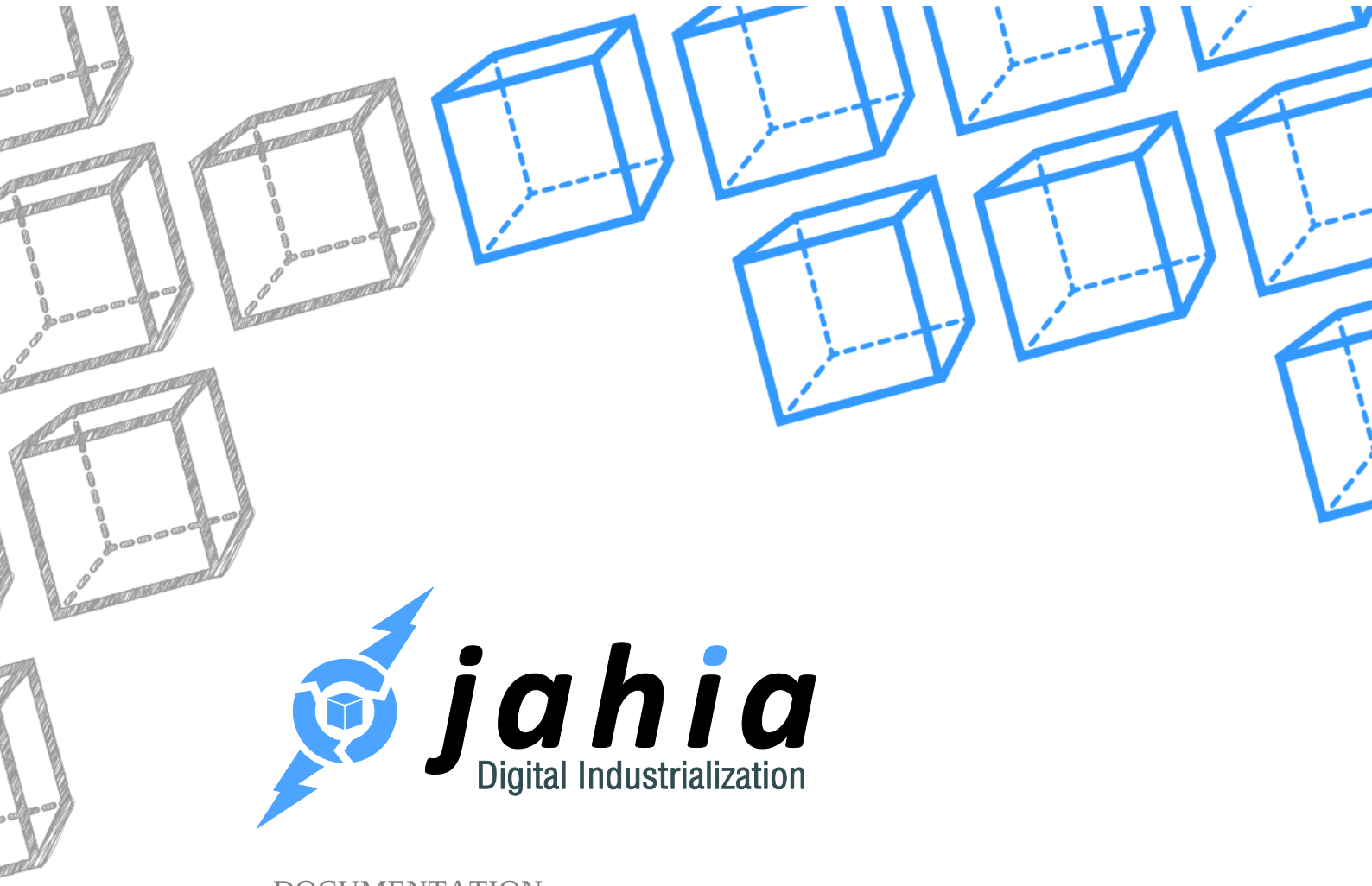


jahia
Digital Industrialization

DOCUMENTATION

Portal Factory 1.0 - CMIS Connector Module documentation

Rooted in Open Source CMS, Jahia's Digital Industrialization paradigm is about streamlining Enterprise digital projects across channels to truly control time-to-market and TCO, project after project.

Jahia Solutions Group SA

9 route des Jeunes,
CH-1227 Les acacias
Geneva, Switzerland

http://www.jahia.com

Summary

1.Overview

1.1. About CMIS

Content Management Interoperability Services (CMIS) is an open standard that allows different content management systems to inter-operate over the Internet. Specifically, CMIS defines an abstraction layer for controlling diverse document management systems and repositories using web protocols.

CMIS defines a domain model plus bindings that can be used by applications. OASIS, a web standards consortium, approved CMIS as an OASIS Specification on May 1, 2010. CMIS 1.1 has been approved as an OASIS specification on December 12, 2012.

CMIS provides a common data model covering typed files and folders with generic properties that can be set or read. There is a set of services for adding and retrieving documents ('objects'). There may be an access control system, a checkout and version control facility, and the ability to define generic relations. Three protocol bindings are defined, one using WSDL and SOAP, another using AtomPub,[4] and a last browser-friendly one using JSON. The model is based on common architectures of document management systems.

1.2. About this module

This module makes it possible to mount CMIS providers as external JCR content providers, making it possible to access folders and files stored in a CMIS repository as if they were part of the local JCR data structure. It then makes it possible to use all of Digital Factory's UIs to manipulate content from a CMIS repository, or even add mixin on top of CMIS external nodes (such as ratings or comments !).

The module makes it possible to:

- Integrate an external repository that complies to the CMIS standard.
- Provide full CRUD support for CMIS repository inside the Digital Factory JCR repository

- Use JCR Search to search inside connected CMIS repository

1.3. Module features

1. The two base CMIS types (document and folder) and any custom sub-types are supported.
2. Full Create, Read, Update and Delete (CRUD) support along with renaming and moving for all supported types. It is also possible to modify content and properties values.
3. Both path and id identification are supported by external provider.
4. Ordering is not supported, as CMIS does not support such a feature.
5. Searching inside a CMIS repository is supported with the following restrictions (see TestQuery.txt for demo queries).
    - Join is not implemented. We return only one node path per row.
    - lower, upper, length, score - are not supported.
    - non mapped properties acts as null.
    - any specific behavior for multivalued properties is not implemented.
    - date type is supported only if cast expression is used (no auto converting).
6. Configurable mapping for custom types. CMIS does not supports language-specific attribute,s so if Jahia queries use language conditions they will not be mapped to CMIS queries.

1.4. Implementation notes

The underlying CMIS implementation CMIS used is Apache Chemistry. In the default configuration the external CMIS repository is mounted at the /external-cmis-mapped path. To map CMIS types to JCR types we use node types cmis:folder and cmis:document. Common properties are declared in a separate mixin called cmis:base.

If it is needed to map custom CMIS type this can be achieved by extending the cmis:folder or cmis:document nodetypes. Mappings for custom types must then be configured in the spring deployment descriptor of the module. The CmisTypeMapping bean supports type inheritance, so no property mapping duplication is needed.

2. Configuration

2.1. Installation

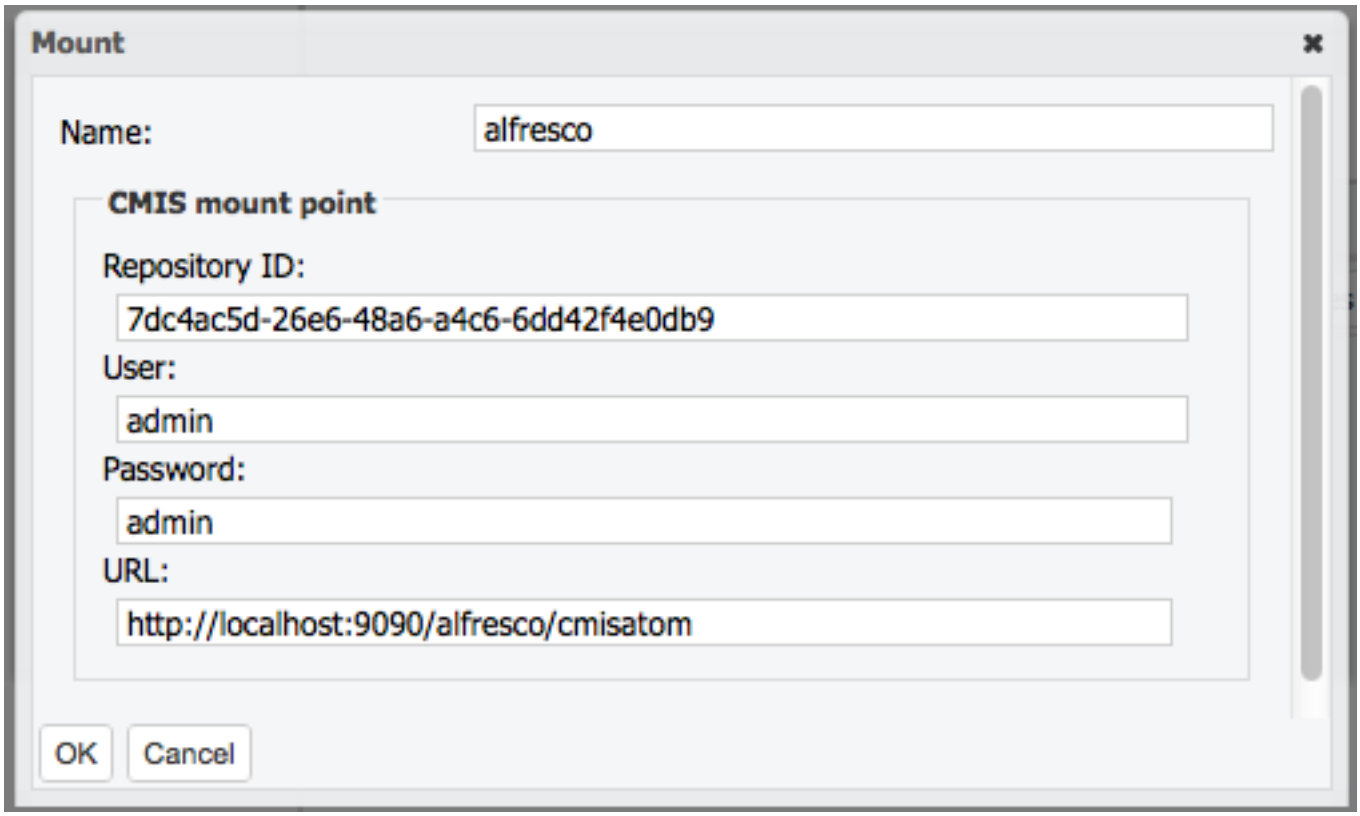
Installing the CMIS provider module is straight-forward. Simply copy the JAR into the WEB-INF/var/modules directory or use the administration « Manage Modules » interface to either install it by uploading the file or installing from a public or private Jahia App Store.

2.2. Connecting to an Alfresco repository

Alfresco provides a CMIS Atom interface making it easy to connect to the file repository remotely using this standard. The following steps detail how to connect Digital Factory to Alfresco using the CMIS external provider module :

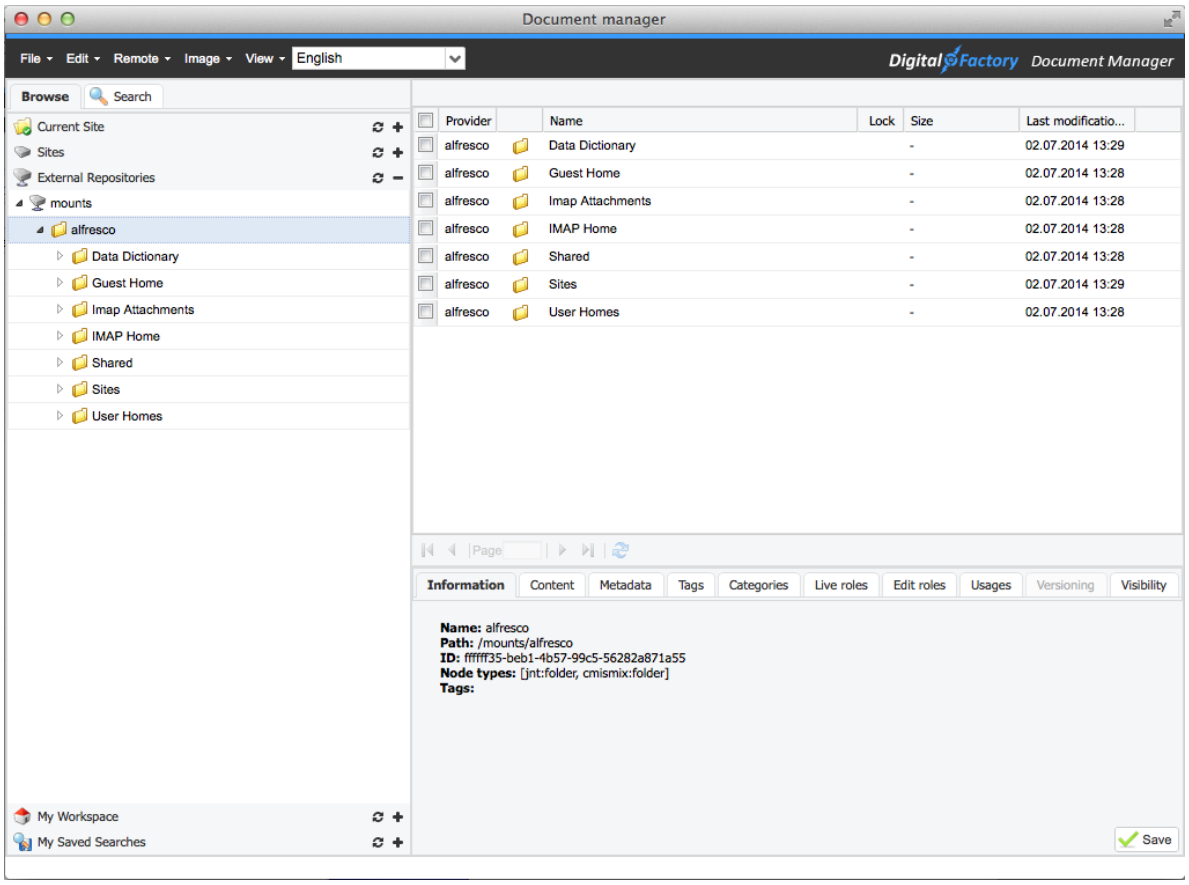
1. Retrieve cmisatom XML descriptor using : http://ALFRESCO_HOSTNAME:ALFRESCO_PORT/alfresco/cmisatom
2. Open the download XML file and look for the repository ID that should look like this :
3. <cmis:repositoryId>7dc4ac5d-26e6-48a6-a4c6-6dd42f4e0db9</cmis:repositoryId>
4. Open the Document Manager, and select « Mount » from the « Remote » menu.
5. Select the CMIS Provider type
6. Fill in the form as in the following example. Note that the repository ID is the one that was retrieved in the cmisatom XML file we downloaded previously. Also don’t forget to specify a

meaningful mount point name (in the example below we used “alfresco”).



7. Click OK
8. You should then be able to open the “External repositories” tab and you should have a “Mounts” node with an “alfresco” sub node. If you click on this node, you should see the

root Alfresco folders as illustrated below:



You can use the search to search inside the CMIS repository, but please note that the language search will not work since the external repositories have no notions of languages, so make sure you leave it empty when searching in CMIS repositories.

2.3. Connecting to an OpenCMIS In-Memory repository

The easiest way to create test environment is to use *OpenCMIS InMemory Repository* https://chemistry.apache.org/java/developing/repositories/dev-repositories-inmemory.html InMemory Repository can be deployed either in separate Tomcat or in the same as for Jahia. Separate Tomcat is preferable because webapp's startup order can not be configured. In other case you must be sure you don't access mount point on startup.

OpenCMIS Workbench may be used as CMIS client.

1. Download and install a new Tomcat instance in which we will deploy the OpenCMIS server. You can download Tomcat from : http://tomcat.apache.org
2. Download the OpenCMIS server from the following location: http://chemistry.apache.org/java/download.html . You should use the “OpenCMIS Server Webapps” commodity package
3. Unpack chemistry-opencmis-server-inmemory-xxx.war into tomcat/webapp/inmemory folder. The repository config file is located in inmemory\WEB-INF\classes\repository.properties. Default configuration is should be ok.
4. You might want to adjust the Tomcat configuration to listen on different ports if running it on the same machine as your Digital Factory installation. To do so modify the ports in TOMCAT_INSTALL_DIR/conf/server.xml
5. Start your Tomcat CMIS instance using the startup scripts in TOMCAT_INSTALL_DIR/bin/startup.sh/.bat
6. Go to Document Manager, select Remote - Mount - CMIS provider
7. Fill in the form with these entries. Configure port, depending on your tomcat config. Make sure you specify a meaningful mount name such as “opencmis”

```
Mount point name=opencmis
repository id=A1
user=dummyuser
password=dummysecret
url=http://localhost:18080/inmemory/atom11
```

Once you confirm the entry there will be CMIS repository mounted in the “External Repositories” tab under /mounts/opencmis.

3.Advanced configuration

The Digital Factory CMIS external provider is fully configurable using a Spring configuration file provided in the CMIS module at the following location : src/main/resources/META-INF/spring/cmis-provider.xml using a special CmisConfiguration bean. The CmisConfiguration bean has two main properties:

- repositoryProperties - map of connection related configuration properties. By default, repositoryProperties are configured for usage of general jahia.properties
- typeMapping - list of CmisTypeMapping beans. There are 2 base mappings: for cmis:document and cmis:folder.

3.1. Mapping configuration

CMIS - JCR mapping is configurable using the Spring configuration file. For each JCR type there must be CmisTypeMapping bean. Each CmisTypeMapping may have list of property mappings. Each property may support 3 modes ("mode") - rwc. R - read , W - write (update), C - property that will be set on creation. By default each property has only read mode active.

Property mapping example

```
<bean class="org.jahia.modules.external.cmis.CmisTypeMapping"
id="cmis_document" p:jcrName="cmis:document" p:cmisName="cmis:document">
    <property name="properties">
        <list >
            <bean p:cmisName="cmis:createdBy" p:jcrName="jcr:createdBy"
class="org.jahia.modules.external.cmis.CmisPropertyMapping" />
            <bean p:cmisName="cmis:description"
p:jcrName="cmis:description"
class="org.jahia.modules.external.cmis.CmisPropertyMapping" p:mode="rwc"
/>
            <bean p:cmisName="cmis:contentStreamFileName"
p:jcrName="cmis:contentStreamFileName"
class="org.jahia.modules.external.cmis.CmisPropertyMapping"
p:mode="rw"/>
        </list>
```

```
    </property>
</bean>
```

**This fragment is not real mapping example. We use it only for documentation purposes. **

In this fragment we map CMIS *cmis:createdBy* property on JCR *jcr:createdBy* property as read only. In the next line *cmis:description* is mapped to a JCR property with the same name in read, create and write modes. In the next line the *cmis:contentStreamFileName* property is mapped to a JCR property with the same name in read and write modes. For this property create mode is not set. it means you can't set a value for this property on node creation, but it may be modified later. If the create mode is set without write mode, it is possible to set property value on creation, but it may not be modified later.

CmisTypeMapping beans may be organized in trees to benefit from inheritance. Child beans will inherit all parent attribute mappings. Local attribute mappings will override inherited. Inheritance may be configured in two ways:

a) by using child properties with a list of embedded beans;
b) by using separate beans linked by parent property.

Example 1

```
<property name="typeMapping">
    <list>
    <bean class="org.jahia.modules.external.cmis.CmisTypeMapping"
id="cmis_document" p:jcrName="cmis:document"
p:cmisName="cmis:document">
        <property name="children">
            <list>
                <bean
class="org.jahia.modules.external.cmis.CmisTypeMapping"
p:jcrName="cmis:image" p:cmisName="cmis:image">
                </bean>
            </list>
        </property>
    </bean>
    </list>
</property>
```

Example 2

```
<property name="typeMapping">
    <list>
    <bean class="org.jahia.modules.external.cmis.CmisTypeMapping"
id="cmis_document" p:jcrName="cmis:document"
p:cmisName="cmis:document">
    </bean>
    <bean class="org.jahia.modules.external.cmis.CmisTypeMapping"
p:jcrName="cmis:image" p:cmisName="cmis:image">
        <property name="parent" ref="cmis_document"/>
    </bean>
    </list>
</property>
```

The first example is more visual while the second one is more flexible. You may even inherit mappings even from other modules. **Don't forget to include then parent bean in the typeMapping list.**

Jahia Solutions Group SA

9 route des Jeunes,
CH-1227 Les acacias
Geneva, Switzerland

http://www.jahia.com