Documentum DQL FAQ & Troubleshooting

FAQs

Specific

What kind of "Documentum ID" is used for indexing?

We use Documentum chronicle_id as the id for indexing because this number stays the same for all versions of one document.

Which version of the document content is used for indexing?

Although document can have many versions in Documentum we use only "current" version of the document for indexing

We are using {SLICES} param in the DQL query. Even though the scanner threads set at '20', at max, the scanner threads count will always be '16'. Is that correct?

In the current DQL connector with {SLICES} Aspire would use up to 16 scanner threads. Without "slices" the whole scan phase would be handled by one scanner thread only.

Can you explain the usage of 'Scanner threads' Vs 'Processing threads'?

Scanner threads in all connectors are basically used for getting list of items for further processing. For classical hierarchical connectors - i.e. File system - scanner threads provides list of files for each traversed directory. DQL connector is somehow "flat" and all items are provided by the specified DQL statement. "Slices" means that we artificially create more DQL statement to achieve some concurrency. But usually the scanning tasks only run the DQL statement and store chronicle_Id, r_object_Id to the NoSQL queue. The time here should not have been critical unless some real slow DQL statements were to be processed.

Processing threads in DQL connector work for all documents like that: 1. getting the object detail by r_object_id from Documentum 2. populating the Aspire item from scan phase by the attributes from object detail - getting metadata and ACL's 3. fetching the content from Documentum by r_object_id and store the content as a stream into the Aspire job (fetcher) 4. extracting text using TIKA for "text" files 5. continuing processing the job over workflow components.

How the FetchUrl is implemented?

Fetch URL is implemented in DQL as the component reading the whole content of the Documentum file into the memory as a byte array and exposing this array as the ByteArrayInputStream object to later stages.

The most atomic operation here is the actual reading the content from Documentum by DFC classes – something like iDfSysObject.getContent() .

Is any connection pool used for the DQL connector?

Aspire connector framework uses his own connection pool. But to understand what is going on here requires always knowledge about how this is implemented in different connectors.

In DQL we store in each Aspire connection object the DFC object IDfSessionManager. We use this object then for all communication with Documentum like this: 1. sessionManager.getSession 2. issue some DQL or other command using this session 3. sessionManager.release(session)

Can we see some real example of DQL to be used for crawling?

```
select for READ r_object_id, i_chronicle_id
from imms_document
where (i_is_deleted=TRUE Or (i_is_deleted=FALSE AND a_full_text=TRUE AND r_content_size > 0 AND
r_content_size < 10485760 ))
AND (Folder('/Clinical', DESCEND) AND r_modify_date >= DATE('7/1/2017'))
AND {SLICES}
```

General

Warning! The question: Why does an incremental crawl last as long as a full crawl is not relevant for this connector!

Why does an incremental crawl last as long as a full crawl?

Some connectors perform incremental crawls based on snapshot entries, which are meant to match the exact documents that have been indexed by the connector to the search engine. On an incremental crawl, the connector fully crawls the repository the same way as a full crawl, but it only indexes the modified, new or deleted documents during that crawl.

For a discussion on crawling, see Full & Incremental Crawls.

Save your content source before creating or editing another one

Failing to save a content source before creating or editing another content source can result in an error.

```
ERROR [aspire]: Exception received attempting to get execute component command com.accenture.aspire.
services.AspireException: Unable to find content source
```

Save the initial content source before creating or working on another.

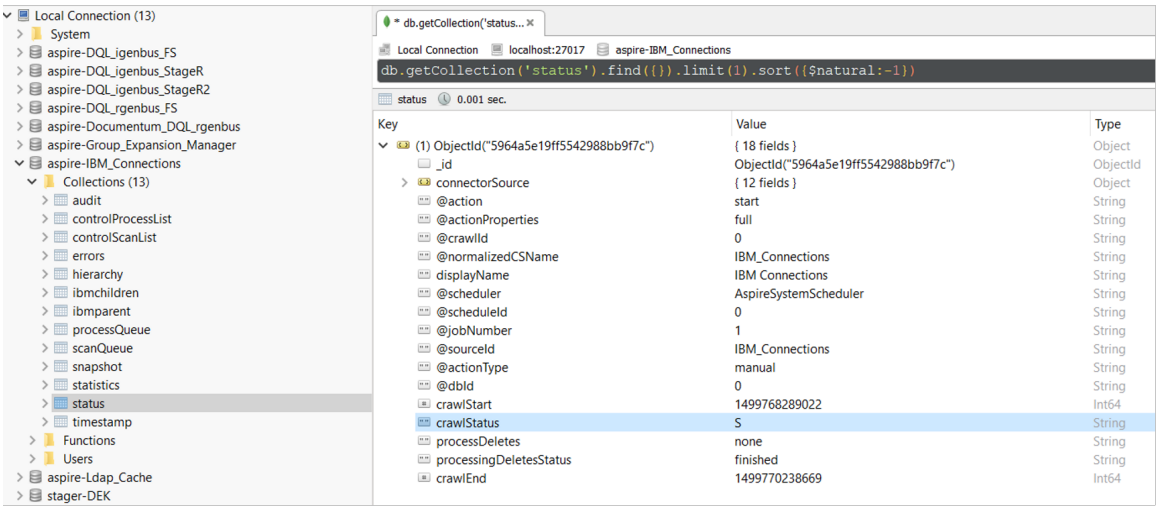
My connector keeps the same status "Running" and is not doing anything

After a crawl has finished, the connector status may not be updated correctly.

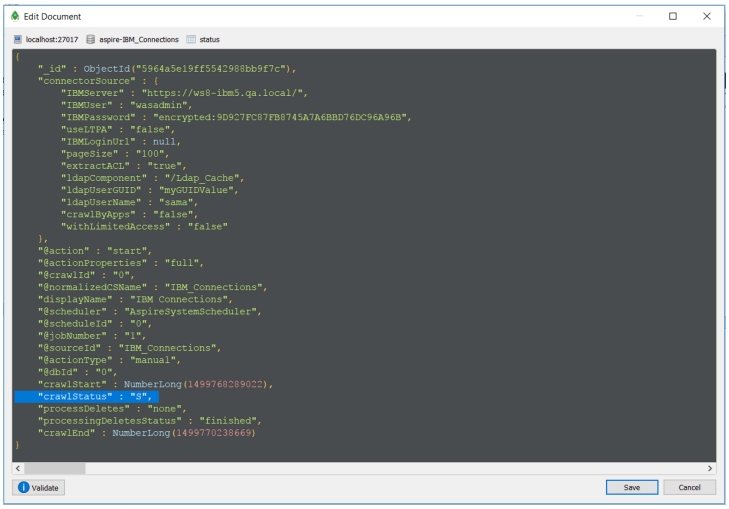
To confirm this, do the following:

1. In Robo 3T (formerly Robomongo), go to your connector database (like: *aspire-nameOfYourConnector*).
2. Open the "Status" collection and perform the following query:

```
db.getCollection('status').find({}).limit(1).sort({$natural:-1})
```



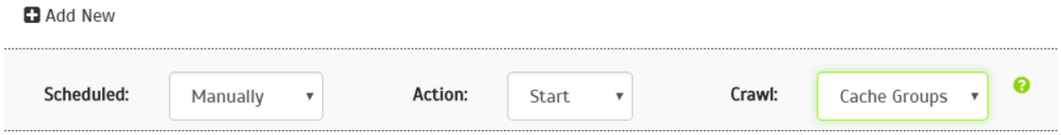
3, Edit the entry and set the status to "S" (Completed).



Note: *To see the full options of "Status" values, see* MongoDB Collection Status*.*

My connector is not providing group expansion results

Make sure your connector has a manual scheduler configured for Group Expansion.

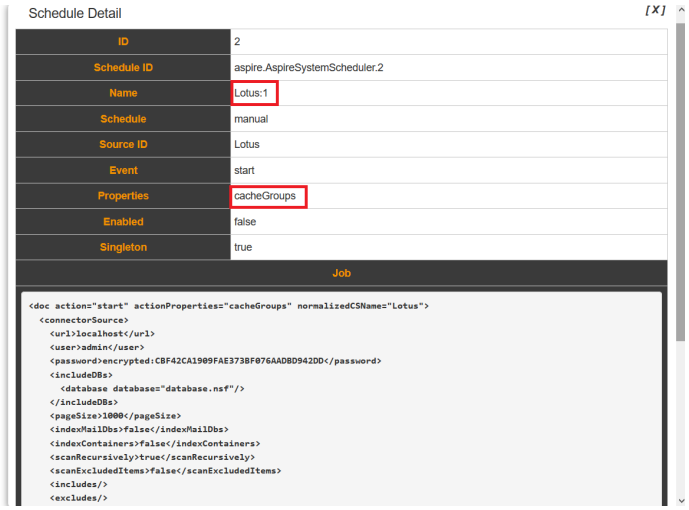


1, Go to the Aspire debug console, and look for the respective scheduler (in the fourth table: Aspire Application Scheduler).

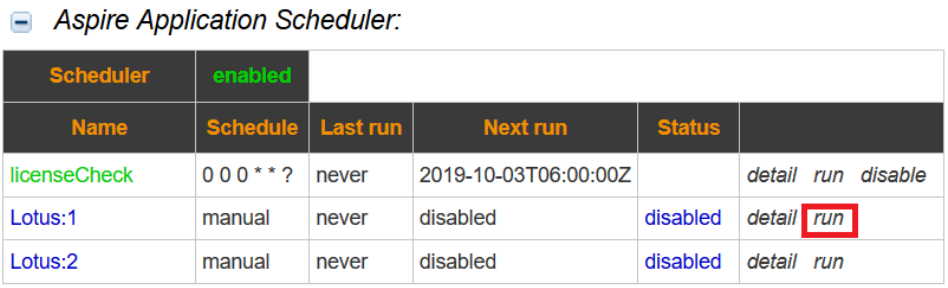
Aspire Application Scheduler:

Scheduler	enabled				
Name	**Schedule**	**Last run**	**Next run**	**Status**	
licenseCheck	0 0 0 * * ?	never	2019-10-03T06:00:00Z		*detail run disable*
Lotus:1	manual	never	disabled	disabled	*detail run*
Lotus:2	manual	never	disabled	disabled	*detail run*

2. If you are unsure which scheduler is for Group Expansion, you can check the Schedule Detail.
    - You can identify it with the value: cacheGroups



3.To run the Group Expansion process, click ***Run***.



Troubleshooting

Problem

FetchUrl is consuming too much time

Solution

Fetch URL is implemented in DQL as reading the whole content of the Documentum file into the memory as a byte array and exposing this array as the ByteArrayInputStream object to later stages.

Scanning threads are not relevant here

Increasing the number of processing threads can help but it must be balanced with heap size assigned to JVM. It also of course depends on the size of the fetching files. More processing threads means also more memory consumed since more possibly large files are processed in parallel way. This whole process could be tuned with the help of for example visualVm graphs which could show also the garbage collector activity etc.

The most atomic operation here is the actual reading the content from Documentum by DFC classes – something like iDfSysObject.getContent() . If this operation is slow then no Aspire related configuration can help.

Problem

If the 'Processing threads' is increased to '200', then noticing 'Max server sessions exceeded' Documentum exceptions for some documents

Solution

About "Max server sessions exceeded" - https://community.emc.com/thread/65223?start=0&tstart=0

We do not cache Documentum session objects and if you get the above mentioned exception this should mean than a lot of threads are talking to Documentum at the same time

Problem

DQL connector performs poorly

Solution

We've attached some numbers from real customer XX – they achieved something like 30 – 50 DPS for 8 mio docs . This varied from several reasons (like connector restarting + stopping etc.)

• They used 32GB RAM for JVM
• They set up "ExtractText strategy":

o Very large files files (> 100MB) were processed in separate crawl – DQL statement with proper filtering can be used for that
o nonText document filtering regex file was carefully used to really skip all "nonText" useless extractions!
o Max extract text size parameter was used for the crawl

• They started testing DQL connector without additional workflow component to isolate it as much as possible
• They knew Documentum very well and was able to tune dfc.properties like - dfc.cache.object.size,..

At some other customer they made some real progress by tuning various parameters:
Scanner threads: 20
Scan queue size: 50
Processing threads: 200
Processing queue size: 500

JAVA_INITIAL_MEMORY=8g
JAVA_MAX_MEMORY=16g

Problem

Downloading jars for Documentum BOF classes does not work

Solution

The special felix jar is needed as described here:

https://jira.contentanalytics.digital.accenture.com/browse/ASPIRE-4620

Problem

DFC_API_W_ATTEMPT_TO_USE_DEPRECATED_CRYPTO_API warning/exception at the start of the crawl

Solution

This was answered by one of the Aspire user:

This is related to the fact that we still use passwords for the BOF registry user which were encrypted with the old non-FIPS compliant algorith MD5. If we would encrypt the credentials with the new algorithm (SHA1 I believe) the warning would go away. However it is not an issue as even latest DFC still support the old algorithm. We decided to stick to the old passwords as we don't see a security risk (PW is stored on protected servers and the account has only very few read permissions within the system).

Problem

Java 11 - annoying log message java.security.policy: error adding Entry: java.net. MalformedURLException: Unknown protocol: jrt

Solution

The issue is with the syntax of default policy file <jdk11>\lib\security\default.policy which DFC uses. The message does not effect the connector functionality. But if you want to get rid of this message you can chnage the content of default.policy for example like this:

```
grant {
permission java.security.AllPermission;
};
```