# FileStore Migration

## Introduction

The FileStore that is included with OA4MP is intended for smallish installations and does not really scale. A scenario is that one installs it for what is intended to be a low-volume service, but which instead becomes very large. For instance, if auto-approve is on, some clients register themselves essentially for every transaction (*i.e.* single use), resulting in hundreds of thousands of "clients" = one for every time a user logs in In that case, the system is overwhelmed and performance slows dramatically.

The Right Way to deal with this is to use a Derby store (which requires no database administration and just works), but how do you get all of your stores migrated? There is a tool for this that makes it quite easy and has been tested with about a million entries. The entire process takes a couple of hours.

## The FSMigration tool

This is a separate jar that is downloadable from GitHub.  It runs at the command line and takes various arguments and does various things. In a nutshell the sequence is

1. Take the original source FileStore
2. Create a migration database (using Derby) that ingests every entry in the FileStore. You do not need to install anything for Derby! This is done automatically.
3. ***Ingest*** entries into a helper database located in the original filestore. This is relatively fast and just reads directly from disk as a first pass.
4. ***Migrate*** the in batches to the new SQL store (which can be any supported SQL store).
5. Allow for reports on what was imported. Since it is assumed you do not want to install Derby itself, this takes the place of doing direct queries.
6. Allow for deleting the migration database.

The reasons for the two stage migration are integrity and performance. Integrity ensures that valid store entries are faithfully transferred. This will also transfer pending transactions so the net effect should be that the system goes down for maintenance during the migration and comes up without any losses. The performance issue means not using the internal FileStore APIs but manually accessing the store contents and optimizing the operations. This means the operations run at easily over 100 times the speed that would happen otherwise in a large FileStore (which is the chief limitation of one).

## Initial Prerequisites

1. The file store exists. There is not a directory named **`ingest`**. This will be created and managed.

2. The target store  may or may not exist. If it does not exist, then it will be created.
3. Space available on the server in question. The entire file store will be processed at some point, though storing an object in a database takes up much less space than a file. If you have as much space available as the size of the file store, you should be fine.

## Running the tool

The tool is named fs-migrate.jar and is an executable jar. Starting it with no parameters shows the help:

```
java -jar fs-migrate.jar
(help is printed)
```

If you want to create the target database directly, then run

```
java -jar fs–migrate.jar -src … -setup
```

Make sure the -src is set and use

The ingestion phase – reading the file store and creating a database of entries so that the migration can be managed – is assumed to be atomic. That is to say, if the ingest database is found to exist, it is assumed to be complete. If the ingestion is interrupted, delete the database. Once the ingestio phase is over, the system will try to migrate the entries and uses the ingestion database to track progress, hence the tool can be stopped and restarted and it should just resume.

Typical output from ingestion is (line breaks added for readability)

```
java -jar fs-migrate.jar \
  -src $SRC_CFG\
  -srcName $SRC_NAME\
  -targetName $TARGET_NAME\
 -v \
 -showConnect
No explicit target configuration, using the source configuration for both.
FSMigrationTool database file=/home/ncsa/temp/oa4mp2/fileStore/ingest
   creating migration database
   done!
connect
'jdbc:derby:/home/ncsa/temp/oa4mp2/fileStore/ingest;dataEncryption=true;user=oa4mp;
bootPassword=Qwertyuiop321;password=Asdfghjkl123';
   :starting to ingest
   :processing 52639 files from
/home/ncsa/temp/oa4mp2/fileStore/adminClients/dataPath.
.........../..............\..........  52000  files ingested @ 71428 Hz
   :updating migration database...
   :       store : adminClients
   :       total : 52639
   :    rejected : 0
   :update speed : 468 Hz
   :       bytes : 122.9746 MB
   :  total time : 1.9150 min.
   :processing 653463 files from /home/ncsa/temp/oa4mp2/fileStore/clients/dataPath.
..............\......../.............  653000  files ingested @ 76923 Hz
   :updating migration database…
```

```
   :       store : clients
   :       total : 653463
   :    rejected : 1
   :update speed : 441 Hz
   :       bytes : 1.5249 GB
   :  total time : 25.0500 min.
   :    file not found:1
   :processing 114027 files
(lots more, then a final ingestion summary)

  :elapsed ingestion time : 35.9222 min.
  :      total file count : 834840
  :       total bad files : 2
  :      total byte count : 1708480061
```

Things to note

- -showConnect means the connection string to the ingestion database is shown. If you have Derby installed and want to use their command line **ij** tool to snoop, you can
- There is a status bar that allows you to watch the ingestion. This can be turned off.
- Each store is ingested and stats for that are displayed.
- If you have upkeep logic in the target store, that will be applied in the next phase. Ingestion is strictly to grab every file name in the store and set up a system to track migrating it.
- The ingestion database is made. This is atomic in the sense that if it exists it is assumed to be intact. If for some reason the process fails, you will have to remove it and its directory, here **`/home/ncsa/temp/oa4mp2/fileStore/ingest`**

A practical issue with migrating a simply huge file store is tracking progress. Since the files are saved as serialized XML and the name of the file is a hash of the ID – not generally reversible – the only way to really check if a file has been migrated would be to read every file and parse it. The ingestion database pulls in the list of files and as they are migrated will track progress. This allows for restarting the process.

## Migration

This is when the entries in the ingestion database are processed. If there are upkeep rules on the target, yo may have them enforced at this time. While not recommended, you can restart the process if it should stop. The ingestion database tries to ensure atomicity, but it is always best to not tempt Fate. A typical migration looks like

```
java -jar fs-migrate.jar \
  -src $SRC_CFG\
  -srcName $SRC_NAME\
  -targetName $TARGET_NAME\
 -v
No explicit target configuration, using the source configuration for both.
target database connection string:
FSMigrationTool database file=/home/ncsa/temp/oa4mp2/fileStore/ingest
```

```
ingest database exists
  :starting to migrate...
  :starting to migrate 52639 items for adminClients
......\......................./......  52639  files migrated in adminClients,
rejected=52639 @ 0 Hz upkept=0
```

In this case, the upkeep rules are being applied and out of 40100 files, 39281 are excluded.