FREE eBook

LEARNING alfresco

#alfresco

Table of Contents

About

Chapter 1: Getting started with alfresco

Remarks

Available Editions

Edition	Commercial Support	Production Ready	Vendor	Last Release Date
Alfresco One	Yes	Yes	Alfresco Software Inc.	
Alfresco Community Edition	No	No	Alfresco Software Inc.	
LXCommunity ECM	Yes	Yes	Loftux AB	

Versions

Examples

Installation or Setup

Development and Evaluation Installations

Alfresco provides a number of different installers for different operating systems and platforms. However, these installers are not recommended for production environments.

https://www.alfresco.com/alfresco-community-download

https://sourceforge.net/projects/alfresco - more Alfresco instalation files, and separate modules a avaiable also there

Production Installations

You can install Alfresco on Ubuntu distributions for by using Alfresco Ubuntu Install script from Loftux AB.

While you are installing using the script, you can select between Alfresco Community Edition (No commercial support) and LXCommunity ECM (With commercial support).

https://loftux.com/en/products-and-add-ons/alfresco-utilities

Addons

Additionally you can extend you Alfresco with community created addons- they are avaiable on https://addons.alfresco.com

About Alfresco

Alfresco is ECM (enterprise content management) system, based on java frameworks and javascript. Alfresco (as a "repository" is the base core of the Alfresco as a product).

It provides eg. encrypted content store, specific permissions settings, content types with specific properties (like name, date of creation, also custom properties can be included).

Alfresco also provides Workflow processes- separately or with documents, workflows are specified by diagram flow (Activiti and jbpm engine is supported; jbpm deprecated).

Product also partly supports Sharepoint (via Alfresco office services- "aos", CIFS, and other features. Key features are:

- Content storage
- Content retrieval
- Content modeling
- Query interface
- Access control
- Audit
- Versioning

User content is wrapped with the security layer- binary content is saved on the disk, but the structure and metadata are saved in the database.

This part- "repository" of the Alfresco as product provides REST API for the extensions like Alfresco Share.

Alfresco Installation Link For Windows

http://docs.alfresco.com/community/tasks/simpleinstall-community-win.html

For Linux

http://docs.alfresco.com/community/tasks/simpleinstall-community-lin.html

Brief Description about Installation

http://docs.alfresco.com/community/concepts/install-community-intro.html

Read Getting started with alfresco online: https://riptutorial.com/alfresco/topic/3748/getting-started-with-alfresco

Chapter 2: Administration

Examples

Starting and Stopping

To start Alfresco:

1. Switch to the alfresco user
2. Change to the $ALFRESCO_HOME directory
3. Run `./alfresco.sh start`

To stop Alfresco:

1. Switch to the alfresco user
2. Change to the $ALFRESCO_HOME directory
3. Run `./alfresco.sh start`

Logging

Alfresco logs live in $ALFRESCO_HOME/tomcat/logs/catalina.out.

Backups

There are many ways to backup an Alfresco system. It is important that you backup the database as well as the content store. You may also want to back up the Solr indices.

Assuming you installed using the binary installer, and everything lives in $ALRESCO_HOME, you can backup the database like this:

1. Stop Alfresco
2. Switch to the alfresco user
3. Change to the $ALFRESCO_HOME/postgresql/bin directory
4. Dump the database with `./pg_dump alfresco --user alfresco > $ALFRESCO_HOME/alf_data/db-backup.sql`. You might be prompted for a password. It should be the same thing as the admin password you provided during installation. If not, check $ALFRESCO_HOME/tomcat/shared/classes/alfresco-global.properties for the database password.

Now you have your database backed up. It is important to do that first. The next step is to backup the content store.

1. Change to the $ALFRESCO_HOME/alf_data/contentstore.deleted directory.
2. Delete everything in here. There's no reason to keep those files around and no reason to back them up.
3. Change to the $ALFRESCO_HOME/alf_data directory.

4. Tar up the whole thing, which will also include the database backup you created in the previous step assuming you placed it in the alf_data directory. Run `tar czvf ~/alfresco-backup.tar.gz .`.

The content of that TAR file now has everything you need to restore your working system.

Auditing

Auditing is an Alfresco feature that allows the user to trace and log some specific events during ECM platform usage.

Enable auditing

To enable auditing you have to add some lines of configuration to the `alfresco-global.properties` file, which resides in `tomcat/shared/classes/`

```
audit.enabled = true
audit.alfresco-access.enabled=true
```

You have to save changes to the `alfresco-global.properties` file and restart the Alfresco server, in order to enable auditing.

Auditing default configuration

Here the complete list of configuration properties that can be overridden modifying the `alfresco-global.properties` file:

```
# Audit configuration

audit.enabled=true
audit.tagging.enabled=true
audit.alfresco-access.enabled=false
audit.alfresco-access.sub-actions.enabled=false
audit.cmischangelog.enabled=false
audit.dod5015.enabled=false

# Setting this flag to true will force startup failure when invalid audit configurations are
detected

audit.config.strict=false

# Audit map filter for AccessAuditor - restricts recorded events to user driven events. In
this case it neglect events issued by a System or a null user, the content or folder path is
under /sys:archivedItem or under /ver: and the node type is not cm:folder, cm:content or
st:site

audit.filter.alfresco-access.default.enabled=false
audit.filter.alfresco-access.transaction.user=~System;~null;.*
audit.filter.alfresco-access.transaction.type=cm:folder;cm:content;st:site
audit.filter.alfresco-access.transaction.path=~/sys:archivedItem;~/ver:;.*

#The default to preserve all cm:auditable data on a node when the process is not directly
driven by a user action
```

```
system.auditableData.preserve=${system.preserve.modificationData}

#Specific control of how the FileFolderService treats cm:auditable data when performing moves

system.auditableData.FileFolderService=${system.auditableData.preserve}

#Specific control of whether ACL changes on a node trigger the cm:auditable aspect

system.auditableData.ACLs=${system.auditableData.preserve}
```

As usual you have to save changes to the alfresco-global.properties file and restart the Alfresco server, in order to enable these modifications.

Audit filters

Audit filters are properties that specify the strategy used to filter audit events by using particular regular expression to include or exclude events. Both custom and default audit filters can be added as overrides in the `alfresco-global.properties` configuration file.

The anatomy of an audit filter property is the following:

```
audit.filter.<data_producer>.<path>
```

where `<data-producer>` is one of the Alfresco built-in data producers:

1. `alfresco-access`: a wide group of high level events such as logins (both successful and failed), property updates, CRUD on nodes, content reads/updates, aspect addition and removal, versioning, check-in/check-out operations
2. `alfresco-node`
3. `alfresco-api`: events issued by the call of low level API methods and services. For example it can be used to list SearchServices search list parameters, properties listing using PropertyServices, operations on nodes using NodeServices and so on.

and `path` is the real path value to filter against.

Property names have an audit.filter.* prefix and use '.' as a separator where as components of rootPath and keys in the audit map use '/'.

Lists are evaluated from left to right and if no match is made by the end of the list the value is rejected. If there is not a property for a given value or an empty list is defined any value is accepted.

Each regular expression in the list is separated by a semicolon (';'). Expressions that include a semicolon can be escaped using a ''.

Note that if the `audit.config.strict` flag is set to true Alfresco startup will fail in case of invalid audit configurations detection.

An expression that starts with a `'~'` indicates that any matching value should be rejected. If the first character of an expression needs to be a `'~'`, it can be escaped with a `'\'`.

Adding `.*` at the end of a filter will include all values that have not been specifically excluded

Filters can be one of the following:

`transaction.user` - specifies what user(s) actions will/will not be audited. For example: Actions from all users except for 'System' will be audited

`transaction.type` - actions that are performed against the specified document type will be audited.

`default.path` - actions that occur on documents within the specified path will be audited

`transaction.action` - specifies what actions will and won't be audited. Some of the auditing events that can be enables or disabled using this property are: READ, MOVE, COPY, CHECK IN, CHECK OUT, CANCEL CHECK OUT, CREATE VERSION, readContent, addNodeAspect, deleteNodeAspect, updateNodeProperties.

For more information about audit filters:

https://github.com/tsgrp/OpenContent/wiki/Alfresco-Audit-Configuration

http://docs.alfresco.com/5.1/concepts/audit-example-filter.html

Read Administration online: https://riptutorial.com/alfresco/topic/6271/administration

Chapter 3: Alfresco model with dynamic list

Examples

Basic example

content-model.xml :

```
....
<constraints>
    <constraint name="my:aConstraintList"
type="x.y.z.project.model.constraint.AConstraintList">
    </constraint>
....
<property name="my:aValue">
    <title>My Value</title>
    <type>d:text</type>
    <constraints>
          <constraint ref="my:aConstraintList"></constraint>
    </constraints>
 </property>
```

and the Java class declared before :

```
public class AConstraintList extends ListOfValuesConstraint implements Serializable {

    ....
    @Override
    public final List<String> getAllowedValues() {
        // Return here the list of values. Enum, call a webservice, etc.
    }

     @Override
     public final String getDisplayLabel(final String value, final MessageLookup
messageLookup) {
         // Return here the label for the value
     }
}
```

Read Alfresco model with dynamic list online: https://riptutorial.com/alfresco/topic/6272/alfresco-model-with-dynamic-list

Chapter 4: Behaviour and Policy

Examples

Auto Version File If Exists With The same name

If file exists with same name then it will update the file with new version.

For register bean Class in `service-context.xml` file

```
<bean id="autoVersionByNameBehaviour" class="test.demoamp.AutoVersionByNameBehaviour" init-
method="init">
     <property name="policyComponent" ref="policyComponent"/>
     <property name="nodeService" ref="NodeService"/>
     <property name="contentService" ref="ContentService"/>
     <property name="siteService" ref="SiteService" />
     <property name="fileFolderService" ref="FileFolderService"/>

     <property name="activityService" ref="activityService"/>
     </bean>
```

and the java class

```
import java.net.URLEncoder;

import org.alfresco.model.ContentModel;
import org.alfresco.repo.node.NodeServicePolicies;
import org.alfresco.repo.policy.Behaviour;
import org.alfresco.repo.policy.JavaBehaviour;
import org.alfresco.repo.policy.PolicyComponent;
import org.alfresco.service.cmr.activities.ActivityService;
import org.alfresco.service.cmr.model.FileFolderService;
import org.alfresco.service.cmr.model.FileInfo;
import org.alfresco.service.cmr.repository.ChildAssociationRef;
import org.alfresco.service.cmr.repository.ContentReader;
import org.alfresco.service.cmr.repository.ContentService;
import org.alfresco.service.cmr.repository.ContentWriter;
import org.alfresco.service.cmr.repository.NodeRef;
import org.alfresco.service.cmr.repository.NodeService;
import org.alfresco.service.cmr.site.SiteInfo;
import org.alfresco.service.cmr.site.SiteService;
import org.apache.commons.io.FilenameUtils;
import org.json.JSONStringer;
import org.json.JSONWriter;

public class AutoVersionByNameBehaviour
implements NodeServicePolicies.OnCreateNodePolicy {
    private PolicyComponent policyComponent;
    private NodeService nodeService;
    private ContentService contentService;
    private ActivityService activityService;
    private SiteService siteService;
    private FileFolderService fileFolderService;

    public void init() {
        this.policyComponent.bindClassBehaviour(NodeServicePolicies.OnCreateNodePolicy.QNAME,
ContentModel.TYPE_CONTENT, (Behaviour)new JavaBehaviour((Object)this, "onCreateNode",
Behaviour.NotificationFrequency.TRANSACTION_COMMIT));
    }

    public void onCreateNode(ChildAssociationRef childAssocRef) {
        NodeRef previouslyExistentDoc;
        NodeRef uploadedNodeRef = childAssocRef.getChildRef();
        if (this.nodeService.exists(uploadedNodeRef) && this.isContentDoc(uploadedNodeRef) &&
(previouslyExistentDoc = this.existedPreviousDocument(uploadedNodeRef)) != null) {
            ContentReader reader = this.contentService.getReader(uploadedNodeRef,
ContentModel.PROP_CONTENT);
            ContentWriter writer = this.contentService.getWriter(previouslyExistentDoc,
ContentModel.PROP_CONTENT, true);
            writer.putContent(reader);
            this.nodeService.addAspect(uploadedNodeRef, ContentModel.ASPECT_HIDDEN, null);
            this.postActivityUpdated(previouslyExistentDoc);
            this.nodeService.deleteNode(uploadedNodeRef);
        }
    }

    private void postActivityUpdated(NodeRef nodeRef) {
        SiteInfo siteInfo = this.siteService.getSite(nodeRef);
        String jsonActivityData = "";
        try {
            JSONWriter jsonWriter = new JSONStringer().object();
            jsonWriter.key("title").value((Object)this.nodeService.getProperty(nodeRef,
ContentModel.PROP_NAME).toString());
            jsonWriter.key("nodeRef").value((Object)nodeRef.toString());
            StringBuilder sb = new StringBuilder("document-details?nodeRef=");
            sb.append(URLEncoder.encode(nodeRef.toString(), "UTF-8"));
            jsonWriter.key("page").value((Object)sb.toString());
            jsonActivityData = jsonWriter.endObject().toString();
        }
        catch (Exception e) {
            throw new RuntimeException(e);
        }
        FileInfo fileInfo = this.fileFolderService.getFileInfo(nodeRef);
        this.activityService.postActivity("org.alfresco.documentlibrary.file-updated",
siteInfo == null ? null : siteInfo.getShortName(), siteInfo == null ? null :
"documentLibrary", jsonActivityData, null, fileInfo);
    }

    private boolean isContentDoc(NodeRef nodeRef) {
        return
this.nodeService.getType(this.nodeService.getPrimaryParent(nodeRef).getParentRef()).isMatch(ContentMode

    }

    private NodeRef existedPreviousDocument(NodeRef currentNodeRef) {
        String fileName =
AutoVersionByNameBehaviour.cleanNumberedSuffixes(this.nodeService.getProperty(currentNodeRef,
ContentModel.PROP_NAME).toString());
        if (!fileName.equals(this.nodeService.getProperty(currentNodeRef,
ContentModel.PROP_NAME).toString())) {
            NodeRef folder = this.nodeService.getPrimaryParent(currentNodeRef).getParentRef();
            return this.nodeService.getChildByName(folder, ContentModel.ASSOC_CONTAINS,
fileName);
        }
        return null;
    }
    public static String cleanNumberedSuffixes(String fileName) {
        String cleanedFileName = fileName;
        String baseName = FilenameUtils.getBaseName((String)fileName);
        if (baseName.indexOf("-") != -1 &&
AutoVersionByNameBehaviour.isInteger(baseName.substring(baseName.lastIndexOf("-") + 1,
baseName.length()))) {
            return baseName.substring(0, baseName.lastIndexOf("-")) +
FilenameUtils.EXTENSION_SEPARATOR_STR + FilenameUtils.getExtension((String)fileName);
        }
        return cleanedFileName;
    }

    public static boolean isInteger(String s) {
        boolean isValidInteger = false;
        try {
            Integer.parseInt(s);
            isValidInteger = true;
        }
        catch (NumberFormatException var2_2) {
            // empty catch block
        }
        return isValidInteger;
    }

    public void setPolicyComponent(PolicyComponent policyComponent) {
        this.policyComponent = policyComponent;
    }

    public void setNodeService(NodeService nodeService) {
        this.nodeService = nodeService;
    }

    public void setContentService(ContentService contentService) {
        this.contentService = contentService;
    }

    public void setActivityService(ActivityService activityService) {
        this.activityService = activityService;
    }

    public void setSiteService(SiteService siteService) {
        this.siteService = siteService;
    }

    public void setFileFolderService(FileFolderService fileFolderService) {
        this.fileFolderService = fileFolderService;
    }
```

}

Read Behaviour and Policy online: https://riptutorial.com/alfresco/topic/9696/behaviour-and-policy

Chapter 5: T-SQL script for creating Alfresco database in SQLServer 2008 - 2014

Introduction

It's useful to have a T-SQL script for creating and configuring a new database and user for Alfresco Installation purposes. It's boring and time-consuming to jump around many pages on MSDN.

I provide the script hereafter:

Remarks

Guidelines for preparing Alfresco Instalation to a SQLServer database, can be read here:

- [http://docs.alfresco.com/3.4/tasks/sqlserver-config.html][1]

Examples

T-SQL_script_4_alfresco

```
/* creates a database for Alfresco, on SQLServer 2008- 2014 */
use master;
GO
CREATE DATABASE alfresco;
GO
/* creates a new LOGIN and associated User
use alfresco;
GO
CREATE LOGIN alfresco WITH PASSWORD = 'alfresco';
GO
use alfresco;
go
CREATE USER alfresco FOR LOGIN alfresco;
GO

/* Now try to add alfresco user to the db_owner Role
   NOTICE: coinnect to alfresco database before
   you can also connect as a local Windows user,
   in order to successfully execute the followings:  */
use alfresco;
GO
EXEC sp_addrolemember N'db_owner', N'alfresco';
GO

/* sets Isolation level */
ALTER DATABASE alfresco SET ALLOW_SNAPSHOT_ISOLATION ON;
GO

/* creates and sets the alfesco schema as the default one */
```

```
use alfresco;
go
CREATE SCHEMA alfresco AUTHORIZATION alfresco;
GO
ALTER USER alfresco WITH DEFAULT_SCHEMA = alfresco;
GO

/* tests table creation */
drop table _buttamiVia_;
GO
create table _buttamiVia_
( id int not null );
GO
```

Read T-SQL script for creating Alfresco database in SQLServer 2008 - 2014 online: https://riptutorial.com/alfresco/topic/9370/t-sql-script-for-creating-alfresco-database-in-sqlserver-2008---2014

Chapter 6: Web Scripts

Introduction

Webscripts are functional modules in the Alfresco, which can just show some informations or also make some things inside the Alfresco (eg. run workflow, working with files, users, groups, or other entities).

Every webscripts has two main parts- code (.js, .java) and Freemaker template (.ftl)

Webscript can have also additional .properties file with text strings used in. Parts are paired in the context.xml file (Spring framework logic).

It's binded to any URL described in this file.

Examples

Hello World Web Script

Let's make a hello world web script. Web scripts have a descriptor, a controller, and, optionally, a view. These files must follow a naming convention.

This descriptor is named helloworld.get.desc.xml.

```
<webscript>
  <shortname>Hello World</shortname>
  <description>Hello world web script</description>
  <url>/example/helloworld?name={nameArgument}</url>
  <authentication>user</authentication>
</webscript>
```

You can see that the descriptor declares that this web script will be mapped to a URL, "/example/helloworld", and that it requires user authentication. The descriptor also declares an argument called name.

Here is the controller. It is named helloworld.get.js.

```
model.foo = "bar";
```

This controller is written in JavaScript but controllers can also be written in Java. With a bit more work you can write controllers in other languages too.

This controller doesn't do much. It just adds a new variable to the model called "foo" and gives it a value of "bar".

Your controller has access to a variety of root scoped variables which are all documented in the official documentation.

Finally, let's look at the view. It's named helloworld.get.html.ftl

```
<html>
  <body>
    <p>Hello, ${args.name!"name not specified"}!</p>
    <p>Foo: ${foo}</p>
  </body>
</html>
```

You can see from the name that this view is implemented as a Freemarker template and outputs HTML. This view grabs the value of "foo" from the model and it also grabs the name argument that was passed in to the web script. If a name argument is not specified the template provides some default text.

If you wanted to produce XML or JSON instead you can--just change the name, then update your template implementation accordingly.

Deployment

Web scripts can be deployed to the classpath or uploaded to the repository. For example, to deploy this web script by uploading to the repository, follow these steps:

1. Upload these three files to Data Dictionary/Web Scripts Extensions
2. Refresh the Web Scripts by going to http://localhost:8080/alfresco/s/index and clicking "Refresh Web Scripts".
3. Navigate to the web script by going to http://localhost:8080/alfresco/s/example/helloworld?name=Jeff

Folder Maker: A Web Script that handles POST

The Hello World Web Script handles GET HTTP methods. But what if you want to create data on the server? For that your web script should handle POST.

Here is a simple example that creates new folders in Company Home. It is invoked by making a POST call with a JSON body that looks like:

```
{'name':'testfolder'}
```

Optionally, you could add a title or a description to the folder by passing those in as part of the body, like:

```
{
  'name':'testfolder',
  'title':'test title',
  'description':'test description'
}
```

The descriptor is called foldermaker.post.desc.xml:

```
<webscript>
```

```
  <shortname>Folder Maker</shortname>
  <description>Creates folders</description>
  <family>Examples</family>
  <url>/example/folders</url>
  <format default="json"></format>
  <authentication>user</authentication>
</webscript>
```

The optional "family" element is a convenient way to group web scripts in the web script index. The "format" element declares that this web script returns JSON.

The controller is called foldermaker.post.json.js:

```
var name = title = desc = null;

var name = json.get('name');

try {
  title = json.get('title');
} catch (err) {}

try {
  desc = json.get('description');
} catch (err) {}

var folder = companyhome.createFolder(name);

var needsSave = false;

if (title != null) {
  folder.properties['cm:title'] = title;
  needsSave = true;
}

if (desc != null) {
  folder.properties['cm:description'] = desc;
  needsSave = true;
}

if (needsSave) {
  folder.save();
}

model.id = folder.nodeRef.toString();
model.name = name;
model.title = title;
model.description = desc;
```

Notice that this controller has "json" in its name. This tells Alfresco to expect a JSON payload. Alfresco will automatically parse the JSON and put it in a root variable called "json".

The controller grabs the name, title, and description from the JSON and creates the folder using the root scope variable called "companyhome".

If a title or description is passed in those properties get saved.

The values that were passed in as well as the new folder's node reference get set on the model before handing control over to the view.

The view is named foldermaker.post.json.ftl:

```
{
  <#if title??>
  "title": "${title}",
  </#if>
  <#if description??>
  "description": "${description}",
  </#if>
  "id": "${id}",
  "name": "${name}"
}
```

This freemarker just echoes back the values set on the model as JSON. The title and description may not always be present, so the view uses the Freemarker null check built-in in an if statement to avoid returning those if they were not set.

Any HTTP client can be used to test this web script out. Here's what it would look like using curl:

```
jpotts$ curl -uadmin:admin -H "content-type: application/json" -X POST
"http://localhost:8080/alfresco/s/example/folders" -d "{'name':'testfolder','title':'test
title', 'description':'test desc'}"
{
  "title": "test title",
  "description": "test desc"
  "id": "workspace://SpacesStore/cc26a12f-306b-41f1-a859-668f11fc2a54",
  "name": "testfolder"
}
```

Note that curl is passing in basic auth credentials. In this case it is using "admin". Because this example creates items in Company Home, you must use a user that has the appropriate permissions to do that.

This web script has no real error checking. If you don't pass in a name or you pass in a name that has already been used you will see an error.

Read Web Scripts online: https://riptutorial.com/alfresco/topic/6066/web-scripts

Credits

S. No	Chapters	Contributors
1	Getting started with alfresco	bhagyas, Community, vikash, xxxvodnikxxx
2	Administration	abarisone, Jeff Potts
3	Alfresco model with dynamic list	Akah
4	Behaviour and Policy	vikash
5	T-SQL script for creating Alfresco database in SQLServer 2008 - 2014	ciroBorrelli
6	Web Scripts	Jeff Potts, xxxvodnikxxx