**Mongo DB Storage Engine**

The *storage engine* is the part of the [MongoDB](http://www.mongodb.org/)server that runs all the CRUD commands and affects how the data is read, written to and removed from disk as well as how it is physically stored on the disk.

In MongoDB 3.0 you now get a second choice of storage engine.

In earlier versions you had the original *MMAP v1* which is still the default storage version in 3.0. But now you have the option to use the *Wired Tiger* storage engine which is the newly supported an open source storage engine that is also used by other databases.

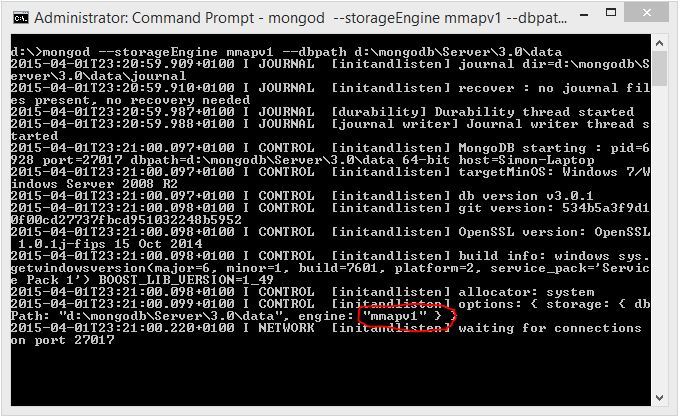
When you start the *mongod* server you can specify which storage engine to use, e.g.

mongod --storageEngine mmapv1

or

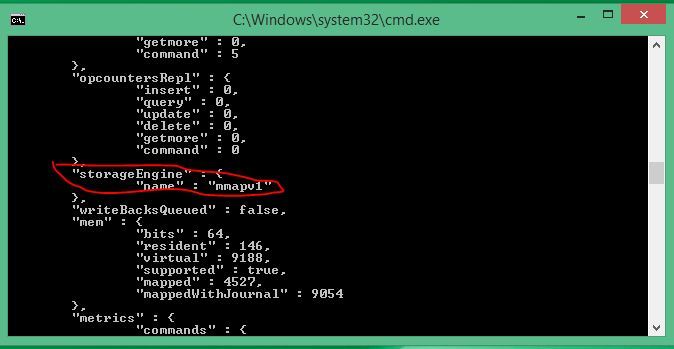
mongod --storageEngine wiredTiger

If you’ve specified a stored engine you’ll see it in the log when *mongod* starts but not if you’re just using the default.

[](https://simonlearningsqlserver.files.wordpress.com/2015/04/011.jpg)

You can also see it by running the following from the shell

db.serverStatus()

[](https://simonlearningsqlserver.files.wordpress.com/2015/04/02.jpg)

You MongoDB instance can only run either MMAP v1 or Wired Tiger. You cannot have both MMAP v1 or Wired Tiger data files in your data path directory.

[](https://simonlearningsqlserver.files.wordpress.com/2015/04/03.jpg)

**MMAP v1**

*MMAP v1* stores the data as BSON and maps the raw BSON directly into virtual memory which allows the operating system to do most the work for the storage engine. If the files are not already in memory then they get pulled into RAM and any updates will get propagated back to disk

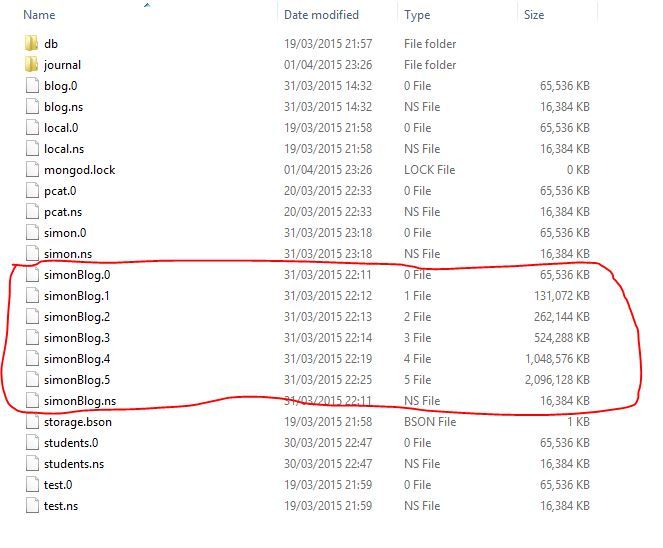
*Collection level locking* is now available in version 3.0 and *database level locking* with version 2.2 to 2.6.

The MongoDB lock is a “*mutliple reader, single writer”* lock which means you can have multiple readers on the data that will lock out all writers, however one writer will lock out all readers and also all other writers.

They don’t have *document level locking* in MMAP v1 as there are other shared resources that can cause contention. For example, if two separate documents exist in the same index, then an update to the first document will result in an update to the index and an update to the second document will result in another update to the index. If the update to both documents occurs simultaneously then we may have issues updating the index.

The *journal* is used a bit like the transaction log in SQL Server. Any update is written to the journal before it is applied to the database. This means that if the file system or the service goes down then the unplayed updates in the *journal*can be performed when everything is back up and running.

As soon as you create a document in a collection in a database a 64MB data file gets created. When the size of the database increases and the original 64 MB file is filled, a new double sized file is created, i.e. a 128MB file. As the database size increases a new data file double the size of the last will be created until the 2GB limit is reached. Once it reaches this limit new 2GB will keep being created.

[](https://simonlearningsqlserver.files.wordpress.com/2015/04/04.jpg)

Prior to version 3.0 MongoDB used to add some padding if it saw documents increasing and would try and guess how much padding to add. In version 3.0 they introduced a *Power of 2* sized allocation similar to what they do with the data files. If a small document less than 32 bytes is inserted it will be allocated a 32B slot in the data file. If the document then grows to 33B it will be moved to a 64B slot. The size of the allocation slot will be a power of 2 up to the 2MB limit and after this more 2MB slots get allocated up to the 16MB document size limit. If a document has to move in the disk then all pointers in any indexes need to be updated, which has a performance overhead. This *power of 2 padding*doesn’t prevent document movement but it can reduce it and a document that has moved will leave behind a standardised slot in the data file for any new document to fit into nicely.

If you have a database where the documents will not grow or you create place holders in the documents to give them a certain size, you can disable the padding by explicitly creating the collection with no padding, i.e.

db.createCollection("simonTest", {noPadding:true})

**Wired Tiger**

There are three main benefits to [*Wired Tiger*](http://www.wiredtiger.com/)

1. Performance gains
2. Document level locking
3. Compression

This is the first release so they are expecting more improvements in MongoDB using Wired Tiger.

Wired Tiger data is stored per collection rather than per database and is stored in B-Trees. Any new documents are added to a separate region of the data files and are moved in with the others in the background. When a document is updated the new version of the document is created as a new document and the old file is removed. Because of this there is no need for padding.  Wired Tiger can apparently perform this with relative ease.

There are two caches: the *Wired Tiger Cache* and the *File System Cache*. All changes to the *Wired Tiger Cache* are moved in to the *File System Cache* via a *checkpoint*. The *checkpoint* runs 60 seconds after the last *checkpoint* ended and the previous but one *checkpoint* is not deleted until the next *checkpoint* has finished. This means that you could turn off the *journal* but you still could potentially lose the last 60 seconds of data. Once in the *File System Cache* the changes are flushed to disk.

Wired Tiger has concurrency protocols that are the equivalent of *document level locking*.

In MMAP v1 the raw BSON is held in the same format in memory as it is stored on disk so there is no compression available. With Wired Tiger the data format is different in the *Wired Tiger Cache* than on the file system so compression is available. There are two type of compression you can enable in the Wired Tiger stored engine: *Snappy*or *zLib* which are both open source compression libraries. By default compression is *Snappy*which is less process intensive than *zLib* but won’t compress the data as much.