**Description**

The Lucene text search module provides functionality to find documents and data fast. It is an easy implementation and it is optimized for Mendix. An alternative when XPath and OQL are not sufficient.

**Typical usage scenario**

Type usage scenarios are

1. Search for text in documents.
2. Find data fast when using more than 100.000 records.
3. Finding data in multiple objects at once, like orders+orderlines+items.

**Features and limitations**

1. Searching in filedocuments, excel (xls and xlsx), word (doc and docx), xml, text and pdf.
2. Automatic recognizing of file formats
3. Define your own indexed texts.
4. Has a delay of 5-10 seconds before the index is updated. It works in the background, so Mendix microflows will proceed.
5. Supports Multiple indexes
6. Backup and restore for Mendix cloud is included
7. Transfer indexes between environments.
8. Searching with wildcards and logical operators like AND – OR
9. The index is connected to a database. It contains the internal id’s Mendix uses to identify data. You cannot combine a database with an index from another environment.
10. You can copy both database and index from for example acceptance to production.
11. Be sure to update the indexes well

**Dependencies**

1. The modules uses the same POI jars as in the excel-importer module

**Installation**

1. Import the module from the appstore
2. Define a unique constant for every index
3. Remove the sample implementation

**Configuration**

Look at the entities ExampleCustomer and ExampleFileDocument for example. You can remove them if your implementation works, if your implemented this module before.

1. Choose the entities that must be indexed.
2. Change the Enumeration Enum\_IndexType and set them that entities.
3. Create an N:1 association from Lucene.SearchResult to al entities that are indexed.
4. If you deploy in the Mendix cloud connect the After startup (ASu\_RestoreBackup) and before shutdown (BSd\_CreateBackup) microflows.
5. Create Before commit and before delete Microflows, use the ones in the module as an example. Two variants: one for entities and one for filedocuments.
6. In the Before commit construct your searchable text, separated with spaces, don’t forget to exclude ‘null’ texts.
7. If you open the search result change the MF\_OpenResult microflow to match it with you constants.
8. Implement search
   * Create a search command
   * Add search text like “Romans AND War”
   * Call the Java action to find data with that combination
   * For every result 1 SearchResult is created with one of the assocations filled in.
9. Expand before shutdown and after startup microflows and process all indexes.

**Concepts**

The modules use the Apache Lucene fulltext module. This module creates a set of files in a directory. We have selected subdirectories of the temp folder. The index is used to find data fast.

**Technical details**

Most of the time of Lucene indexing is consumed by updating the index. Updating 1 object will cost almost the same time a 500 objects. The real work is done in the background and will wait for 5 seconds or 500 objects to update the index. This is fast enough to run an Excel import with Before commit events without significant delay in the performance of the foreground proces.

After a time-out of 10 minutes to index is closed, so Nightly backups can be made.

The backup files are stored in the temp folder on the cloud. This folder is cleared when a new version is deployed. This is solved by a backup in a filedocument in a before shutdown event and restore after startup. Be sure to connect this events.

The search command has special characters, you have to escaped them if you provide end-user search functionality. A java-action is provided for that.

Lognode: Lucene

Lucene version is 4.3