Sean" and "Jen". We're all good.



Contents

Quick Start.....	1
Change Schedules	2
Table based scheduling.....	2
Change Default Settings.....	4
Top 20 Features	6
Architecture Overview	8
Configuration Settings Hierarchy.....	8
The Configuration Settings Hierarchy Rule	9
Example 1: Proper Configuration	9
Example 2: Improper Configuration	9
Example 3: The “Exclude” Exception.....	10
Include and Exclude Precedence	10
Include and Exclude strings	11
Exclude bit.....	11
Run Time Configuration.....	12
Logging	13
Alerting.....	13
“How To” Topics: Basic Configuration.....	15
How To: Configure settings for a single database.....	15
How To: Configure settings for all databases.....	17
How To: Back up databases in a specific order	18
How To: Change backup schedules	21
Table based scheduling.....	21
Parameter based scheduling (traditional approach)	21
Hybrid scheduling	22
How To: Generate back up statements only.....	22
How To: Back up only databases that are not marked READ_ONLY	23
How To: Include databases in backups	23
Include databases in table based scheduling	23

Include databases in traditional scheduling.....	24
How To: Exclude databases from backups.....	26
Exclude a database from all backups.....	26
Exclude databases in table based scheduling.....	27
Exclude databases in traditional scheduling.....	28
How To: Run code before or after backups.....	29
Database precode and postcode.....	29
Batch precode and postcode.....	31
How To: Configure backup file retention.....	32
“How To” Topics: Backup Mirrors and File Actions.....	33
How to: Set up mirror backups.....	33
How to: Copy files after backup (single and multiple locations).....	36
How to: Move files to a location after backup.....	39
How to: Copy and move backup files.....	41
How to: Back up to multiple files in a single location.....	41
How to: Back up to multiple locations.....	44
“How To” Topics: Advanced.....	47
How to: Install Minion Backup across multiple instances.....	47
How to: Shrink log files after log backup.....	48
How to: Configure certificate backups.....	49
How to: Encrypt backups.....	50
Encrypt backups for one database.....	51
Encrypt backups for all databases.....	52
How to: Synchronize backup settings and logs among instances.....	53
Example: Data Waiter serves one partner.....	53
Example: Data Waiter serves Availability Group members.....	55
How to: Set up backups on Availability Groups.....	57
How to: Set up dynamic backup tuning thresholds.....	60
Example 1: Modify existing, default tuning thresholds.....	60
Example 2: Tune backups for one database based on file size.....	61
Example 3: Tune backup types for all databases based on data + index size.....	63

Moving Parts	66
Overview of Tables	66
Settings Tables Detail.....	67
Minion.BackupCert	67
Minion.BackupEncryption	67
Minion.BackupSettings	68
Minion.BackupSettingsPath.....	76
Minion.BackupSettingsServer.....	79
Minion.BackupTuningThresholds.....	85
Minion.DBMaintRegexLookup.....	87
Minion.SyncServer	87
Log Tables Detail	89
Minion.BackupDebug.....	89
Minion.BackupDebugLogDetails.....	89
Minion.BackupFileListOnly	89
Minion.BackupFiles.....	90
Minion.BackupHeaderOnlyWork.....	92
Minion.BackupLog	92
Minion.BackupLogDetails	94
Minion.SyncCmds	99
Minion.SyncErrorCmds	100
Minion.Work	101
Overview of Procedures	101
Procedures Detail.....	102
Minion.BackupDB	102
Minion.BackupFileAction	103
Minion.BackupFilesDelete	103
Minion.BackupMaster	106
Minion.BackupRestoreDB.....	107
Minion.BackupSyncLogs	108
Minion.BackupSyncSettings.....	109

Minion.BackupStatusMonitor	109
Minion.BackupStmtGet	109
Minion.CloneSettings.....	111
Minion.HELP.....	111
Minion.SyncPush.....	112
Overview of Jobs.....	113
“About” Topics.....	113
About: Backup Schedules	114
Table based scheduling.....	114
Parameter Based Scheduling.....	115
About: Backup file retention	115
About: Synchronizing settings and log data with the Data Waiter.....	116
Moving Parts.....	116
Use Cases	117
Failure Handling.....	118
Enabling Data Waiter while using parameter based scheduling	118
About: Dynamic Backup Tuning Thresholds.....	118
Introduction	119
Enabled by Default.....	119
Essential Guidelines	120
Important Backup Tuning Concepts	120
Backup Tuning Threshold Precedence	121
Business Aware Dynamic Backup Tuning	121
Tuning Log Backups.....	122
FAQ.....	123
Why does Minion Backup use xp_cmdshell instead of SQL CLR?.....	123
Why should I run Minion.BackupMaster instead of Minion.BackupDB?	123
Why must I supply values for all backup types for a database in the settings tables?	124
Why isn't my log backup working for this database?	124
Why isn't my log file shrinking after a log backup?	125
Why doesn't Minion Backup offer a “shrink data file after full backup” feature?.....	125

Why isn't MB using my backup tuning thresholds? 125
Can I back up to Azure? 126
You guys are the MidnightDBAs, and you run MidnightSQL. What's with "MinionWare"? 126
About Us..... 126

