Documentation for DMS to OpenKM migration toc

Table of Contents

Table of Contents 2
DMS to OpenKM migration tool 1.0 3
License 3
User guide 4
Export data 4
Import data 4
Supported DMS and versions 5
Migration tool 1.0 5
Knowledge Tree 6
Export data 6
Development 8
Pom dependency 8
Services implementation 8
Create your own implementation for a new DMS 10
DocumentSrv 10
FolderSrv 11
RoleSrv 11
UserSrv 11
Example 12

DMS to OpenKM migration tool 1.0

This project provides a core code and is used to migrate from a Document Management System (DMS) to OpenKM.

Please check Supported DMS and versions section for more information.

The program will read the data from your system and generate an intermediate format that OpenKM knows how to read.

License

OpenKM Migration tool is licensed under the terms of the EULA - OpenKM End User License Agreement and published by OpenKM Knowledge Management System S.L.

This program is distributed WITHOUT ANY WARRANTY not even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the **EULA - OpenKM End User License Agreement** for more details.

User guide

There are two different phases to migrate a repository to OpenKM:

1. Export data from a DMS to OpenKM intermediate format.
2. Import data to OpenKM.

Export data

It depends on the implementation how to extract data from a DMS system. The core of dms2okm provides useful methods to help extract data and generate intermediate format for OpenKM.

Please see Supported DMS and versions section for more info about available DMS implementations.

Import data

This step is the same for whichever DMS you want to export/import.

The export phase generate an intermediate format that be used by OpenKM to import your content.

To import the data you must follow these steps:

1. Go to **Administration tab**.
2. Click **Repository import** button.
3. Select the **File System path** and check **Metadata** and **History** (please do not select Restore UUIDs checkbox).
4. Click **import** button.

After a while (depends on the amount of data) you will see your data imported correctly.

OpenKM will prompt any problem with the importation of files and folders.

Supported DMS and versions

This section provides a table with the compatibility between a DMS and OpenKM what has been checked by OpenKM or reported by thirdparty users.

The migration tool might run with older or newer versions of these DMS. You can report us success, problems, etc. at our public forum at http://forum.openkm.com.

Migration tool 1.0

DMS	Version	OpenKM Community	Professional version
Knowledge Tree	3.7.0.2	**6.2.x and upper**	**OpenKM Professional 6.2.x and upper**

Knowledge Tree

This section contains information about the implementation of the exportation tool for Knowledge Tree.

You can download from here: dms2okm-knowledge-tree-1.0.zip

Please read the license terms before using this application.

OpenKM Migration tool is licensed under the terms of the EULA - OpenKM End User License Agreement and published by OpenKM Knowledge Management System S.L.

This program is distributed WITHOUT ANY WARRANTY not even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the **EULA - OpenKM End User License Agreement** for more details.

Export data

The program used to export data is a jar file which can be executed using the terminal.

The command to launch is the following:

```
java -jar dms2okm-knowledge-tree.jar db_url db_user db_password sys
```

- **db_url**: JDBC connection URL.
- **db_user**: JDBC connection user.
- **db_password**: JDBC connection password.
- **system_path**: where the exportation content will be stored.
- **repo_path**: path to the repository. Used to recover content documents.
- **root_path**: path to the parent folder.

You must provide a valid MySQL connection URL because right now only MySQL protocol is supported.

Only Knowledge Tree repository is supported yet. To see a complete list of supported DMS please check Supported DMS and versions section.

Example of execution:

java -jar dms2okm-knowledge-tree.jar jdbc:mysql://localhost:3306/database user password /home/user/export /home/repoPath ""

Note that this step could take longer to execute (up to hours) because it makes a full copy of the whole repository (documents included). Also it needs enough space in your disk to store the copy of the repository.

Development

This section provides a guide to extend the migration tool to add any DMS to export data from.

Pom dependency

To add dms2okm core to your project you have to include the OpenKM maven repository and dms2okm dependency in your pom.xml:

Repository

```
<repositories>
 <repository>
  <id>openkm.com</id>
  <name>OpenKM Maven Repository</name>
  <url>http://maven.openkm.com/maven2</url>
 </repository>
</repositories>
```

Core dependency

```
<dependency>
 <groupId>com.openkm</groupId>
 <artifactId>dms2okm</artifactId>
 <version>1.0</version>
</dependency>
```

Services implementation

The project uses Google guice (https://github.com/google/guice) to create and inject the services needed.

First you need an injector:

```
package com.openkm;

import com.google.inject.AbstractModule;
import com.openkm.srv.DocumentSrv;
import com.openkm.srv.FolderSrv;
import com.openkm.srv.RoleSrv;
import com.openkm.srv.UserSrv;
import com.openkm.srv.impl.example.DocumentSrvExample;
import com.openkm.srv.impl.example.FolderSrvExample;
import com.openkm.srv.impl.example.RoleSrvExample;
import com.openkm.srv.impl.example.UserSrvExample;
```

```
public class ExampleInjector extends AbstractModule {

    @Override
    protected void configure() {
        bind(FolderSrv.class).to(FolderSrvExample.class);
        bind(DocumentSrv.class).to(DocumentSrvExample.class);
        bind(UserSrv.class).to(UserSrvExample.class);
        bind(RoleSrv.class).to(RoleSrvExample.class);
    };
}
```

The class has a method called **configure()** what binding the interfaces with service implementations.

In your main class you need the following in order to **use the injector**:

```
Injector injector = Guice.createInjector(new ExampleInjector());

RepositoryExporter exporter = injector.getInstance(RepositoryExport
```

The **previous code use the injector to set the exporter services** implementation.

Finally **ensure any service implementation** should **have @Singleton annotation**. For example:

```
package com.openkm.srv.impl.example;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.SQLException;
import java.util.List;

import javax.inject.Singleton;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.openkm.dao.ExampleMockDao;
import com.openkm.srv.RoleSrv;

@Singleton
public class RoleSrvExample implements RoleSrv {

    private static Logger log = LoggerFactory.getLogger(FolderSrvEx

    private ExampleMockDao mockDao = new ExampleMockDao();

 @Override
 public void generateRoleData(String systemPath) throws SQLExceptic
     log.info("generateRoleData({})", systemPath);
     List<String> roles = mockDao.getRoles();
     File usersDataFile = new File(systemPath + File.separator + "r
        usersDataFile.createNewFile();
        PrintWriter writer = new PrintWriter(usersDataFile);
        for (String role : roles) {
            String roleQuery = "insert into okm_role(rol_id, rol_ac
```

```
                writer.write(roleQuery + "\n");
            }
            writer.flush();
            writer.close();
    }

}
```

Create your own implementation for a new DMS

The code from **dms2okm** core is generic. So there is no need to make any change on it's of methods.

You should only add modifications in your DMS export implementation.

In order to add new DMS to the tool you should provide implementations for the following interfaces:

- **DocumentSrv**: contains methods to get children documents from a folder and document security.
- **FolderSrv**: contains methods to get children folders from a folder and folder security.
- **RoleSrv**: methods to extract roles information.
- **UserSrv**: contains methods to extract users data.

DocumentSrv

This interface contains methods related to documents management:

Method	Return	Description
getChildrenDocuments(String nodePath)	**List<Document>**	Return existing documents for a given path
getGrantedUsers(String nodePath)	**Map<String, Integer>**	Get user permissions of a document
getGrantedRoles(String nodePath)	**Map<String, Integer>**	Get roles permissions of a document
countTotalDocuments()	**Integer**	Count number of documents to export

getFileInputStream(String nodePath)	**InputStream**	Get and InputStream to the content of this file

FolderSrv

This interface contains methods related to folders management:

Method	Return	Description
getChildrenFolders(String nodePath)	**List<Folder>**	Return a list of existing folders for a given path
findFolderByPath(String nodePath)	**Folder**	Return a Folder object from a path
getGrantedUsers(String nodePath)	**Map<String, Integer>**	Get user permissions of a folder
getGrantedRoles(String nodePath)	**Map<String, Integer>**	Get role permissions of a folder
countTotalFolders()	**Integer**	Count number of folders to export

RoleSrv

This interface allows the tool to export roles information from your DMS:

Method	Return	Description
generateRoleData(String nodePath)	**void**	This method is used to extract the information about the roles from the DMS. Using this information the tool will generate an sql file to migrate them.

UserSrv

This interface allows the tool to export users information from your DMS:

Method	Return	Description
generateUserData(String		This method is used to extract the information about

nodePath)	**void**	the users from the DMS. Using this information the tool will generate an sql file to migrate them.

Example

This code contains an example of implementation. It creates files and folders without any DMS installed. Just create mock files and folders ready to import by OpenKM.

You can download source code for an example of implementation from here: dms2okm-example.zip