[Home](https://www.sqlserverscience.com/) » [Internals](https://www.sqlserverscience.com/internals/) » Adding a New Transaction Log File when the Existing Log is Full

Adding a New Transaction Log File when the Existing Log is Full

**2019-03-19 · by** [**Max Vernon**](https://www.sqlserverscience.com/author/admin/)

**FROM:** <https://www.sqlserverscience.com/internals/adding-new-transaction-log-file-when-existing-log-full/>

SQL Server Transaction Log files store details about *every* change made to a SQL Server Data File. This includes when you need to add a 2nd log file. The act of adding a 2nd log file changes data in the primary filegroup for the database. Since *every* change is logged, *that change* is logged too. So, what happens if you completely run out of log file space? Can you add a 2nd log file to avoid calamity? Turns out you can.

[[](https://www.sqlserverscience.com/wp-content/uploads/2019/03/winter-scene.png)<img aria-describedby="caption-attachment-1321" src="/wp-content/uploads/2019/03/winter-scene-300x199.png" alt="Not Adding Transaction Log Files" width="300" height="199" class="size-medium wp-image-1321" srcset="https://i0.wp.com/www.sqlserverscience.com/wp-content/uploads/2019/03/winter-scene.png?resize=300%2C199&amp;ssl=1 300w, https://i0.wp.com/www.sqlserverscience.com/wp-content/uploads/2019/03/winter-scene.png?resize=768%2C509&amp;ssl=1 768w, https://i0.wp.com/www.sqlserverscience.com/wp-content/uploads/2019/03/winter-scene.png?w=892&amp;ssl=1 892w" sizes="(max-width: 300px) 100vw, 300px" />](https://www.sqlserverscience.com/wp-content/uploads/2019/03/winter-scene.png)

     What does this have to do with transaction log files?  
           Not much; but that’s a pretty big log!

SQL Server 2008+ sets aside some space inside the most recently used [virtual log file](https://www.sqlserverscience.com/recovery/optimal-log-file-growth-and-virtual-log-files/), or VLF, such that you are able to add another log file. The code below shows that in action.

First, we’ll create a database with a 10MB log file that we can fill up:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | USE master;  IF DB\_ID(N'LogGrowthTest') IS NOT NULL  BEGIN      ALTER DATABASE LogGrowthTest SET SINGLE\_USER WITH ROLLBACK IMMEDIATE;      DROP DATABASE LogGrowthTest;  END  CREATE DATABASE LogGrowthTest  ON (      NAME = 'PRIMARY'      , FILENAME = 'G:\Data\LogGrowthTest\_primary.mdf'      , SIZE = 100MB      , FILEGROWTH = 100MB      , MAXSIZE = 100MB      )  LOG ON (      NAME = 'LOG'      , FILENAME = 'F:\Data\LogGrowthTest\_logfile.mdf'      , SIZE = 10MB      , FILEGROWTH = 10MB      , MAXSIZE = 10MB      )  COLLATE Latin1\_General\_100\_CI\_AS; |

Next we’ll set the [recovery model](https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/recovery-models-sql-server?utm_medium=referral&utm_source=sqlserverscience.com#RMov) to full:



|  |  |
| --- | --- |
| 1  2  3 | ALTER DATABASE LogGrowthTest SET RECOVERY FULL;  BACKUP DATABASE LogGrowthTest TO DISK = N'NUL:';  GO |

This creates a table, and adds enough data to fill up the log file:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | USE LogGrowthTest;  CREATE TABLE dbo.d  (      dd varchar(8000) NOT NULL  ) ON [PRIMARY];  GO  SET NOCOUNT ON;  ;WITH nums AS  (      SELECT v.n      FROM (VALUES (0), (1), (2), (3), (4), (5), (6), (7), (8), (9))v(n)  )  INSERT INTO dbo.d (dd)  SELECT TOP(1280) REPLICATE(' ',8000)  FROM nums n1      CROSS JOIN nums n2      CROSS JOIN nums n3      CROSS JOIN nums n4; |

Run this code several times until you see a message stating the log file is full:



|  |  |
| --- | --- |
| 1  2 | INSERT INTO dbo.d (dd)  VALUES ('1'); |

The error will look like:

Msg 9002, Level 17, State 2, Line 21

The transaction log for database 'LogGrowthTest' is full due to 'LOG\_BACKUP'.

Now, issue a [checkpoint](https://docs.microsoft.com/en-us/sql/t-sql/language-elements/checkpoint-transact-sql) just for fun:



|  |  |
| --- | --- |
| 1 | CHECKPOINT; |

And, finally, add a new log file:



|  |  |
| --- | --- |
| 1  2 | ALTER DATABASE LogGrowthTest  ADD LOG FILE (NAME = 'LOG2', FILENAME = 'F:\Data\LogGrowthTest\_Logfile2.mdf') |

We can check which VLF has the details for the newly added log file with the undocumented table-valued-function, sys.fn\_dblog. sys.fn\_dblog returns a row for each active, uncleared record in the transaction log. With the database in simple recovery model, you’ll only see rows returned that haven’t been committed and subsequently cleared via [log truncation](https://docs.microsoft.com/en-us/sql/relational-databases/logs/the-transaction-log-sql-server?utm_medium=referral&utm_source=sqlserverscience.com#Truncation). We set the recovery model to full when we created the test database so VLFs don’t get truncated.



|  |  |
| --- | --- |
| 1  2 | SELECT \*  FROM sys.fn\_dblog(null, null) |

Down towards the end of the output we see LOP\_CREATE\_FILE in the “Operation” column, while the “Current LSN” column provides the log sequence number. The LSN looks like:

00000028:00001385:000a

The numbers are in hexadecimal format. The first piece, 00000028, is the VLF File Sequence Number (FSeqNo). This number uniquely identifies the VLF where the action was recorded. The last two pieces aren’t important for this discussion; they consist of the log block, and the log record offset within that block. Anyway, if you look at the log with DBCC LOGINFO, you’ll see that VLF FSeqNo 40 (which is 0x28 in hex – the first piece of the LSN above) exists in FileID 2, which is the file ID from sys.database\_files for the first log file. The output of DBCC LOGINFO for my test with the relevant row highlighted, looks like:

╔════════════════╦════════╦══════════╦═════════════╦════════╦════════╦════════╦═══════════╗

║ RecoveryUnitId ║ FileId ║ FileSize ║ StartOffset ║ FSeqNo ║ Status ║ Parity ║ CreateLSN ║

╠════════════════╬════════╬══════════╬═════════════╬════════╬════════╬════════╬═══════════╣

║ 0 ║ 2 ║ 2555904 ║ 8192 ║ 37 ║ 2 ║ 64 ║ 0 ║

║ 0 ║ 2 ║ 2555904 ║ 2564096 ║ 38 ║ 2 ║ 64 ║ 0 ║

║ 0 ║ 2 ║ 2555904 ║ 5120000 ║ 39 ║ 2 ║ 64 ║ 0 ║

║ 0 ║ 2 ║ 2809856 ║ 7675904 ║ 40 ║ 2 ║ 64 ║ 0 ║

║ 0 ║ 3 ║ 2555904 ║ 8192 ║ 0 ║ 0 ║ 0 ║ 0 ║

║ 0 ║ 3 ║ 2555904 ║ 2564096 ║ 0 ║ 0 ║ 0 ║ 0 ║

║ 0 ║ 3 ║ 2555904 ║ 5120000 ║ 0 ║ 0 ║ 0 ║ 0 ║

║ 0 ║ 3 ║ 2809856 ║ 7675904 ║ 0 ║ 0 ║ 0 ║ 0 ║

╚════════════════╩════════╩══════════╩═════════════╩════════╩════════╩════════╩═══════════╝

You can get the file\_id, name, and physical name of the files with this query:



|  |  |
| --- | --- |
| 1  2  3  4  5 | SELECT df.name      , df.file\_id      , df.physical\_name  FROM sys.database\_files df  ORDER BY df.file\_id |

As you can see, file\_id 2 is LOG, the initial log file, and file\_id 3 is the newly added log file, LOG2.

╔═════════╦═════════╦════════════════════════════════════╗

║ name ║ file\_id ║ physical\_name ║

╠═════════╬═════════╬════════════════════════════════════╣

║ PRIMARY ║ 1 ║ G:\Data\LogGrowthTest\_primary.mdf ║

║ LOG ║ 2 ║ F:\Data\LogGrowthTest\_logfile.mdf ║

║ LOG2 ║ 3 ║ F:\Data\LogGrowthTest\_Logfile2.mdf ║

╚═════════╩═════════╩════════════════════════════════════╝

It’s pretty great that Microsoft allows you to add a log file even when the existing log file is full, however I recommend [pro-actively managing your log file](https://www.sqlserverscience.com/recovery/optimal-log-file-growth-and-virtual-log-files/).

This post is part of our series on [Database Internals](https://www.sqlserverscience.com/internals) and [Database Recovery](https://www.sqlserverscience.com/recovery).

See Microsoft’s Documentation for an [overview of log sequence numbers](https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/recover-to-a-log-sequence-number-sql-server#LSNs).