Adobe® Experience Manager Sites: Advanced Developer

Student Workbook

ADOBE® TRAINING SERVICES

AEM Sites: Advanced Developer

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

201406

Table of Contents

AEM 6.0 Advanced Developer V

Introduction V

Standards and Open Source VI
Just a review – The AEM Architecture Stack VII
Jackrabbit Oak. VIII

01 Development Models for Teams 1-1

Team Development Models 1-1
Maven 1-3
The Maven POM. 1-3
General project information 1-3
Build settings 1-3
Build environment 1-4
POM relationships 1-4
Snapshot Versions 1-5
Property References 1-6
Plugins 1-6
Use Maven to Create a Bundle. 1-6
Installing Maven - Instructions for Windows 1-7
Setting Up the New Project 1-8
VLT 1-17
Mapping CRX/CQ Content to the File System. 1-17
VLT and SVN 1-18
EXERCISE 1.2 - Installing the VLT Tool. 1-18
EXERCISE 1.3 - Use VLT to change content from the command line 1-19
EXERCISE 1.4 - Installing Eclipse 1-22
EXERCISE 1.1 - Installing Maven - Instructions for Mac. 1-6

02 OSGi Components, Annotations 2-1

OSGi 2-1
OSGi Alliance. 2-1
OSGi Overview 2-1
Apache Felix 2-2
Managing your OSGi applications. 2-2
OSGi Bundles, Services, Components 2-3
Bundles 2-3
Dependency Management Resolution 2-4
Service Registry Model. 2-5
Dynamic service lookup. 2-6
OSGi Service Advantages 2-7
Declarative Services 2-8
Components 2-9
Bundles 2-11
Annotations 2-11
@Component 2-12
@Activate, @Deactivate, and @Modified 2-12
@Service 2-13
@Reference 2-13

@Property 2-13
Java Compiler Annotations 2-14
@Override 2-14
@SuppressWarnings 2-14
Configurable Services 2-15
OSGi Component 2-16
EXERCISE - 2.1 2-16

03 Sling, Resources, REST 3-1

Representational State Transfer (REST) 3-1
Advantages of REST 3-1
REST and Apache Sling 3-2
Apache Sling 3-2
Default GET Servlet 3-2
Sling and Resources 3-4
Resource-First Request Processing 3-5
Resource and representation 3-5
Basic Request Processing 3-6
Resource Resolver 3-7
Mappings for Resource Resolution 3-8
Mapping Nodes to Resources 3-10
Adapting Resources 3-10
.adaptTo() Use Cases 3-10
Additional Information 3-13
Servlets 3-13

04 Sling Events 4-1

Listening to OSGi Events 4-2
Publishing Events 4-2
Job Events 4-3
Sending Job Events 4-3
Admin User Interface 4-3
EXERCISE 4.1 - Event Handling 4-4
ReplicationLogger.java 4-7

05 Sling Scheduling 5-1

OSGi Service Fired By Quartz 5-2
Scheduling At Periodic Times 5-2
Preventing Concurrent Execution 5-2
Quartz Trigger Syntax 5-3
Programmatically Scheduling Jobs 5-3

06 JCR Basics, Content Modeling, Indexing, Search 6-1

Repository Basics 6-1
JCR Review 6-1
JCR Features 6-2
Repository Model 6-2
Node Types 6-2
Node Type Definitions 6-3
Node Type Inheritance 6-4
Content Modeling: David's Model 6-4
Data First. Structure Later. Maybe 6-5
Drive the content hierarchy, don't let it happen 6-5
Workspaces are for clone(), merge() and update() 6-5

Beware of Same Name Siblings 6-6
References are considered harmful 6-6
Files are Files are Files. 6-6
IDs are evil 6-6
Repository Internals 6-6
BlobStore (Jackrabbit 3.x) and DataStore (Jackrabbit 2.x) 6-6
Microkernels 6-7
Journal. 6-8
Query Index 6-8
Repository Configuration. 6-9
Basic Content Access 6-9
Batch Processing 6-10
CRX Search Features not specified by the JCR 6-10
Query Syntax 6-10
Basic AQM Concepts. 6-11
AQM Concepts - Constraints 6-11
Search Basics 6-11
Query Examples - SQL2 6-12
Java Query Object Model (JQOM) 6-12
JQOM Examples. 6-13
Search Performance 6-13
Testing Queries 6-14
EXERCISE 6.1 - Debugging and Logging Queries 6-14
EXERCISE 6.2 – Search 6-15

07 JCR Versioning, Observation 7-1

Observation 7-1
Event Model. 7-1
Scope of Event Reporting 7-2
The Event Object 7-2
Event Types 7-2
Event Information 7-2
Externally Caused NODE_MOVED Event 7-3
User ID 7-3
User Data 7-3
Event Date 7-4
Event Bundling 7-4
Event Ordering 7-4
Asynchronous Observation 7-4
Observation Manager 7-4
Adding an Event Listener 7-5
Event Filtering 7-5
Access Privileges 7-5
Event Types 7-5
Local and Nonlocal 7-6
Node Characteristics 7-6
Location 7-6
Identifier 7-6
Node Type 7-6
Re-registration of Event Listeners 7-6
Event Iterator 7-6
Listing Event Listeners 7-7
EventListenerIterator. 7-7
Removing Event Listeners 7-7
User Data 7-7
Journaled Observation 7-7
Event Journal. 7-7

Journaling Configuration 7-8
Event Bundling in Journaled Observation 7-8
Importing Content 7-8
Exceptions 7-8
Event topics used on AEM-level 7-9

08 Jackrabbit Oak 8-1

Oak Architecture (also known as Hamburger Architecture) 8-2
The MicroKernel Data Model: 8-4
DocumentMK 8-4
Implementation 8-4
Documents 8-5
Revisions 8-5
Branches 8-6
Previous Documents 8-6
Background operations 8-6
Cluster node metadata 8-6
SegmentMK 8-6
Implementation 8-7
Segments 8-7
Journals 8-7
Records 8-7
Binary Data Stores 8-8
DataStores and MikroKernels 8-8
Introduction to MongoDB 8-10
Basic operations in MongoDB 8-10
Read operations (queries) 8-10
Document models in MongoDB 8-13
Sharding 8-15
Reasons for not sharding MongoDB 8-18
Replica sets 8-18

9 Users, Groups, Permissions 9-1

Permissions and ACLs 9-1
Actions 9-3
Access Control Lists and how they are evaluated 9-3
Concurrent permission on ACLs 9-5

10 Testing (Sling and Maven) 10-1

Junit 10-1
EasyMock 10-2
PowerMock 10-4
EXERCISE 10.1 - Unit Tests using Junit and Maven 10-4
EXERCISE 10.2 - Unit tests with Junit, EasyMock, PowerMock and Maven 10-8
EXERCISE 10.3 - Running tests on the server (Sling-based tests) 10-10
Server-side tests depending on an injected environment object 10-14
Running scriptable server side tests 10-15
Testing framework 10-20

11 Deployment and Packaging 11-1

Packaging 11-1
Considerations 11-1
Packaging - Style 1. 11-2
How To Create The config Package 11-2
Packaging - Style 2. 11-2
Runmodes 11-3
Setting Runmodes 11-3
Configurations per run mode 11-3
Configurations For Different Runmodes 11-4
Configurations Per Run Mode 11-4
Deployment. 11-4

12 Dispatcher, Reverse Replication 12-1

Dispatcher 12-1
The Basics Revisited 12-2
How the Dispatcher Returns Documents 12-3
Cache Invalidation (Expiration) 12-3
The Dispatcher's Role in AEM Projects 12-3
Configuring the Cache - dispatcher.any. 12-3
What to Cache - the Rules Section 12-4
Denying Access - The Filter Section 12-4
Additional Information on configuring the Dispatcher. 12-5
Caching - Getting Better Performance. 12-6
Caching Queries 12-6
Caching Fragments 12-7
Caching and Periodic Importers 12-7
Advanced Dispatcher 12-7
Additional Dispatcher Performance Tips 12-7
Finding Server-side Execution Problems 12-8
Reverse Replication. 12-9
The Basics. 12-10
Reverse replication and clustering 12-10
Programmatically Triggering Reverse Replication. 12-11
Additional Information. 12-11

13 Content Automation, Periodic Importers 13-1

Custom Periodic Importers. 13-2
Example code 13-3
Alternative Approaches 13-3
Overview 13-4

14 Overlays, Extending Foundation Components, Content Reuse. . . . 14-1

Reuse: Overlays, Extending the Foundation Components 14-1
Use Cases 14-2
Beware. 14-2
Extending the Foundation Components. 14-2

15 Content Migration/Import 15-1

Vlt-based migration. 15-1
Sling POST Servlet 15-1

JCR API 15-2
EXERCISE 15.1 - Explore Approaches To Import Content From Legacy CMSs 15-2
Using vlt 15-2
Using The Sling POST servlet 15-6
Installing Curl 15-6
Using Curl 15-7
Using The JCR API 15-8
Upgrading to AEM 6.0 15-11
DataStore Migration 15-11

16 Higher Level APIs 16-1

EXERCISE 16.3 - Creating, tagging and activating an Asset using high- level APIs 16-8

17 Client Libraries (Extra Credit) 17-1

Client- or HTML Libraries 17-1
AEM has introduced a very interesting concept: Client Libraries, aka "clientlibs" 17-1
Client Library Conventions 17-1
Examples of Client Libraries 17-2
Include Client Libraries 17-3
Manage Client Libraries 17-4
Planning Client Libraries 17-4
EXERCISE 17.1 - Client Libraries (global) 17-5
EXERCISE 17.2 - Client Libraries (below Components) 17-9

Appendix 1 18-1

Useful API definitions 18-1
EXERCISE - Useful API definitions 18-1

AEM 6.0 Advanced

Developer

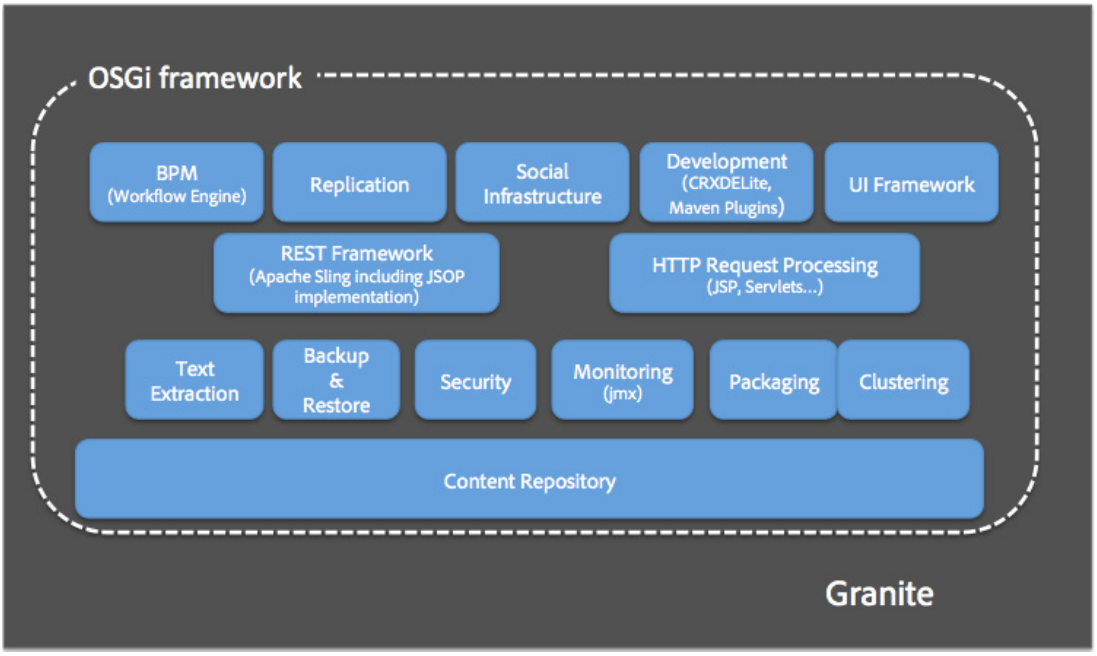
Introduction

AEM, part of the Adobe Experience Manager (AEM) is an enterprise-grade content management platform with a wide array of powerful features.

All of these things are done through a rich graphical interface accessible through any modern browser, enabling such desktop-like features as in-place editing of text and graphics, drag and drop of page elements and visual design of workflows.

The AEM application modules sit on top of the AEM shared framework [granite], which contains all functionality that is shared among all the application modules; for example mobile functionality, multi-site manager, taxonomy management, and workflow.

In addition to the shared application framework, the AEM applications (Sites, Assets, and Apps, etc) share the same infrastructure and same UI framework.

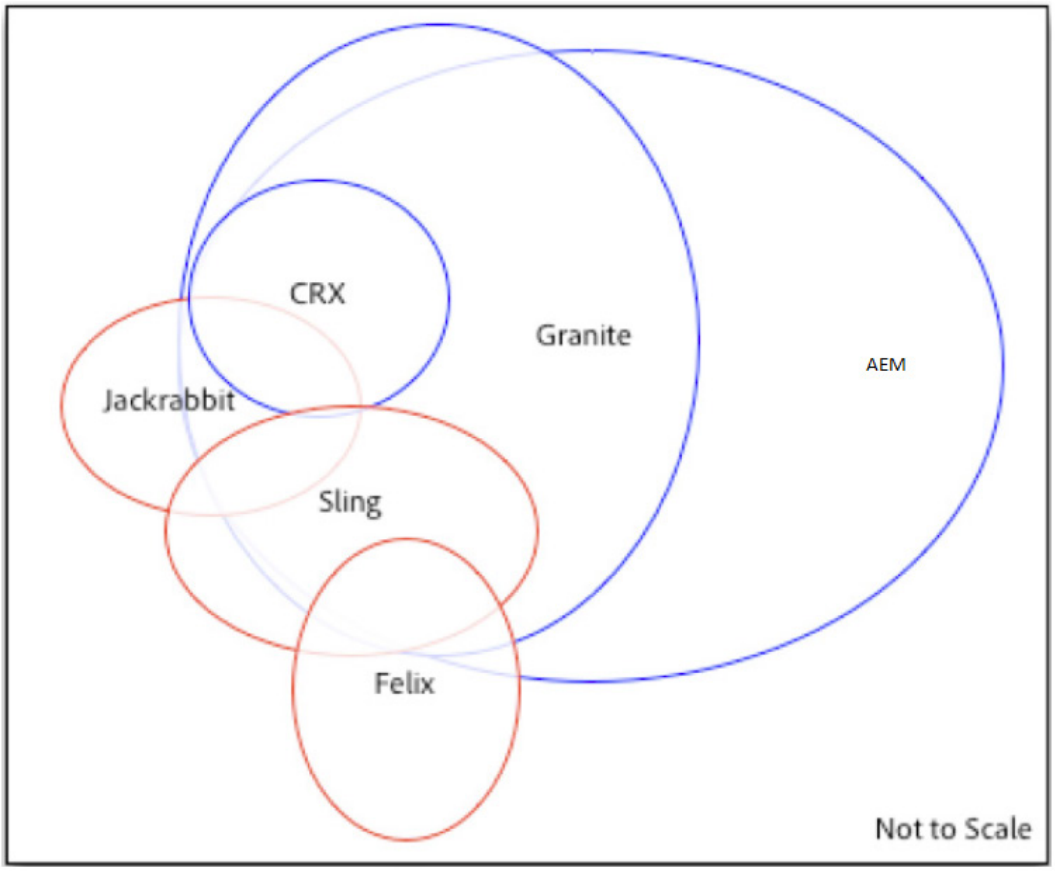


AEM is implemented as a Java web application. It runs in any server that supports the Java Servlet API 2.4 (or higher). AEM comes pre-configured with its own built-in servlet engine, but can also be installed in any compatible third-party application server. Since the system can be accessed from any computer with a modern web browser, no installation of client software is required, so even large installations with thousands of users can be rolled out and upgraded easily.

Standards and Open Source

AEM is built using standards and open source products. The most important of those are:

- 100% Java / J2EE
- JSR 283—a Java Content Repository (JCR)
- Apache Jackrabbit Oak—reference implementation for JSR 283
- Apache Felix—an OSGi framework
- Apache Sling—a Web framework for the Java platform designed to create content-centric applications on top of a JSR



Just a review – The AEM Architecture Stack

AEM is built within an OSGi application framework. OSGi is a dynamic module system for Java that provides a framework within which small, reusable standardized components can be composed into an application and deployed.

According to JSR-283, **the *Java Content Repository API defines an abstract model and a Java API for data storage and related services commonly used by content- oriented applications.***

A Java Content Repository is an object database that provides various services for storing, accessing, and managing content. In addition to a hierarchically structured storage, common services of a content repository are versioning, access control, full text searching, and event monitoring.

The JCR provides a generic application data store for both structured and unstructured content. File systems provide excellent storage for unstructured, hierarchical content. Databases provide excellent storage for structured data due to transactional services and referential integrity functionality.

The JCR provides the best of both data storage architectures, plus observation, versioning, and full text search.

Apache Sling is a web framework that uses a Java Content Repository, such as Apache Jackrabbit or Adobe CRX to store and manage content. Sling applications use either scripts or Java servlets, selected based on simple name conventions, to process HTTP requests in a RESTful way.

The Apache Sling framework is designed to expose a JCR content repository through an HTTP-based REST API. Both AEM's native functionality and the functionality of any website built with AEM are delivered through this framework.

Jackrabbit Oak

Jackrabbit Oak is an effort to implement a scalable and efficient hierarchical content repository for use as the foundation of modern, efficient web sites, and other demanding content applications.

Jackrabbit Oak implements the Java Content Repository (JCR) spec. The most used parts of JSR-283 are implemented, but Jackrabbit Oak does not aim to be a reference implementation of JSR-283. AEM 6.0 is built on top of Jackrabbit Oak.

Following are the goals of Jackrabbit Oak:

- Scalability: Oak supports big repositories and distributed repository with many cluster nodes.
- Improved throughput: Parallel write operations are possible using Oak.
- Oak Supports large number of child nodes.

For more information on Jackrabbit Oak, read the Jackrabbit Oak chapter.

01

Development Models for Teams

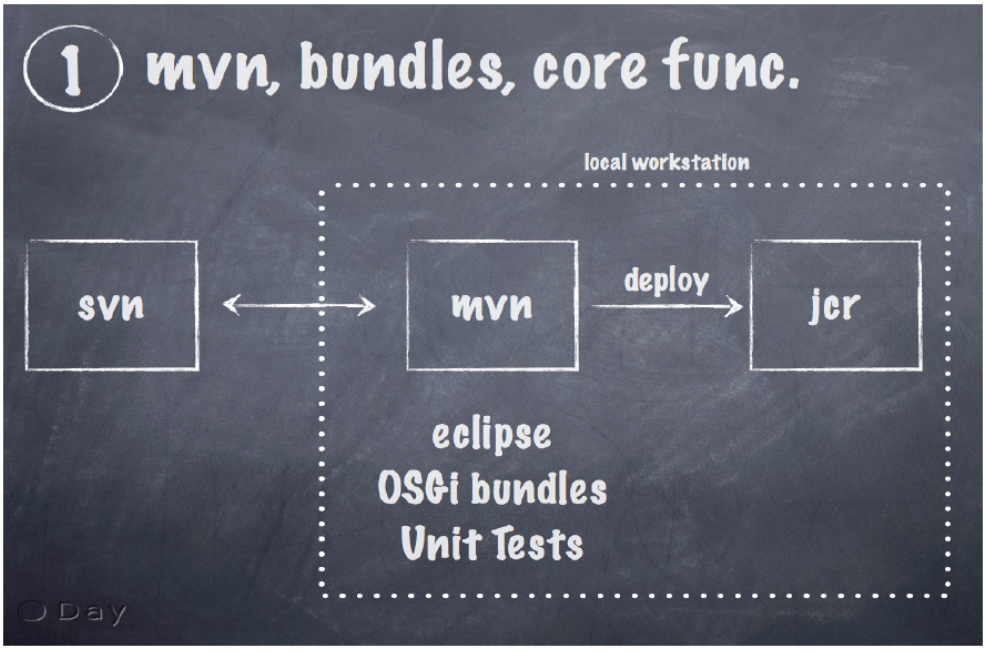
Team Development Models

In this section, we will look at Maven and vlt, and their use for managing projects.
We will:

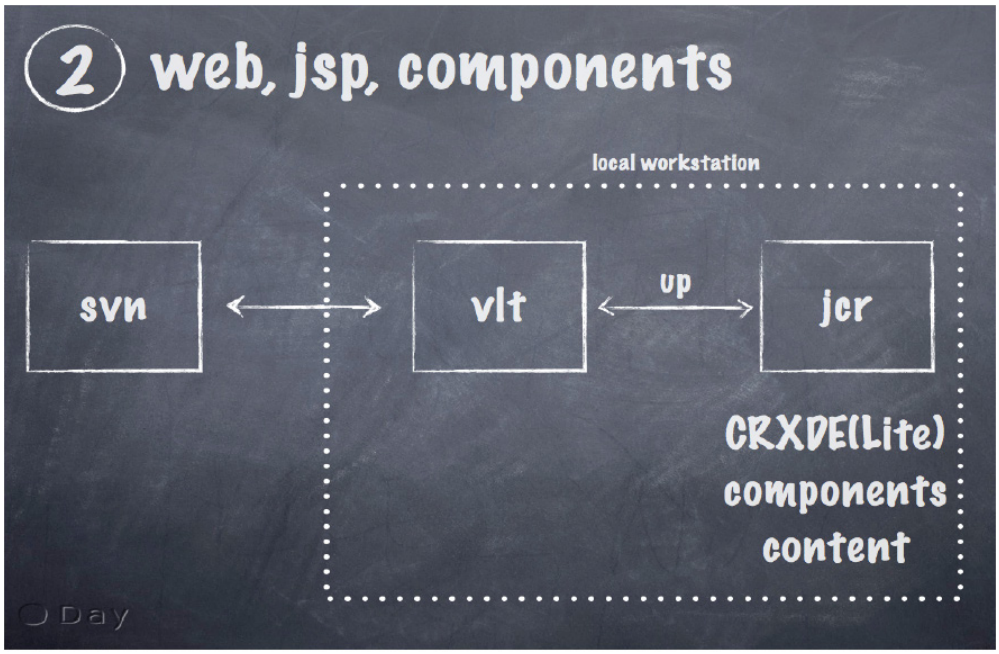
- Inspect the anatomy of the project
- Compile the project
- Deploy into a local AEM instance
- Use vlt to change locally and in the repository

Some of the notes given here on Maven were drawn from an online document which you can refer to for more details: http://www.sonatype.com/books/mvnrefbook/reference/index.html

In the first exercise we will use Maven to build and deploy a bundle to the jcr repository. In practice we would usually use Eclipse for editing the code and building OSGi bundles and Unit tests. An SVN repository could hold a common copy of the project.



In the second exercise, we see how to use vlt to upload and download content to the jcr repository. The developer can alter content checked out of an svn repository, and, to test it, use VLT to push those changes to his local CRX instance (using vlt checkin). Once the developer is satisfied with the changes, he or she can push them to the central SVN repository (using svn checkin) to share with other developers. In the second exercise, we see how to use vlt to upload and download content to the jcr repository.



Maven

Maven's primary goal is to allow a developer to easily and quickly comprehend the complete state of a development effort. To attain this goal there are several areas of concern that Maven attempts to deal with:

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Providing guidelines for best practice development
- Allowing transparent migration to new features

The Maven POM

Maven allows a project to build using its project object model (POM) and a set of plugins that are shared by all projects using Maven, providing a uniform build system. Once you familiarize yourself with how one Maven project builds you automatically know how all Maven projects build saving you immense amounts of time when trying to navigate many projects.

The POM is where a project's identity and structure are declared, builds are configured, and projects are related to one another. The POM is always in a file called pom.xml, in the base directory of the Maven project, so presence of a pom. xml file defines a Maven project. While pom.xml has some similarities to a make file or an Ant build file, the POM focusses on descriptions giving details such as where resources, source code, packages, are found. There are no explicit instructions for compiling, for example. The Maven POM is declarative, and in fact Maven can be used for building projects in other languages or even composing books or other documents.

There are four main sections of description and configuration:

General project information

This includes the project's name, the website url, the organization, and can include a list of developers and contributors along with the license for a project.

Build settings

In this section, we can change the ***directory settings of source*** and tests, add new ***plugins***, attach plugin goals to the lifecycle, and customize the site generation parameters.

Build environment

The build environment consists of profiles that can be activated for use in different environments. For example, during development you may want to deploy to a development server, whereas in production you want to deploy to a production server. The build environment customizes the build settings for specific environments and is often supplemented by a custom settings.xml in ~/.m2. The Build settings can include packaging details such as jar, war, ear, maven-plugin details.

POM relationships

Projects usually depend on other projects; they may inherit POM settings from parent projects, define their own coordinates, and may include submodules. The POM relationships section includes:

- groupId
- artifactId
- version
- Dependencies

Maven project POMs extent a POM called the Super POM which is located in `${M2_HOME}/lib/maven-3.0.3-uber.jar` in a file named `pom-4.0.0.xml` under the org. `apache.maven.project` package. This defines the central Maven repository; a single remote Maven repository with an ID of central. All Maven clients are configured to read from this repository by default, and it is also the default plugins repository. These can be overridden by a custom settings. xml file.

Project information such as change log documentation, dependency lists and unit test reports can be generated by Maven. It can be used to guide a project towards best practice development, for example by inclusion of unit tests as part of the normal build cycle, and through project workflow.

Dependencies in Maven use the `groupId`, `artifactId` and `version` attributes. A `modelVersion` attribute is also required as a minimum. The simplest Maven pom.xml file could look like this:

```
<project>
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.adobe</groupId>
      <artifactId>training</artifactId>
      <version>1.0-SNAPSHOT</version>
</project>
```

This does not specify any other projects, it has no dependencies, and it lacks basic information such as a name and a URL. If you were to create this file and then create the subdirectory src/main/java with some source code, running mvn package would produce a JAR in target/simple-project-1.jar. Some default folders are being used in this case, as they are not specified:

src/main

src/main/java

src/main/resources

src/test

src/test/java

src/test/resources

Snapshot Versions

Maven versions can contain the string "-SNAPSHOT," to indicate that a project is currently under active development. Deploying a snapshot means you are releasing a snapshot of a component at a specific time. Maven will expand the -SNAPSHOT token to a date and time value converted to UTC (Coordinated Universal Time) when you install or release this component. For example, if your project has a version of "1.0-SNAPSHOT" and you deploy this project's artifacts to a Maven repository at 8:08 PM on January 4th, 2012 UTC, Maven would expand this version to "1.0-20120104-200805-1".

SNAPSHOT versions are used for projects under active development. If your project depends on a software component that is under active development, you can depend on a SNAPSHOT release, and Maven will periodically attempt to download the latest snapshot from a repository when you run a build. Similarly, if the next release of your system is going to have a version "1.4", your project would have a version "1.4-SNAPSHOT" until it was formally released.

As a default setting, Maven will not check for SNAPSHOT releases on remote repositories. To depend on SNAPSHOT releases, users must explicitly enable the ability to download snapshots using a repository or pluginRepository element in the POM.

When releasing a project, you should resolve all dependencies on SNAPSHOT versions to dependencies on released versions. If a project depends on a SNAPSHOT, it is not stable as the dependencies may change over time. Artifacts published to non-snapshot Maven repositories such as http://repo1.maven.org/maven2 cannot depend on SNAPSHOT versions, as Maven's Super POM has snapshots disabled from the Central repository. SNAPSHOT versions are for development only.

Property References

Property references can be used to refer to properties from another part of the pom.xml file, java system properties, or implicit environment variables. The syntax is `${...}` with the property name in the curly brackets. Maven will replace the reference with the actual value of the property when it loads the pom.xml file.

The three supported implicit environment variables are:

- ***env*** - for environment variables such as PATH, included by `${env.PATH}`
- ***project*** - exposes the POM. You can use a dot-notated (.) path to reference the value of a POM element. `${project.groupId}` would give the groupId value, `com.adobe` in the earlier simple `pom.xml`.
- ***settings*** - exposes Maven settings information. For example, `${settings.offline}` would reference the value of the offline element in `~/.m2/settings.xml`.

Plugins

The core of Maven is very small and simple. Most of the functionality is provided through plugins. These do things like compiling source, packaging a WAR file, or running JUnit tests. The Maven Plugins are retrieved from the Maven Repository - either the Central Maven Repository, where the core Maven Plugins are located, or another specified repository. This means that if new functionality is added to a plugin, you can make use of it very simply, by updating a version number of a plugin in a single POM configuration file.

Use Maven to Create a Bundle

Before installing Maven, check that you already have Java 1.7 installed - at a command prompt type java -version to display the version that is installed. If you do not see Java version 1.7, then please install it first.

These exercises have been tested with Maven version 3.0.3, and this version of Maven is supplied to you as a zip file on the USB stick (in folder Distribution/Maven). Please use this version for the course, however if you need to obtain other versions in the future they can be found at http://maven.apache.org/download.html

EXERCISE 1.1 - Installing Maven - Instructions for Mac

1. Uncompress the file apache-maven-3.0.0-bin.tar.gz into the /usr/local folder. This should create a folder called apache-maven-3.0.0 with sub-folders “bin”, “boot”, “conf” and “lib”.

2. You may want to create a symbolic link to make it easier to work with and to avoid the need to change any environment configuration when you upgrade to a newer version. In a command window, type the following commands:

   - Change to /usr/local directory:

     ```
     cd /usr/local
     ```

   - Create a symbolic link for maven:

     ```
     ln -s apache-maven-3.0.3 maven
     ```

NOTE: You may need to use sudo for this command :
`sudo ln -s apache-maven-3.0.3 maven`

3. You will also need to have an environment variable for Maven.

   - Set the environment variable M2_HOME to the top-level directory you installed (in this example, /usr/local/maven):

     ```
     export M2_HOME=/usr/local/maven
     ```

4. Finally you need to make sure that the bin directory is in the command path:

   ```
   export PATH=${M2_HOME}/bin:${PATH}
   ```

5. To set these values automatically each time you log on, you can modify the .bash_profile script under your user directory by adding the following lines:

   ```
   export M2_HOME=/usr/local/maven

   export PATH=${M2_HOME}/bin:${PATH}
   ```

Installing Maven - Instructions for Windows

Installing Maven on Windows is very similar to installing Maven on Mac OSX, the main differences being the installation location and the setting of an environment variable. For this course we will use a Maven installation directory of `c:\Maven` although you can use the `Program Files` directory if you prefer, as long as you configure the proper environment variables.

1. Create a folder called "C:\Maven", then uncompress the file apache-maven-3.0.3-bin.zip into the new folder. You should now have a folder called "C:\Maven apachemaven-3.0.3" containing sub-folders "bin", "boot", "conf" and "lib".

2. Once you've unpacked Maven to the installation directory, you will need to set two environment variables; PATH and M2_HOME. To set these from the command-line, type the following commands:

```
C:\> set M2_HOME=c:\Maven\apache-maven-3.0.3

C:\> set PATH=%PATH%;%M2_HOME%\bin
```

3. Setting these environment variables on the command-line will allow you to run Maven in your current session, but unless you add them to the System environment variables through the control panel, you'll have to execute these two lines every time you log into your system. You should modify both of these variables through the Control Panel in Microsoft Windows.

4. Check that your settings are correct by typing `mvn -v` at the command line - you should see the Maven and Java versions and other information displayed. If this does not work as expected you may see an error message that says you also need to set a JAVA_HOME environment variable, if it was not created when you installed Java; for example:

```
C:\> set JAVA_HOME=C:\Program Files\Java\<java installation dir>
```

5. Again, this can be typed at the command prompt but should also be added through the `Control Panel>System>Advanced System Settings- Environment Variables...` button in order to persist the setting.

Setting Up the New Project

1. On Windows, create a new directory for the project called:

   ```
   C:\Adobe\AEM60AdvDev\
   ```

   and unzip sampleproject.zip, (on the USB stick in the Exercises folder) saving the files in the newly created directory.

2. On MAC, create a new folder for the project called /Adobe/AEM60AdvDev and unzip, sampleproject.zip (on the usb stick on the USB stick) saving the files in the newly created folder. It contains the skeleton for a project named "company" with 2 Maven modules:

   * CRX Package company-ui (that will contain jsps, content nodes, etc.)
   * OSGi bundle company-core

3. Inspect the parent pom.xml under the sampleproject directory. (Not the pom. xml in company-core or in company-ui). Notice the attributes discussed earlier: `modelVersion, groupId, artifactId` and `version`, and the use of `-SNAPSHOT`. Also notice the use of the property reference `${company.name}`.

```
<?xml version="1.0" encoding="utf-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" amlns:xsi="http://www.w3org/2001/XMLSchema-ins
<modelVersion>4.0.0</modelVersion>
<groupId>com.adobe.training</groupId>
<packaging>pom</packaging>
<version>0.0.1-SNAPSHOT,/version>
<properties> [7 lines]
<name>${company.name} - Parent</name>
<description>The Adobe Training Project (Parent and Reactor) <description>
<build> [159 lines]
<dependencyManagement> [189 lines]
</project>
```

Further down the file, in the `<build>` section, notice the Configured Maven Plugins. In particular, look at the Apache Felix Maven Bundle Plugin:

```
<!-- Apache Felix Bundle Plugin -->
<plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</artifactId>
        <version>2.3.7</version>
        <inherited>true</inherited>
</plugin>
```

This plugin allows Maven to build a bundle .jar file containing the bundle meta data, which can be calculated as far as possible and can be specified in the pom. The bundle is installed into the Apache Felix OSGi, as we will see next.

Another important plugin is the Maven Service Component Runtime (SCR) plugin. OSGi components and services descriptor files are defined through annotations, and the plugin creates the necessary descriptors for the OSGi Declarative Services, Config Admin and Metatype services. The main sections are shown below and you can find the entry for this plugin in the pom.xml file:

```
<!-- Apache Felix SCR Plugin -->
<plugin>
<groupId>org.apache.felix</groupId>
<artifactId>maven-scr-plugin</artifactId>
<executions>
        <execution>
                <id>generate-scr-scrdescriptor</id>
                <goals> <goal>scr</goal> </goals>
        </execution>
</executions>
</plugin>
```

Take a look at these and other plugins in the pom.xml file:

```
<build>
    <plugins>
        <!-- Maven Release Plugin -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-release-plugin</artifactId>
            <version>2.1</version>
            <configuration>
                <scmCommentPrefix>[maven-scm] :</scmCommentPrefix>
                <preparationGoals>clean install</preparationGoals>
                <goals>install</goals>
                <releaseProfiles>release</releaseProfiles>
            </configuration>
        </plugin>
        <!-- Maven Source Plugin -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-source-plugin</artifactId>
            <version>2.0.4</version>
            <inherited>true</inherited>
        </plugin>
        <!-- Maven Resources Plugin -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-resources-plugin</artifactId>
            <configuration>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
        <!-- Maven Enforcer Plugin -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-enforcer-plugin</artifactId>
            <executions>
                <execution>
                    <id>enforce-maven</id>
                    <goals>
                        <goal>enforce</goal>
                    </goals>
                    <configuration>
                        <rules>
                            <requireMavenVersion>
                                <version>[2.2.1,)</version>
                            </requireMavenVersion>
                            <requireJavaVersion>
                                <message>
                                    Project must be compiled with Java 6
                                </message>
```

The <profiles> section includes the different Maven modules:

```
<profiles>
    <!-- Default Profile: Build all modules -->
    <profile>
        <id>default</id>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
        <modules>
            <module>${module.prefix}-core</module>
            <module>${module.prefix}-ui</module>
        </modules>
    </profile>
</profiles>
```

And in the <dependencyManagement> section, the <dependencies> including their versions and scope:

Project-specific plugins can be defined in the pom.xml present in a specific project. Open the pom.xml file present in the company-ui folder. The content-package-maven-plugin is an important plugin. This plugin builds a content package that is already defined in CRX.

```
<plugin>
    <groupId>com.day.jcr.vault</groupId>
    <artifactId>content-package-maven-plugin</artifactId>
    <version>0.0.18</version>
    <extensions>true</extensions>
    <configuration>
<!-- parameters and values common to all goals, as required -->
    </configuration>
</plugin>
```

```
    <dependencies>
        <!-- Project Dependencies -->
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>${module.prefix}-core</artifactId>
            <version>${project.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>${module.prefix}-components</artifactId>
            <version>${project.version}</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.felix</groupId>
            <artifactId>org.apache.felix.scr</artifactId>
            <version>1.6.1-R1236132</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.felix</groupId>
            <artifactId>org.apache.felix.scr.annotations</artifactId>
            <version>1.4.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.osgi</groupId>
            <artifactId>osgi_R4_core</artifactId>
            <version>1.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.osgi</groupId>
            <artifactId>osgi_R4_compendium</artifactId>
            <version>1.0</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.4.3</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>1.5.2</version>
            <scope>provided</scope>
        </dependency>

        <!-- Testing -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <scope>test</scope>
            <version>4.8.2</version>
        </dependency>
        <dependency>
            <groupId>org.easymock</groupId>
            <artifactId>easymock</artifactId>
            <version>3.0</version>
            <scope>test</scope>
        </dependency>
```

For example, check that the dependency org.apache.felix.scr is already in the JCR repository. Notice that it is declared as provided in the pom:

4. Now compile the project. Open a command prompt window and change to the directory containing the parent pom.xml file:

`C:\Adobe\AEM60AdvDev\sampleproject\sampleproject` on Windows
`/Adobe/AEM60AdvDev/sampleproject` on Mac / Unix

then type:

```
mvn clean install
```

N.B. On a Mac, you may need to use the SUDO command depending on the permissions settings on this directory, e.g.

```
sudo mvn clean install
```

...and then enter the machine password to complete the command when prompted. This command tells Maven to install the Maven artifacts into your ***local Maven repository***. Notice how maven goes to the default repository http://repo1.maven.org/maven2 to download various .pom and .jar files.

A directory will be created under your local user directory, called .m2. On windows XP this will be similar to C:\Documents and Settings\<username>\.m2. On Windows Vista and Windows 7 it will be `C:\Users\<username>\.m2` and on the Mac it is `/users/<username>/.m2`. Check under your user directory for the directory `~/.m2/repository/com/adobe/training` on your local machine. This local repository area is created in your home directory and is the location that all downloaded binaries and the projects you built are stored.

NOTE: You will get the following Build Error: Failed to resolve artifact. The error occurs because we have not yet told Maven where its central repository is.

---

5. Now we need to configure Adobe's public mvn repository. Copy the file `settings.xml` from your USB stick into the local repository directory `~/.m2/` and examine these entries in the `<repositories>` section:

```
<repository>
        <id>central</id>
        <name>Adobe Proxy Repository</name>
        <url>http://repo.adobe.com/nexus/content/groups/public</url>
</repository>
```

and in the <pluginRepositories> section:

```
<repository>
        <id>central</id>
        <name>Adobe Proxy Repository</name>
        <url>http://repo.adobe.com/nexus/content/groups/public</url>
</repository>
</pluginRepository>
```

Notice that we are declaring a new Proxy Repository for Maven, which allows us to use artifacts which are not available in the Central Maven repository. Maven will check all repositories listed in this way when attempting to download artifacts and plugins.

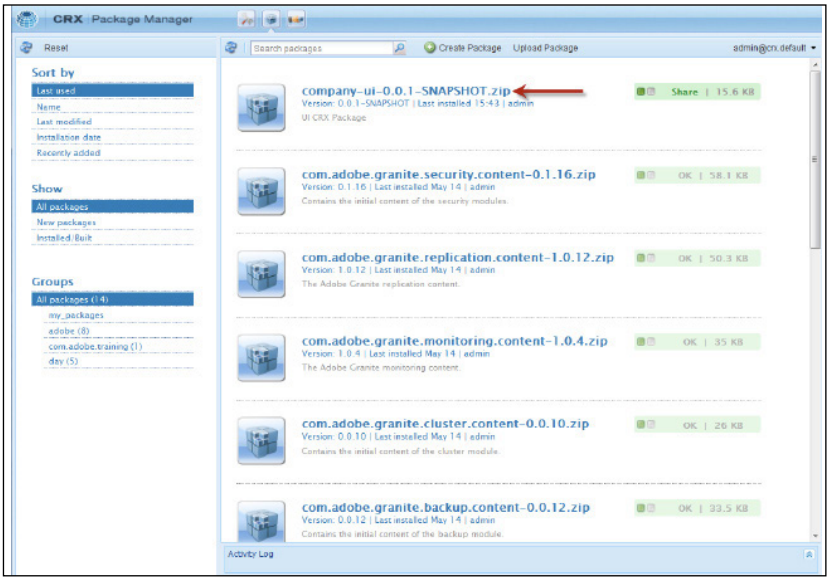
6. Start the AEM instance on port 4502.

7. Compile the project and deploy to AEM; on the command line run `mvn clean install -P full`

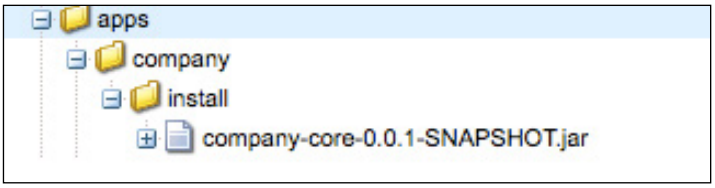
The -P option is for activating build profiles, so here we are using a profile called full. This builds the OSGi bundle and creates a CRX Package that gets uploaded to the local CRX instance. You will find the profile definition in the company-ui pom. xml. Find the package in:

```
company-ui/target/company-ui-0.0.1-SNAPSHOT.zip
```

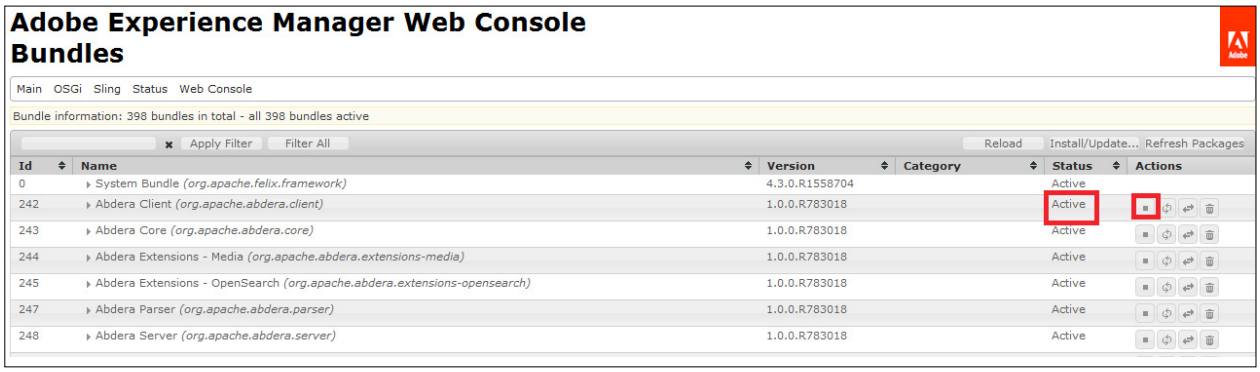
Open the PackageManager http://localhost:4502/crx/packmgr/index.jsp



In CRXDE Lite you should see:



This bundle is now installed, check in AEM Web Console.



Congratulations! - you have successfully compiled a project to an OSGi bundle and deployed it to CQ using Maven. We are now going to introduce vlt to make changes both locally and in the repository.

VLT

The FileVault tool (VLT) was created by Day Software (now Adobe Systems) specifically for use with CRX, and is provided in a standard CQ installation in the directory `crx-quickstart/opt/filevault`. VLT has a client side code set that issues HTTP commands to the JCR and a server side piece that outputs to the file system.

It lets you push and pull the content of a CRX or CQ system to and from your local file system, in a manner similar to using a source code control system client like Subversion (SVN) to push and pull data to and from a code repository.

The VLT tool has functions similar to those of a source control system client, providing normal check-in, check-out and management operations, as well as configuration options for flexible representation of the content on your local file system. You run the VLT tool from the command line.

Mapping CRX/CQ Content to the File System

The default scheme used by VLT to map the CRX/CQ repository content to the file system is as follows:

- Nodes of type `nt:file` are rendered as files in the file system, with the data from the jcr:content subnode transparently rendered as the contents of the file.
- Nodes of type `nt:folder` are rendered as directories in the file system.
- Nodes of any other types are rendered as directories in the file system with their set of properties and values recorded in a special file called `.content.xml` within the node's directory.

VLT and SVN

The usual way to use VLT is in conjunction with an SVN repository (or similar source code control system). The central SVN repository is used to hold a common copy of the full content of a CRX instance, in the file/folder format defined by the VLT mapping. This content tree can be pulled by a developer (using svn checkout) into his or her local file system, just as one would with any other SVN-managed codebase. The developer can alter the content and, to test it, use VLT to push those changes to his local CRX instance (using vlt checkin). Once the developer is satisfied with the changes, he or she can push them to the central SVN repository (using svn checkin) to share with other developers.

EXERCISE 1.2 - Installing the VLT Tool

1. The VLT tool is found in the directory `<cq installation dir>/crx-quickstart/opt/filevault`. There you will find two compressed files, `filevault.tgz` and `filevault.zip`. Copy the file which is most convenient to use on your system to a suitable directory (see below), then unpack it producing the directory `vault-cli-<version>`.

2. Under `vault-cli-<version>/bin` you will find the executables `vlt` and `vlt.bat` (the former for Unix, the latter for Windows).

The directory `vault-cli-<version>` can be left in place or moved to another location on your system. In either case, make sure that you include the full path to the `vault-cli-<version>/bin` directory in your system path. In windows, you can use the command:

NOTE: Additional information on installing and using vlt is available from http:// dev.day.com/docs/en/crx/ current/how_to/how_to_ use_the_vlttool.html.

```
SET PATH=%PATH%;<Path to filevault bin folder>
```

You can also add this to the PATH environment variable using the Control Panel. On a Mac, use:

```
export PATH=<Path to filevault bin folder>:${PATH}
```

3. You can test whether the tool is correctly installed by typing

   ```
   vlt --version
   ```

   at a command line.

EXERCISE 1.3 - Use VLT to change content from the command line

1. Make sure that vlt is on your $PATH

2. cd to `company-ui/src/main/content/jcr_root` and execute

vlt --credentials admin:admin co http://localhost:4502/crx/ **. --force**

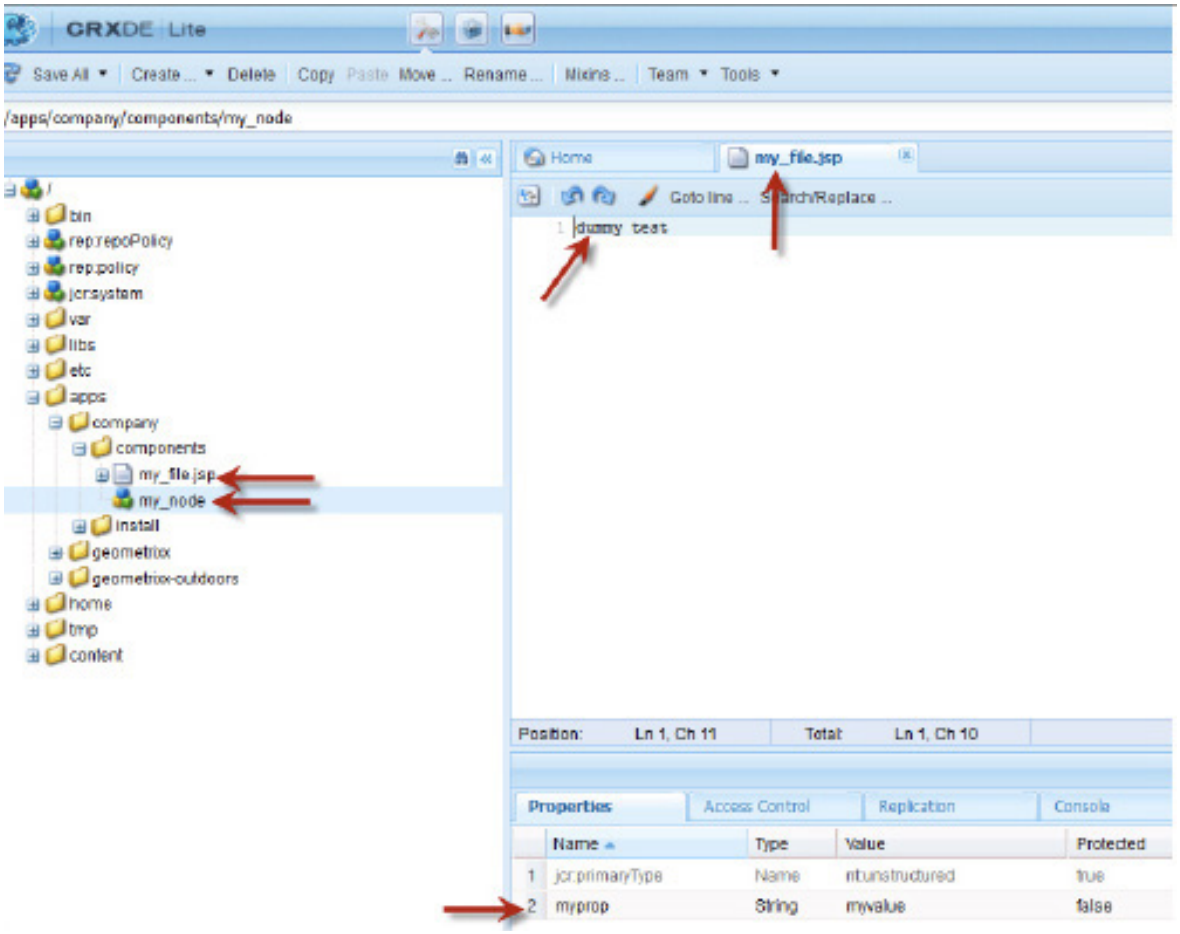
This will perform a vlt checkout of the JCR repository to your local filesystem. You will now have file serializations of the JCR content, e.g. in company-ui/src/main/content/jcr_root/.content.xml

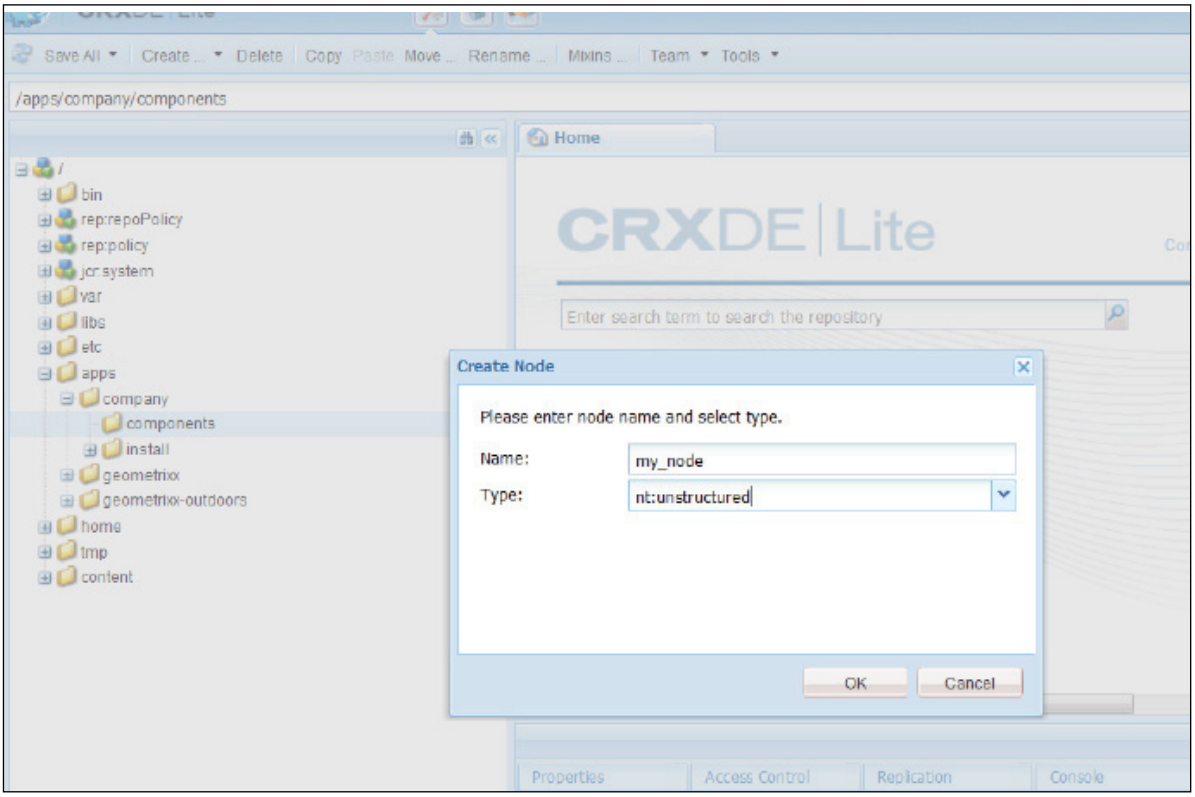
The JCR paths that do get checked out are configured in `company-ui/src/main/content/META-INF/vault/filter.xml`. Inspect this file. As an alternative to using this folder, you can specify the filter file in the vlt command line with `--filter filter.xml`, for example:

vlt co --filter filter.xml http://localhost:4502/crx/ .

NOTE: The credentials have to be specified only once upon your initial checkout. They will then be stored in your home directory inside `~/.vault/auth.xml`.

3. In CRXDE Lite create a node, of type sling:Folder, named **/apps/company/components**. Save. Inside the folder add a node of type nt:unstructured and a file, as suggested in the screenshot below. Add a property to the node as suggested in the screenshot below.





On the command line cd to `company-ui/src/main/content/jcr _ root` and execute

vlt up (vlt up is short for vlt update) You should see (with paths abbreviated):

Connecting via JCR remoting to http://localhost:4502/crx/server
A apps/company/components
A apps/company/components/.content.xml (text/xml)
A apps/company/components/my_node
A apps/company/components/my_node/.content.xml (text/xml)
A apps/company/components/my_file.jsp (text/plain)

4. Inspect apps/company/components/my_node/.content.xml. In your sampleproject directory, cd to sampleproject/company-ui/src/main/content/jcr_root/apps/company/components/my_node. Open .content.xml and add an XML property:

```
<?xml version="1.0" encoding="UTF-8"?>
<jcr:root xmlns:jcr="http://www.jcp.org/jcr/1.0" xmlns:nt="http://www.jcp.org/
jcr/nt/1.0"
jcr:primaryType="nt:unstructured"
myprop="myvalue"
myotherproperty="some value"
/>
```

Commit to CRX by executing:

vlt ci apps/company/components/my_node/.content.xml

You should see:

```
Connecting via JCR remoting to http://localhost:4502/crx/
server
Collecting commit information...
sending.... apps/company/components/my _ node/.content.xml
Transmitting file data...
U apps/company/components/my _ node/.content.xml
done.
```

Verify the change in CRXDE Lite

5. Change the content of apps/company/components/my_file.jsp and commit to CRX not via vlt, but by building the package. Change directory to the project root directory and type:

   ```
   mvn clean install -P full
   ```

   Verify the change in CRXDE Lite.

Congratulations - you have installed vlt and used it to checkout files from, and commit some changes to the CQ repository.

EXERCISE 1.4 - Installing Eclipse

At this point we will start to use Eclipse for working with our project, so if it is not already installed, you will need to install Eclipse.

Eclipse is open source software used to edit the project source locally on your file system. In this section, you will install Eclipse and a Maven plugin which embeds the Maven functionality within Eclipse:

1. Suitable versions of Eclipse for Mac and Windows are provided on the USB memory stick in the /Distribution/Eclipse folder. You can also download Eclipse from http://www.eclipse.org/downloads - select the Eclipse IDE for Java EE Developers option and the version suitable for your operating system. However for consistency with the course materials, please use the version of eclipse provided for this course if possible.

2. Install Eclipse; extract from the downloaded zip file to your destination director

3. Start Eclipse:

Navigate to the directory into which you extracted the contents of the Eclipse installation zip file. For example `C:\Program Files\Eclipse\.` On Windows, or Applications/eclipse on the Mac. You may need to use the sudo command on the Mac to do this. Double-click **eclipse. exe** (or **eclipse.app)** to start Eclipse.

4. Click on the Workbench logo in the top right-hand corner to close the Welcome screen and show the main workbench.

5. Generate Eclipse project files, from the project root execute:

```
mvn eclipse:eclipse -DdownloadSources=true
                    -DdownloadJavadocs=true
```

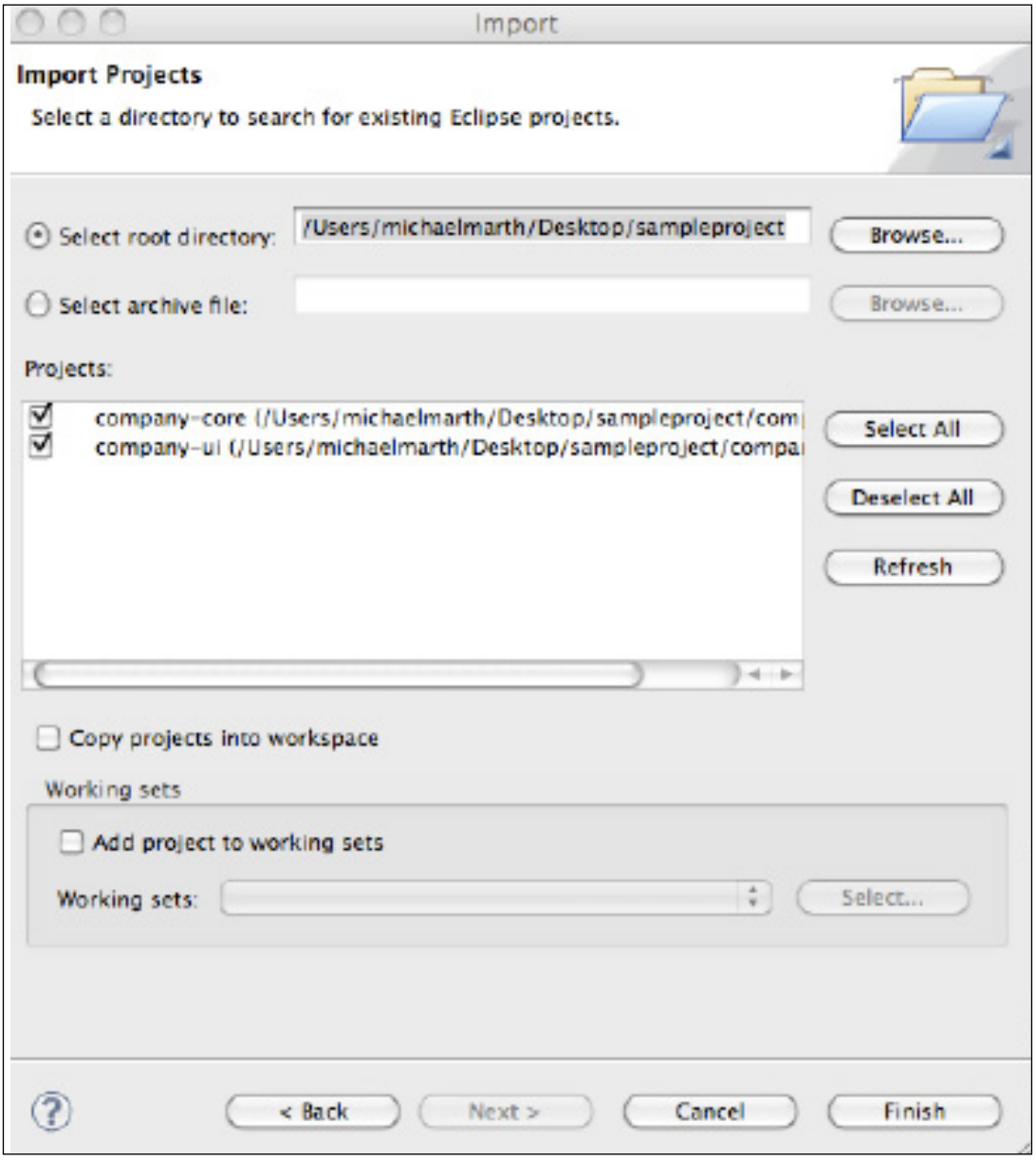
This will generate 2 files with extensions .project which can be imported into Eclipse:

```
sampleproject/company-core/.project
sampleproject/company-ui/.project
```

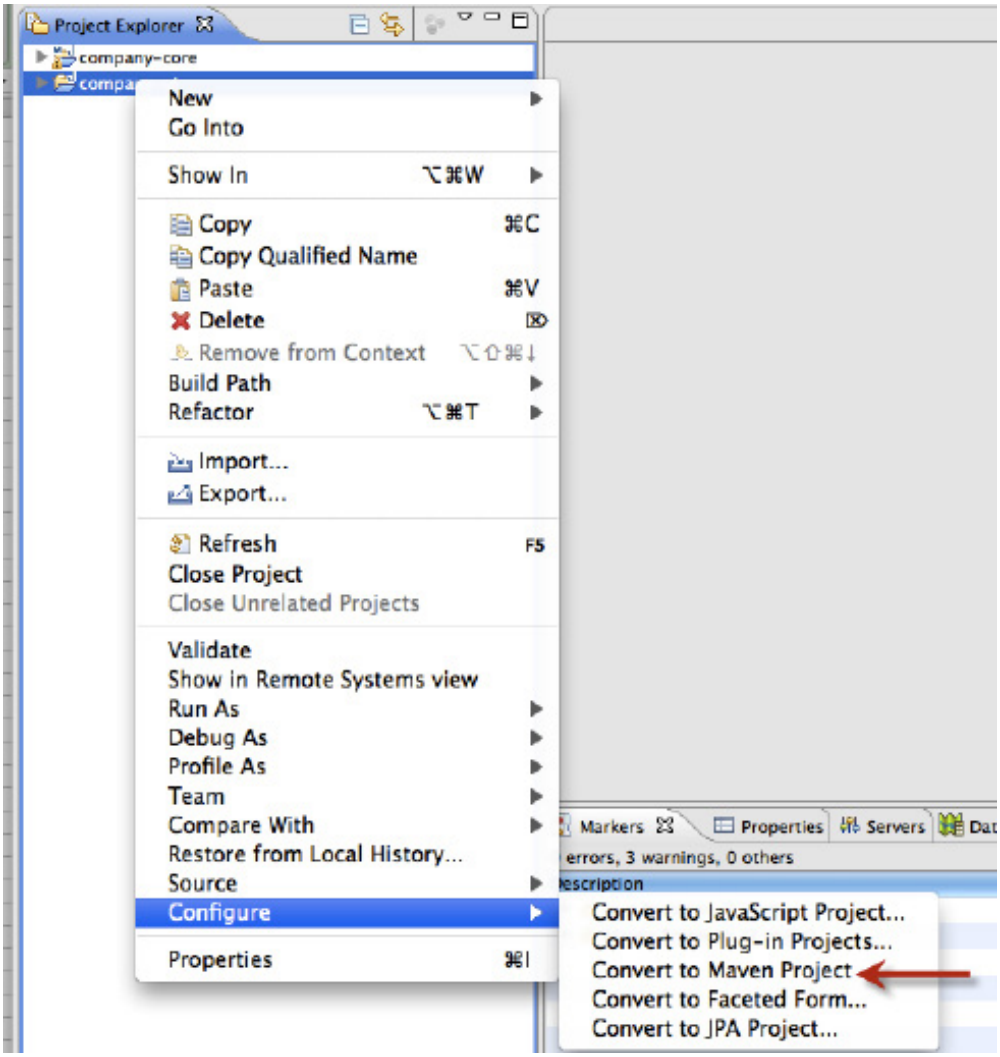
Import these into Eclipse using the menu option:

```
File > Import > General > Existing Projects Into Workspace
```

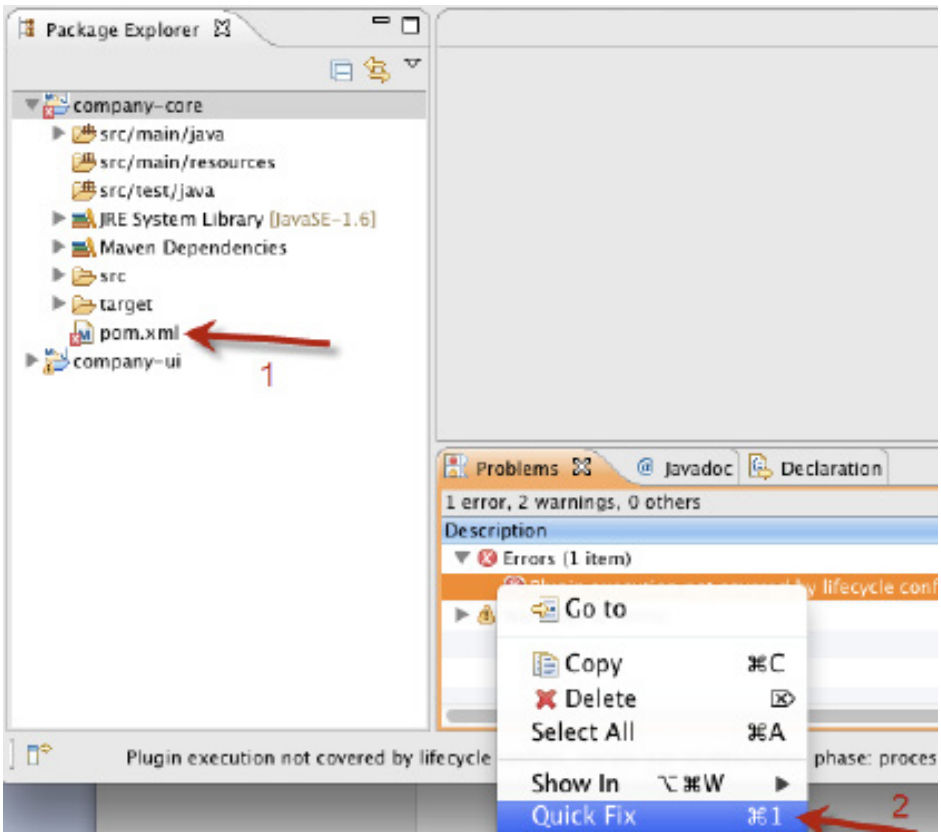
Click the Browse button and navigate to the sampleproject directory. Eclipse will find the two project files in this directory tree.



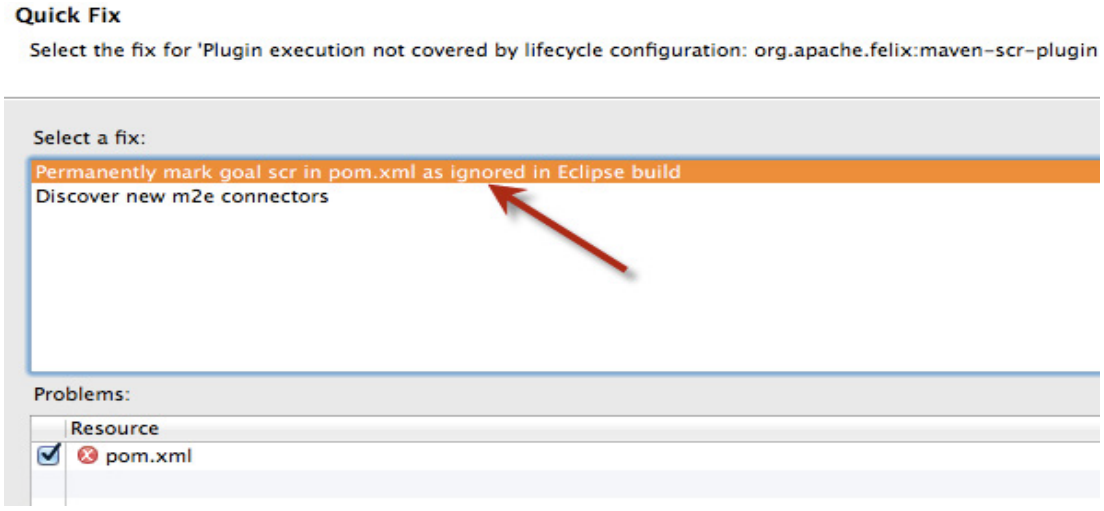
6. **Right-click** on both projects (one by one) and select Configure > Convert to Maven Project



7. **If you see an error in the problem panel, right click** on the error and select "Quick Fix"



8. Make sure the first option "Permanently mark goal scr in pom.xml as ignored in Eclipse build" is selected, and click **Finish**



Congratulations! - you have installed Eclipse and included the M2e plugin. We will use this tool throughout the rest of the course.

NOTE: It may happen that you will encounter the following error while updating your project:
" Unsupported IClasspathEntry kind=4" This can happen because the m2e plugin doesn't support correctly classpathentry with the kind attribute set to var (resulting from mvn eclipse:eclipse)
To fix this please follow these steps:

1. Make sure m2e plugin is installed
2. Disable maven nature of the project (right-click menu)
3. run mvn eclipse:clean while project is open
4. re enable maven nature

02

OSGi Components, Annotations

OSGi

The Open Services Gateway Initiative (OSGi), also known as the Dynamic Module for Java, defines an architecture for modular application development: a component-oriented framework.

OSGi Alliance

The OSGi Alliance is an industry consortium – including Adobe. Several expert groups within the consortium define the specifications.
OSGi was initially made popular by the Eclipse community, that needed a way to support dynamic plug-ins. There are multiple popular implementations of the OSGi specification: Equinox OSGi, Apache Felix, and Knopflerfish OSGi.

OSGi Overview

The OSGi specification defines two things: a set of services that an OSGi container must implement and a contract between the container and your application.

To get around the problems resulting from the global, flat classpath in traditional Java, OSGi takes a completely different approach: each module/bundle has its own class loader, separate from the class loader of all other modules. The class space is managed by OSGi.

OSGi specifies exactly how classes can be shared across modules, using a mechanism of declaring explicit imports and exports. In OSGi, only packages that are explicitly exported from a bundle can be shared with (imported and used in) another bundle. Developing on the OSGi platform means first building your application using OSGi APIs, then deploying it in an OSGi container. The OSGi specification allows for a dynamic, managed life cycle. From a developer's perspective, OSGi offers the following advantages:

- You can install, uninstall, start, and stop different modules of your application dynamically without restarting the container.
- Your application can have more than one version of a particular module running at the same time.
- OSGi provides very good infrastructure for developing service-oriented applications, as well as embedded, mobile, and rich internet apps.

Multiple, modular applications and services run in a single VM.

An OSGi application is:

- A collection of bundles that interact via service interfaces
- Bundles may be independently developed and deployed
- Bundles and their associated services may appear or disappear at any time

The resulting application follows a Service-Oriented Component Model approach.

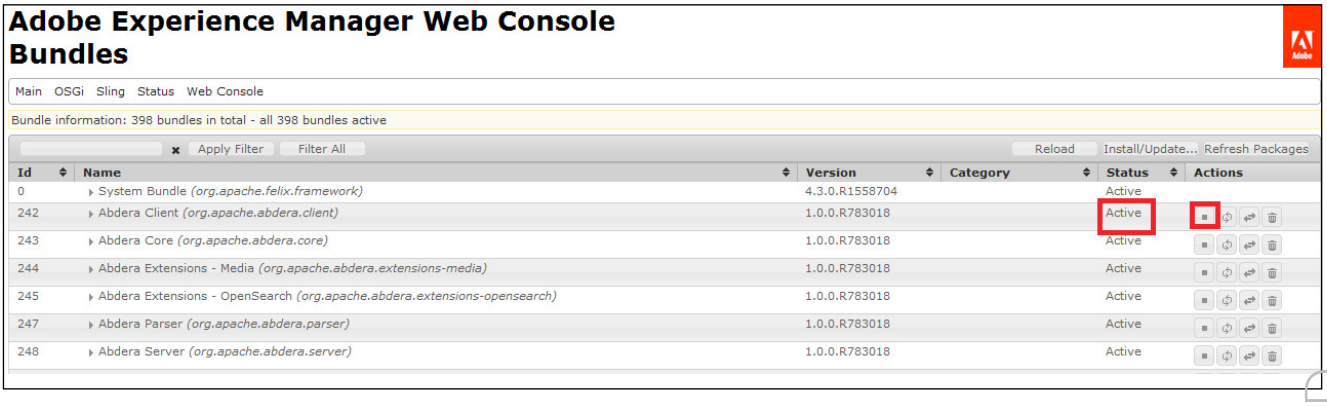
The AEM and the CRX applications are shipped and deployed as OSGi bundles. We use Apache Felix as our OSGi container.

Apache Felix

Apache Felix is an implementation of the OSGi R4 Service Platform. The OSGi specifications are ideally suited for projects based on the principles of modularity, component-orientation, and/or service-orientation. OSGi technology defines a dynamic service deployment framework that supports remote management.

Managing your OSGi applications

The Adobe AEM Web Console provides for the dynamic management and configuration of OSGi bundles. You can see the figure below the Bundles tab of the Adobe AEM Web Console. This console tab allows you to determine the status of each bundle in the container. You can also see the actions to which each bundle will respond.



OSGi Bundles, Services, Components

Bundles

Building on Java's existing standard way of packaging together classes and resources: the JAR file, an OSGi bundle is just a JAR file. Metadata is added to promote a JAR file into a bundle. The additional metadata is added to the manifest. The OSGi metadata is provided as additional header information in the META- INF/MANIFEST.MF file. The additional information consists of:

- Bundle Name(s):
  - a "symbolic" name used by OSGi to determine the bundle's unique identity
  - optional human-readable, descriptive name
- Bundle Version
- The list of services imported and exported by this bundle
- Optional, additional information, such as:
  - minimum Java version that the bundle requires
  - vendor of the bundle
  - copyright statement
  - contact address, etc.

Bundles in OSGi can be installed, updated and uninstalled without taking down the entire application. This makes modular application deployment possible, for example, upgrading parts of a server application — either to include new features or to fix bugs found in production — without affecting the running of other parts.

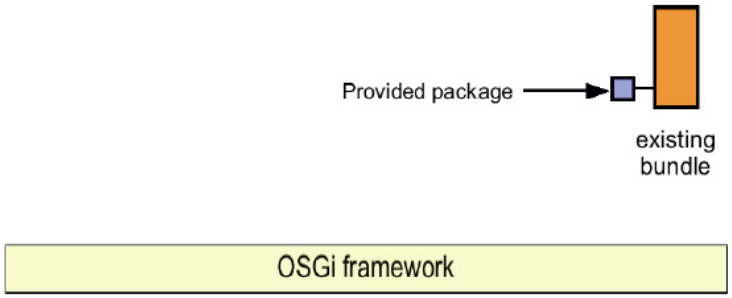
The bundles are loosely coupled.

- Package imports and exports with versions

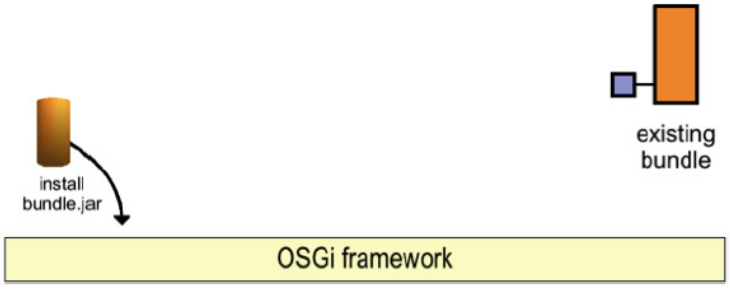
- Dependencies are independent from bundle organization
- "Someone" provides the self-describing package
- OSGi provides error management for unresolved bundles
- Requires modular thinking during application design
- Requires proper meta data and consistent version management

Dependency Management Resolution

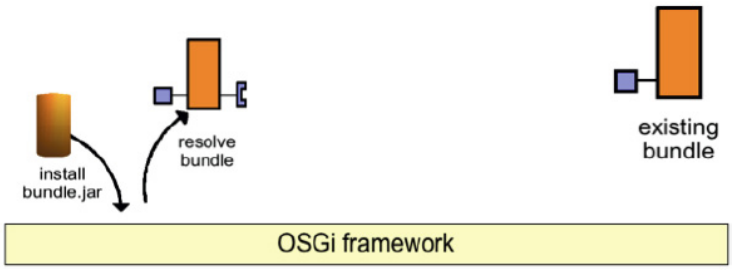
A bundle is present in the container:



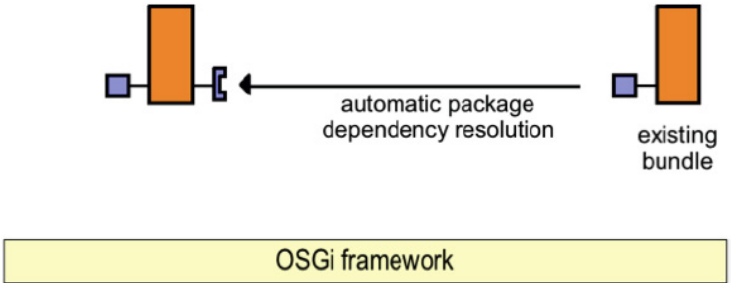
"Someone" provides a new bundle. The bundle is installed into the OSGi container:



The OSGi container resolves the new bundle:



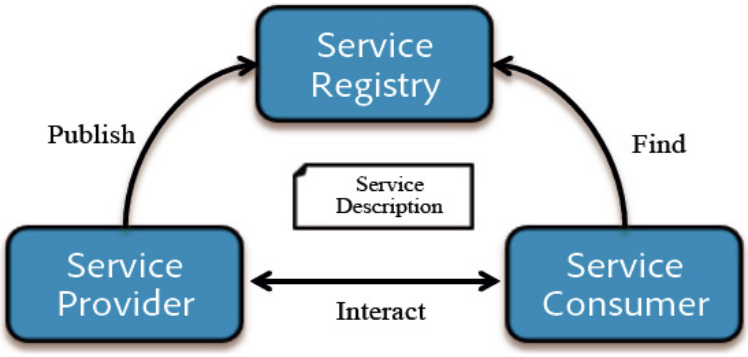
Automatic dependency resolution is provided by the container:



Service Registry Model

OSGi provides a service-oriented component model using a publish/find/bind mechanism. Using the OSGi declarative services, the consuming bundle says "I need X" and "X" gets injected.

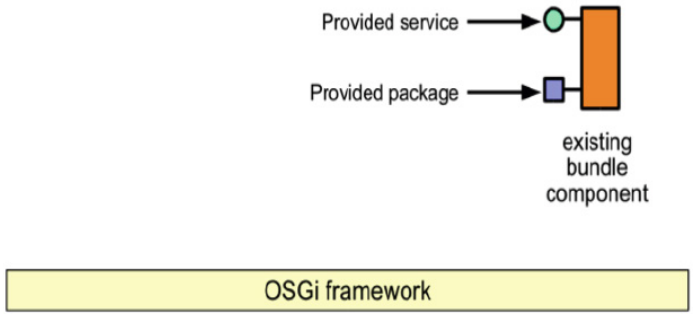
Because you cannot depend on any particular listener or a consumer being present in the container at any particular time, OSGi provides dynamic service look up using the Whiteboard registry pattern. Briefly defined, the Whiteboard registry pattern works as follows:



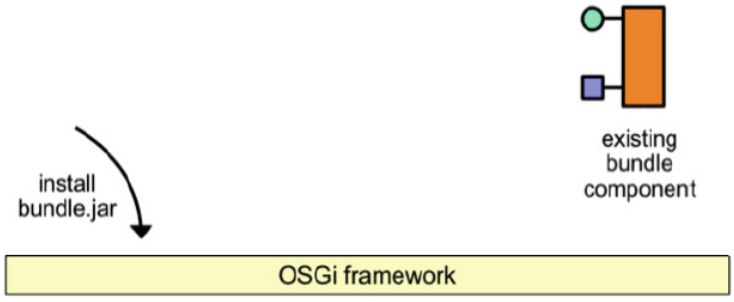
Instead of registering a listener/consumer object with the service provider, the consumer creates an object that implements OSGi listener interface, providing a method that should be called when the event of interest occurs. When the desired event occurs, the service provider requests a list of all of the services of type the object, and then calls the action method for each of those services. The burden of maintaining the relationships between service providers and service consumers is shifted to the OSGi framework. The advantage to doing this is that the OSGi framework is aware of bundle status and lifecycle and will unregister a bundle's services when the bundle stops.

Dynamic service lookup

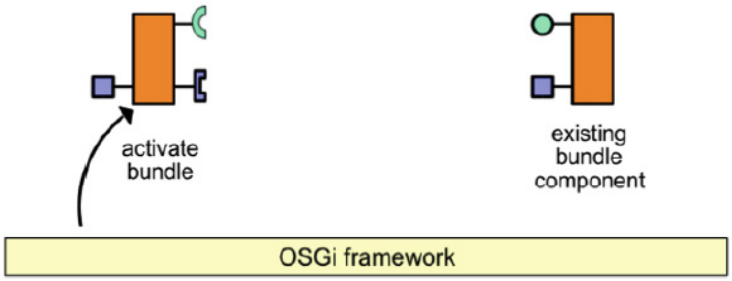
Step 1 - Existing service



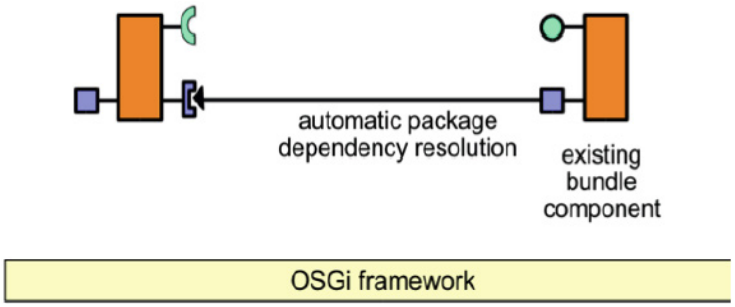
Step 2 - New bundle install



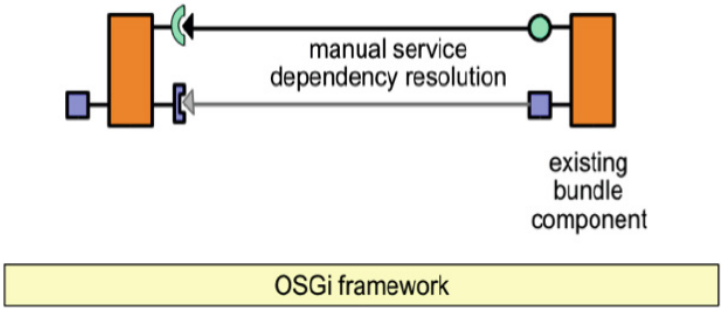
Step 3 - New bundle activation



Step 4 - Automatic package dependency resolution



Step 5 - Manual service dependency resolution



OSGi Service Advantages

In OSGi based systems, functionality is mainly provided through services. Services implement one or more interfaces, which define the type of service provided. It is the lifecycle of the bundle, which defines the lifecycle of the service: A service object may be instantiated when the bundle is started and will automatically be removed when the bundle is stopped.

The advantages of OSGi Services are as follows:

- Lightweight services
- Lookup is based on interface name
- Direct method invocation
- Good design practice
- Separates interface from implementation
- Enables reuse, substitutability, loose coupling, and late binding

Declarative Services

Usually, the functionality of a bundle is made available to the rest of the system, when the bundle is started.

The drawback of this method of service registration is that the services may have a dependency on other services and also have to take into account that services may come and go at any time. Though the implementation of this set of dependencies is straightforward as a ServiceListener, which listens for service registration and unregistration events, this is somewhat tedious and repetitive for each service using other services.

To overcome this situation, the OSGi framework provides a set of Declarative Services. The Declarative Services specification enables the declaration of services in configuration files, which are read by the Declarative Services Runtime to observe dependencies and activate (register) and deactivate (unregister) services depending on whether requirements/dependencies are met.

Additionally, the dependencies may be supplied through declared methods. The specification calls a class declared this way a ***component***. A component may or may not be a service registered with the service registry.

Components are declared using XML configuration files contained in the respective bundle and listed in the Service-Component bundle manifest header. These configuration files may be handwritten and registered.

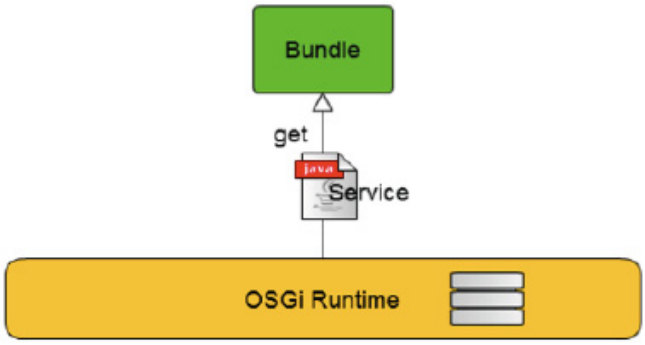
Declarative Services are a good alternative to:

- Writing an Activator
- Registering the bundle in the framework
- Using the service tracker

The OSGi Service Component (Declarative Services) reads the descriptions from started bundles. The descriptions are in the form of XML file(s) which define the set of components for a bundle. It is through the XML configuration definition information that the container:

- Registers the bundle's services
- Keeps track of dependencies among the bundles

- Starts/stops services
- Invokes the optional activation and deactivation method
- Provides access to bundle configuration



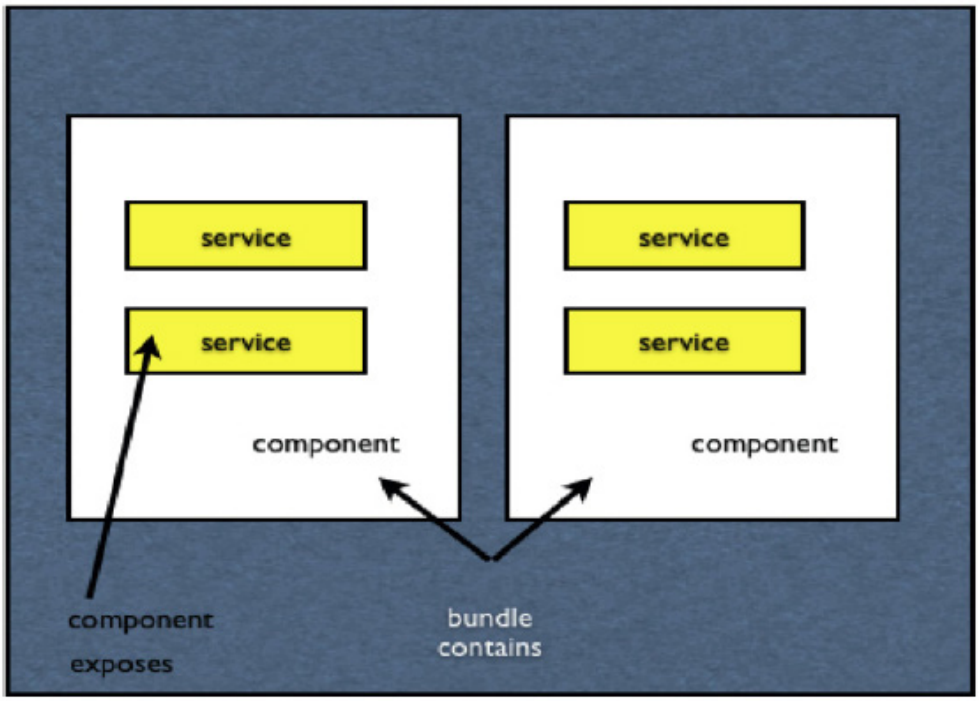
Components

Components are the main building blocks for OSGi applications. Components in OSGi are, by definition, provided by a bundle. A bundle will contain/provide one or more components.

A component is like a run-time service. They can publish themselves as a service, and/or they can have dependencies on other components/services. These dependencies will influence their life cycle as the component will only be activated by the OSGi container when all required dependencies are met/available. Each component has an implementation class, and can optionally implement a public interface, effectively providing this "service".

NOTE:

- A Service is by default only started if someone else uses it. The immediate flag forces a service start
- The OSGi Runtime calls the bundle and not the other way around
- Do not confuse OSGi components with AEM components!



Best Practice: Always upload the bundle using the JCR. That way the release engineers and system administraters have 1 common mechanism for managing bundles and configurations.

A component is:

- Piece of software managed by an OSGi container
- Java object created and managed by a container

The OSGi container manages component configuration and the list of consumed services.

- A component can provide a service
- Component can implement one or more Java interfaces as services

A service can be consumed/used by components and other services. Basically a bundle needs four things to become a component:

- XML file where you describe the service the bundle provides and the dependencies of the other services of the OSGI framework.

- Manifest file header entry to declare that the bundle behaves as a component.
- Activate and Deactivate methods in the implementation class. (Or **bind** and **unbind** methods)
- Service component runtime. A service of the OSGI framework to manage the components: Declarative service of Equinox or Apache Felix SCR .

Bundles

Bundles can be uploaded into the Felix container through the Adobe AEM Web console or through the placement of the bundle into a folder, named "install", in the JCR. Use of the JCR allows easy maintenance of the bundle throughout its lifecycle. For example, deletion of the bundle from the JCR causes a delete of the Felix bundle by Sling. Subsequent replacement of the jar with a new version in the JCR will cause the new version of the bundle in the felix container.

Annotations

Service components can be annotated using the annotations provided by the *org.apache.felix.dependencymanager.annotation* bundle.

The following annotations are supported:

- Component
- Activate
- Deactivate
- Modified
- Service
- Property
- Reference

@Component

To register your components without the Annotations, you would have to implement Activators, which extend the DependencyActivatorBase class. The **@Component** annotation allows the OSGi Declarative Services to register your component for you. The **@Component** annotation is the only required annotation.

If this annotation is not declared for a Java class, the class is not declared as a component. This annotation is used to declare the `<component>` element of the component declaration. The required `<implementation>` element is automatically generated with the fully qualified name of the class containing the Component annotation.

```
package com.adobe.osgitraining.impl;
import org.apache.felix.scr.annotations.Component;
@Component
public class MyComponent {
}
```

@Activate, @Deactivate, and @Modified

The OSGi Declarative Service allows you to specify the name for the activate, deactivate and modified methods. Please note, in the code below, the annotations for the **@Activate** and **@Deactivate** action annotations. These actions specify what happens on activation and deactivation of the component.

```
package com.adobe.osgitraining.impl;
import org.apache.felix.scr.annotaions.Activate;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Deactivate;
@Component
public class MyComponent {
        @Activate
        protected void activate() {
          // do something
        }
        @Deactivate
        protected void deactivate() {          // do something
        }
}
```

NOTE: See section 112.4.6, Service Elements, in the OSGi Service Platform Service Compendium Specification or more information.

@Service

The **@Service** annotation defines whether and which service interfaces are provided by the component. This is a class annotation.

```
package com.adobe.osgitraining.impl;
import org.apache.felix.scr.annotaions.Component;
import org.apache.felix.scr.annotaions.Service;
import org.osgi.service.event.EventHandler
@Component
@Service(value=EventHandler.class)
public class MyComponent implements EventHandler {
```

NOTE: See section 112.4.7, Reference Element, in the OSGi Service Platform Service Compendium Specification for more information.

The **@Service** tag is used to declare the `<service>` and `<provide>` elements of the component declaration.

@Reference

The **@Reference** annotation defines references to other services. These other services (consumed services) are made available to the component by the Service Component Runtime. This annotation is used to declare the `<reference>` elements of the component declaration.

The **@Reference** annotation may be declared on a Class level or on any Java field to which it might apply. Depending on where the annotation is declared, the parameters may have different default values.

@Property

The **@Property** annotation defines properties which are made available to the component through the `ComponentContext.getProperties()` method. These tags are not strictly required but may be used by components to define initial configuration. This tag is used to declare `<property>` elements of the component declaration. This tag may also be defined in the Java Class comment of the component or in a comment to a field defining a constant with the name of the property.

```
@Component
@Service(value-EventHandler.class)
@Properties({
        @Property(name="event.topics", value="*", propertyPrivate=true),
        @Property(name="event.filter", value="(event.distribute=*)",
                propertyPrivate-true)
})
public class DistributeEventHandler
        implements EventHandler {
        protected static final int DEFAULT CLEANUP PERIOD = 15
        @Property(intValue=DEFAULT CLEANUP PERIOD)
        private static final String PROPERTY_CLEANUP_PERIOD
="cleanup.period";
        @Reference
        protected ThreadPool threadPool;
        @Activate
        protected void activate (final Map<String, Object> props){
                this.cleanupPeriod = toInt(props.get(PROP_CLEANUP_PERIOD));
        }
```

NOTE: See section 112.4.5, Property Elements, In the OSGi Service Platform Service Compendium Specification for more information.

Additionally, properties may be set, using this tag, to identify the component if it is registered as a service, for example the `service.description` and `service.vendor` properties.

Java Compiler Annotations

Please note that the following Annotations are defined by the Java compiler:

@Override

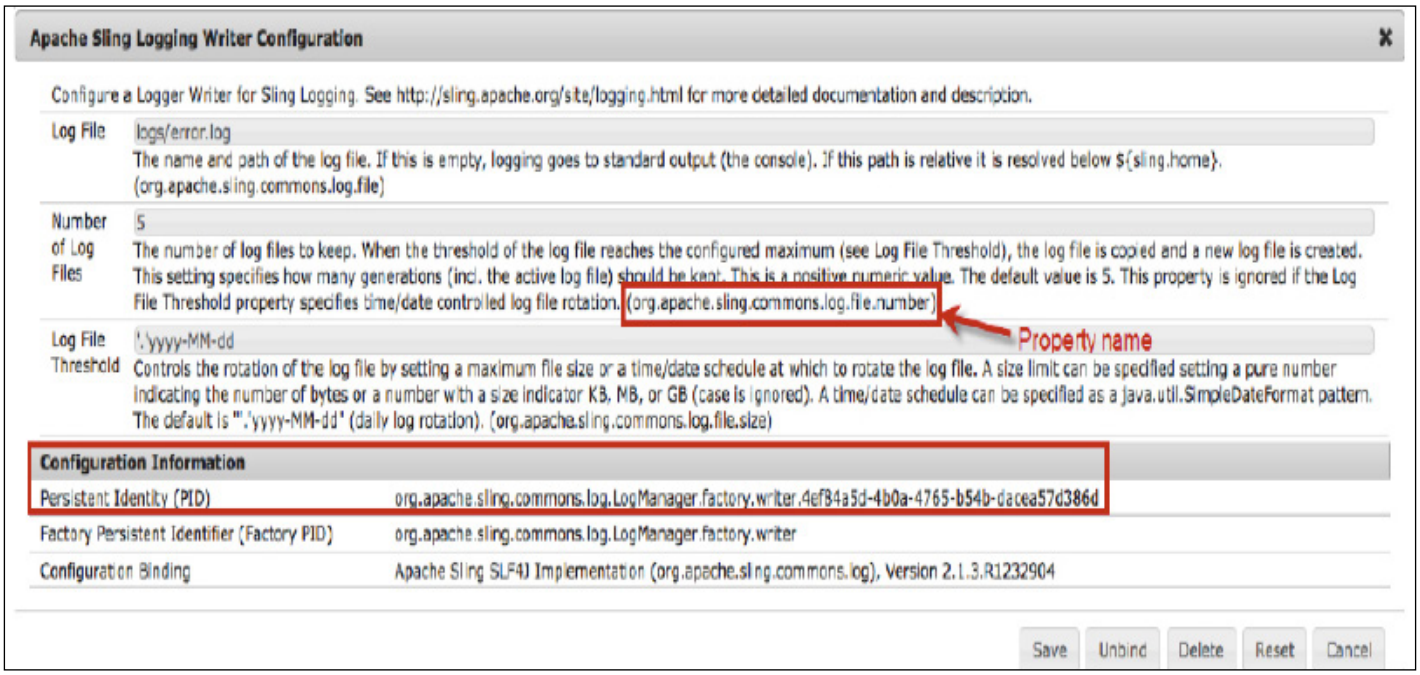
The @Override annotation informs the compiler that the element is meant to override an element declared in a superclass.

@SuppressWarnings

The @SuppressWarnings annotation tells the compiler to suppress specific warnings that it would otherwise generate.

Configurable Services

The OSGi Configuration Admin Service provides for storing configuration and for the delivery of the configuration automatically or on-demand to clients. Configuration objects are identified by so-called Persistent Identifiers (PID) and are bound to bundles when used. For Declarative Services, the name of the component is used as the PID to retrieve the configuration from the Configuration Admin Service.



The Configuration Admin Service not only allows components to get or retrieve configuration, it also

provides the entry point for Management Agents to retrieve and update configuration data. The combination of the configuration properties and meta type description for a given PID is used to build the user interface to configure the service and/or component.

OSGi Component

EXERCISE - 2.1

In this exercise we will:

- Create an OSGi service (interface and implementation)
- Log the (de)activation of the service
- Retrieve a reference to the repository and log the repository name
- Consume the service from a JSP

1. Go to the company-core project and locate the Activator.java.

2. Observe the way logging is added:

```
package com.adobe.training.core;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Activator implements BundleActivator {

  private static final Logger LOGGER = LoggerFactory.getLogger(Activator.class);

  /*
   * (non-Javadoc)
   * @see org.osgi.framework.BundleActivator#start(org.osgi.framework.BundleContext)
   */
  public void start(BundleContext context) throws Exception {
    LOGGER.info("bundle started");
  }

  /*
   * (non-Javadoc)
   * @see org.osgi.framework.BundleActivator#stop(org.osgi.framework.BundleContext)
   */
  public void stop(BundleContext context) throws Exception {
    LOGGER.info("bundle stopped");
  }

}}
```

3. Right-click the company-core project and create a new java class file named ***RepositoryService.java*** and add the following code.

```
package com.adobe.training.core;
public interface RepositoryService {
      public String getRepositoryName();
}
```

4. Implement the service interface. Create a folder named **impl** under the package directory.

5. In the **impl** folder that you just created, create a java class file named ***RepositoryServiceImpl.java***. Add the following code:

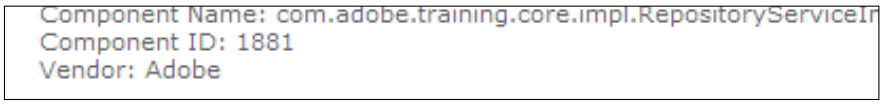
```
package com.adobe.training.core.impl;

import com.adobe.training.core.RepositoryService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Activate;
import org.apache.felix.scr.annotations.Deactivate;
import org.apache.felix.scr.annotations.Service;
import org.apache.felix.scr.annotations.Reference;
import javax.jcr.Repository;

@Component(immediate = true, metatype = true, label = "Hello Bundle")
@Service(value = RepositoryService.class)
public class RepositoryServiceImpl implements RepositoryService {

  private static final Logger LOGGER = LoggerFactory.getLogger(RepositoryServiceImpl.class);

  @Reference
  private Repository repository;

  public String getRepositoryName() {
    return repository.getDescriptor(Repository.REP_NAME_DESC);
  }

  @Activate
  protected void activate() {
    LOGGER.info("service activated" );
  }

  @Deactivate
  protected void deactivate() {
    LOGGER.info ("service deactivated");
  }
}
```

6. Notice the use of the **@Component, @Service and @Reference** annotations.

7. Deploy the bundle using mvn clean install -P bundle.

8. Go to http://localhost:4502/system/console/bundles and check that the company-core bundle.

9. Note that the RepositoryService has added in the bundle.



10. Go to the Component tab and see the component

11. Consume the service in a JSP by editing `content.jsp` in the geometrixx homepage component. (`/apps/geometrixx/components/homepage/content.jsp`)

```
<%@include file="/libs/foundation/global.jsp" %>
<div class="container_16">

 <%
      com.adobe.training.core.RepositoryService repositoryService=
      sling.getService(com.adobe.training.core.RepositoryService.class);
   %>
   Hello, my name is <%= repositoryService.getRepositoryName() %>

<div class="grid _ 16">
      <cq:include path="carousel" resourceType="foundation/components/carousel"/>
   </div>
   <div class="grid_12 body_container">
      <cq:include path="lead" resourceType="geometrixx/components/lead"/>
      <cq:include path="par" resourceType="foundation/components/parsys"/>
   </div>
   <div class="grid_4 right _ container">
      <cq:include path="rightpar" resourceType="foundation/components/parsys"/>
   </div>
   <div class="clear"></div>
</div>
```

12. Access the Geometrixx home page using the following URL:
    `http://localhost:4502/content/geometrixx/en.html`
    You should see the following on the Geometrixxx home page:

Congratulations! You have used Maven to create an OSGi service. Now you have a working knowledge of the annotations.

03

Sling, Resources, REST

Representational State Transfer (REST)

Roy Fielding coined the term in his doctoral thesis, REST is an architectural style, not a standard or implementation specification. Application data and state are represented as a set of addressable resources which present a uniform interface that allows transfers of state (e.g. reading and updating of the resource's state). The largest REST application is the World Wide Web itself, characterized by the use of HTTP for transport and URLs as addressing mechanisms. Systems that follow Roy Fielding's REST principles can be called REST-ful.

In the most general terms, REST outlines how to define and address resources. Resources are referred to individually with a universal resource identifier (URI), such as the URL used for Web addresses. Examples of resources might be: News entry, product, photo.

Since each request contains all relevant information, REST-ful systems are stateless. REST defines a restricted collection of interactions with resources: HTTP methods (GET,POST) for operations.

Advantages of REST

- Uses well documented, well established, well used technology and methodology
- Resource-centric rather than method-centric
- Given a URI anyone already knows how to access it

- Not another protocol on top of another protocol on top of another protocol on top of...
- Response payload can be of any format
- Uses the inherent HTTP security model
- Certain methods to certain URIs can easily be restricted by firewall configuration, unlike other XML over HTTP messaging formats

REST and Apache Sling

Being a REST framework, Sling is oriented around resources, which usually map into JCR nodes. Unlike traditional web applications that select a processing script, based on the URL, and then attempt to load data to render a result, Apache Sling request processing takes what, at first might seem like an inside-out approach to handling requests in that a request URL is first resolved to a resource, then based on the resource (and only the resource) it selects the actual servlet or script to handle the request.

With the GET method, there is a default behavior, so no extra parameters are necessary. Through URL decomposition, Apache Sling determines the resource URI and any other available information to be used in processing that resource. The POST method is handled by the Apache Sling PostServlet. The PostServlet enables writes to a datastore (usually the JCR) directly from HTML forms or from the command line, using CURL.

Again, just to remind you - Apache Sling is all about Resource-first request processing!

Apache Sling

Apache Sling is a web framework that uses a Java Content Repository, such as Apache Jackrabbit or Adobe CRX to store and manage content. Sling applications use either scripts or Java servlets, selected based on simple name conventions, to process HTTP requests in a RESTful way.

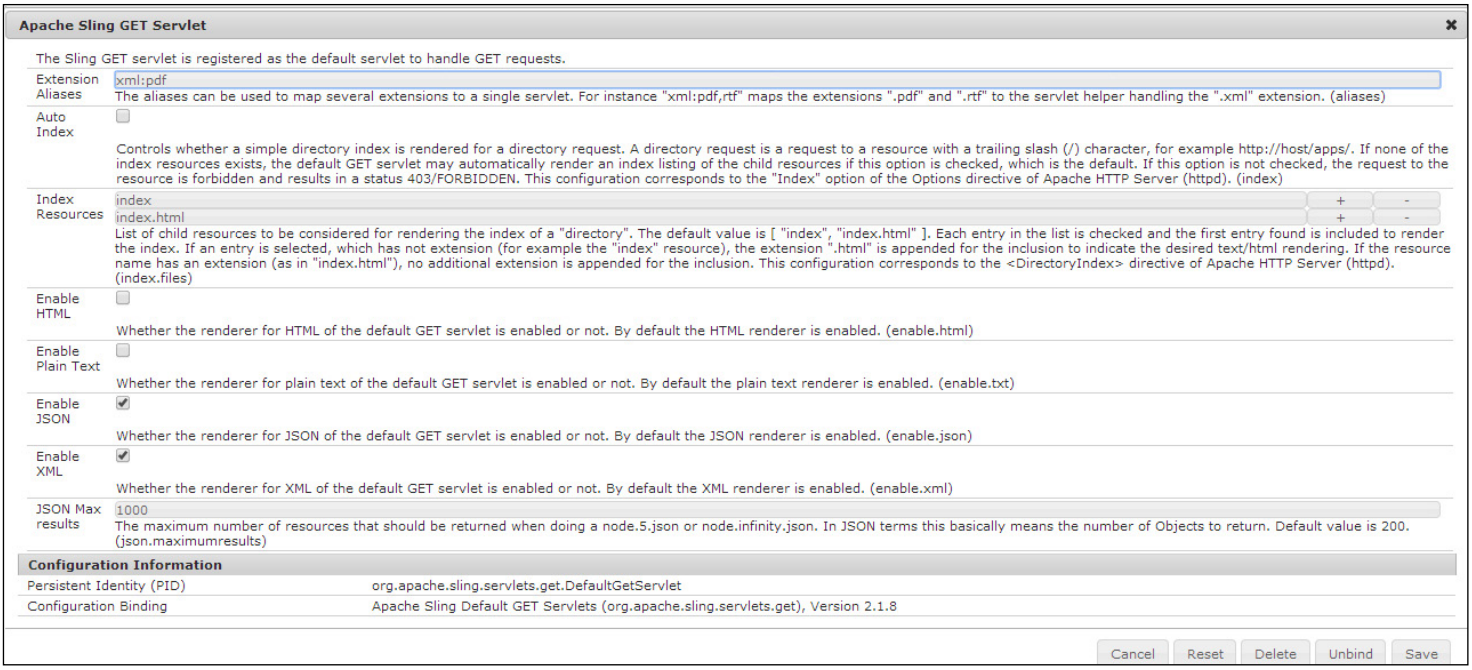
Default GET Servlet

For the default GET Servlet, Apache Sling provides default renderers for the following mime types:

- txt
- json

  e.g. mypage.3.json -> 3 levels deep
- docview.xml
- sysview.xml
- html

You can configure the Apache Sling GET Servlet using the Adobe AEM Web Console, similar to the figure below:



Sling POST Servlet

You have multiple options for modifying a JCR repository using Sling, two of which are WebDAV and the Sling default POST Servlet also called the SlingPostServlet.

To create content in the JCR, you simply send an HTTP POST request using the path of the node where you want to store the content and the actual content, sent as request parameters. So one possibility to do just that is by having an HTML Form similar to the following:

```
<form method="POST" action="http://host/mycontent"enctype="multipart-form/data">
        <input type="text" name="title" value="" />
        <input type="text" name="text" value="" />
</form>
```

The simple form shown above will set the title and text properties on a node at / mycontent. If this node does not exist it will be created, otherwise the existing content at that URI would be modified.

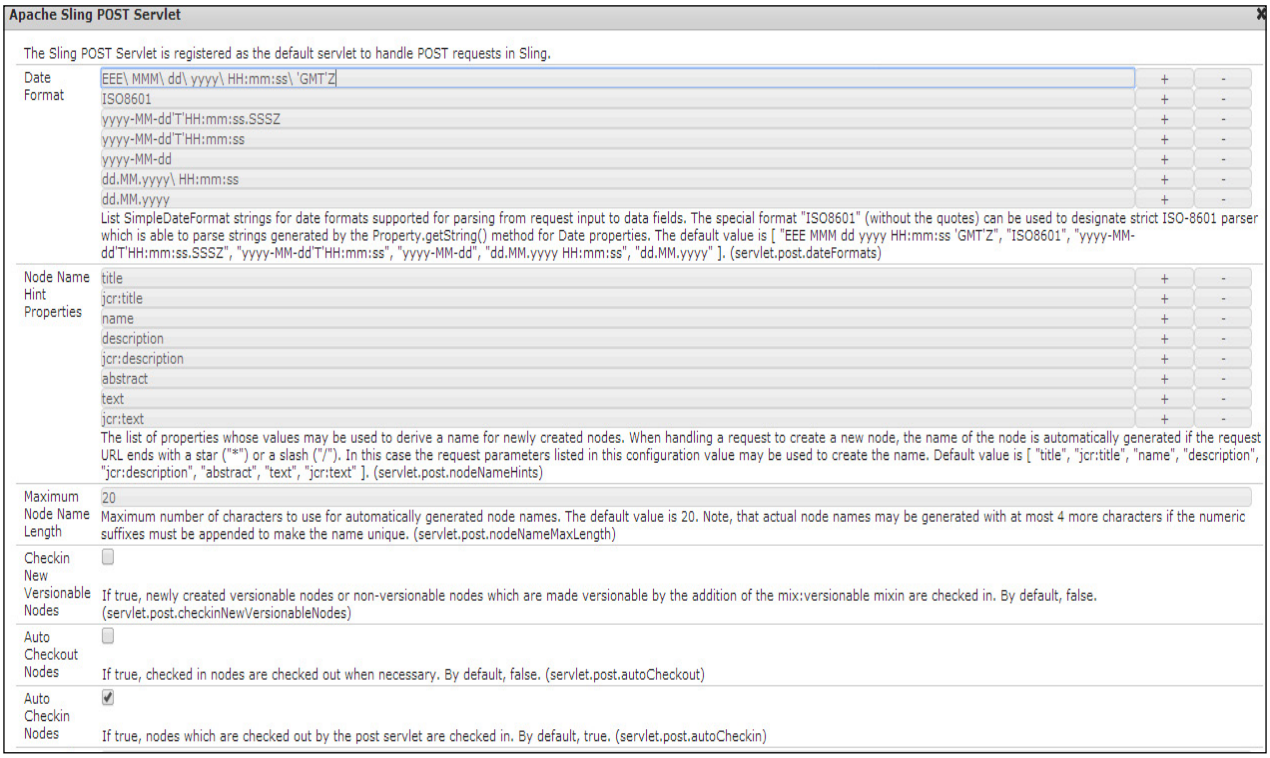
Similarly you can do this using the curl command line tool:

```
curl       -F"jcr:primaryType=nt:unstructured" -Ftitle="some titletext"

-Ftext="some body text content" http://host/some/new/content

curl       -F":operation=delete" http://host/content/sample
```

NOTE: If you try this command, you may need to authenticate the operation by adding the option:
-u admin:admin

See the following URL for more information on using the Sling POST Servlet
http://sling.apache.org/site/manipulating-content-the-slingpostservletservletspost.html

You can configure the Apache Sling POST Servlet using the Adobe AEM Web Console, similar to the figure below:



Sling and Resources

The *Resource* is one of the central concepts of Sling. Extending from JCR's "Everything is Content", Sling assumes "Everything is a Resource". Sling maintains a virtual tree of resources. This tree is a result of merging the actual content objects in the JCR Repository and resources provided by the so-called resource providers. By doing this Sling fits very well into the paradigm of the REST architecture. What is a resource?

- Sling's abstraction of the thing addressed by the request URI
- Usually mapped to a JCR node
- Could be mapped to a filesystem or a database

Properties of a resource

- Path - JCR Item path
- Type - JCR node type
- Metadata - e.g. last modification date

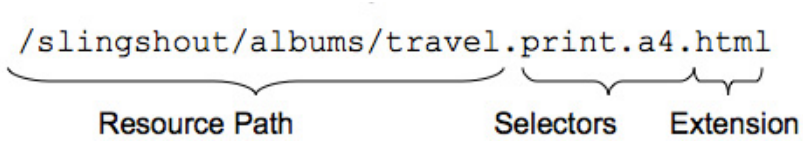
Resource-First Request Processing

Sling takes a unique approach to handling requests in that a request URL is first resolved to a resource, then based on the resource (and only the resource) it selects the actual servlet or script to handle the request.

Traditional Web Application frameworks employ methods to select a Servlet or Controller based on the request URL. That servlet or controller then tries to load some data (usually from a database) to act upon and finally to render the result somehow.

Sling turns this processing around in that it places the data to act upon at the center of the work and then uses the request URL to first resolve the data/content to process. This data/content is internally represented as an instance of the Resource interface. Based on this resource, the request method and more properties parsed from request URL, a script or servlet is then selected to handle the request.

Resource and representation



Apache Sling's mechanism for URL decomposition removes one or more data models. Previously the following were needed:

- URL structure
- Business objects
- DB schema

This is now reduced to:

- URL = resource = JCR structure

Basic Request Processing

Using Sling, the type of content to be rendered is not the first processing consideration. Instead the main consideration is whether the URL resolves to a content object for which a script can then be found to perform the rendering. This provides excellent support for Web content authors to build Pages which are easily customized to their requirements.

The advantages of this flexibility are apparent in applications with a wide range of different content elements, or when you need Pages that can be easily customized/viewed differently.

In Sling, and therefore also AEM, processing is driven by the URL of the HTTP request. This defines the content to be displayed by the appropriate scripts. To do this, information is extracted from the URL.

When the appropriate resource (content node) is located, the resource type is extracted from the resource properties. The sling:resourceType property is a path, which locates the script to be used for rendering the content.

Traditional web application frameworks usually begin servicing a request by selecting a procedure (servlet, script etc.) based on the request URI. This procedure then loads some data, usually from a database, and uses it to construct and send back a response. For example, such a framework might use a URI like this:

http://www.example.com/product.jsp?id=1234

This URL addresses the script to be used to render the result (product.jsp), not the resource that is being requested (product #1234).

Sling turns this around by placing the content at the center of the process. Each content item in the JCR repository is exposed as an HTTP resource, so the request URL addresses the data to be processed, not the procedure that does the processing. Once the content is determined, the script or servlet to be used to handle the request is determined in cascading resolution that checks, in order of priority:

- Properties of the content item itself.
- The HTTP method used to make the request.
- A simple naming convention within the URI that provides secondary information.

For example, a Sling application might use a URL like this:

http://www.example.com/cars/audi/s4.details.html

This URL would be decomposed as follows:

- cars/audi/s4 refers to the repository path to the content item (e.g., /content/cars/audi/s4).
- details.html refers to the script to be used to render this content (e.g. /apps/cars/details/html.jsp).

Because a Sling URL addresses content and not presentation logic, such a URL tends to be more meaningful and more stable over time.

The following steps represent the basic request processing steps:

1. URL decomposition
2. Resolve the resource (using URL)
3. Resolve rendering script
4. Source: resource type
5. Create rendering chain
6. Configurable (servlet) filters
7. Rendering servlet
8. Invoke rendering chain

Resource Resolver

The Sling Resource Resolver is the gateway for finding/accessing (resolving) resources. The Resource Resolver:

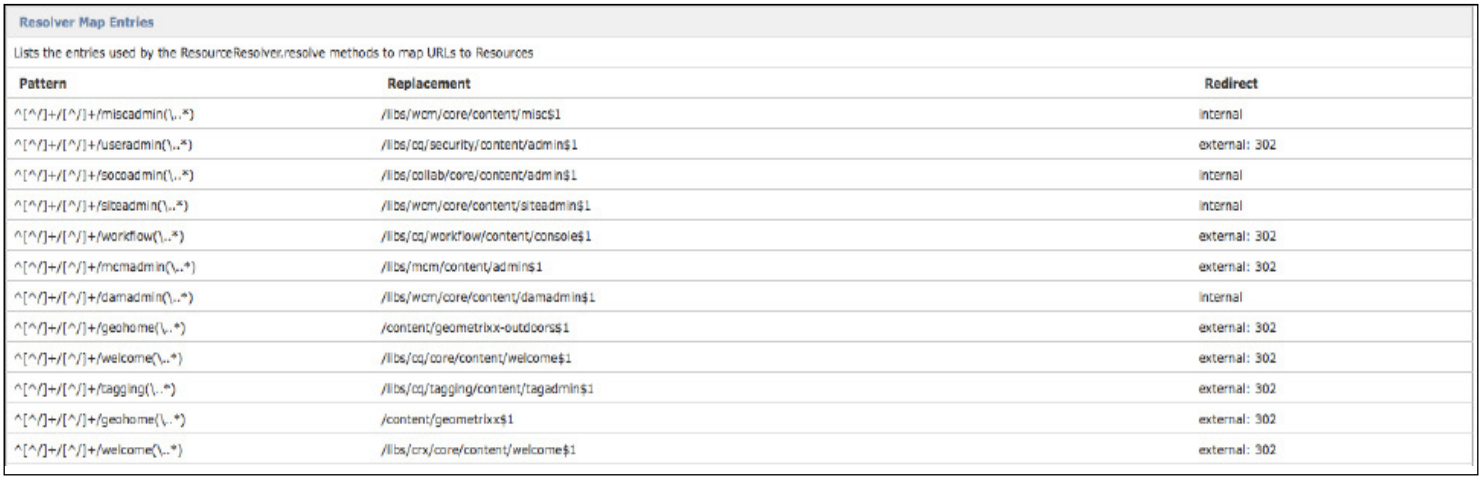
- abstracts the path resolution
- abstracts access to the persistence layer(s)

The Resource Resolver is configurable, through the AEM Web Console and also through standard OSGi configuration mechanisms in the CRX.

In AEM, resource mapping is used to define redirects, vanity URLs and virtual hosts.

You can examine the Resolver Map Entries from the Felix Console, as shown below, at:

```
/system/console/jcrresolver
```



Resolver Map Entries

Lists the entries used by the ResourceResolver.resolve methods to map URLs to Resources

Pattern	Replacement	Redirect
^[^/]+/[^/]+/miscadmin(\..*)	/libs/wcm/core/content/misc$1	internal
^[^/]+/[^/]+/useradmin(\..*)	/libs/cq/security/content/admin$1	external: 302
^[^/]+/[^/]+/socoadmin(\..*)	/libs/collab/core/content/admin$1	internal
^[^/]+/[^/]+/siteadmin(\..*)	/libs/wcm/core/content/siteadmin$1	internal
^[^/]+/[^/]+/workflow(\..*)	/libs/cq/workflow/content/console$1	external: 302
^[^/]+/[^/]+/mcmadmin(\..*)	/libs/mcm/content/admin$1	external: 302
^[^/]+/[^/]+/damadmin(\..*)	/libs/wcm/core/content/damadmin$1	internal
^[^/]+/[^/]+/geohome(\..*)	/content/geometrixx-outdoors$1	external: 302
^[^/]+/[^/]+/welcome(\..*)	/libs/cq/core/content/welcome$1	external: 302
^[^/]+/[^/]+/tagging(\..*)	/libs/cq/tagging/content/tagadmin$1	external: 302
^[^/]+/[^/]+/geohome(\..*)	/content/geometrixx$1	external: 302
^[^/]+/[^/]+/welcome(\..*)	/libs/crx/core/content/welcome$1	external: 302

Mappings for Resource Resolution

The following node types and properties provide for the definition of resource mappings:

Node Types

sling:ResourceAlias - mixin node type defines the sling:alias property, may be attached to any node, which does not allow setting a property named sling:alias

sling:MappingSpec - mixin node type defines the sling:match, sling:redirect, sling:status, and sling:internalRedirect properties

sling:Mapping - primary node type used to construct entries in /etc/map

Properties

sling:match - defines a partial regular expression used on node's name to match the incoming request

sling:redirect - causes a redirect response to be sent to the client

sling:status - defines the HTTP status code sent to the client

sling:internalRedirect - causes the current path to be modified internally to continue with resource resolution

sling:alias - indicates an alias name for the resource

Each entry in the mapping table is a regular expression, which is constructed from the resource path below ***/etc/map***. The following rules apply:

- If any resource along the path has a sling:match property, the respective value is used in the corresponding segment instead of the resource name.
- Only resources either having a ***sling:redirect*** or ***sling:internalRedirect*** property are used as table entries. Other resources in the tree are just used to build the mapping structure.

The following content:

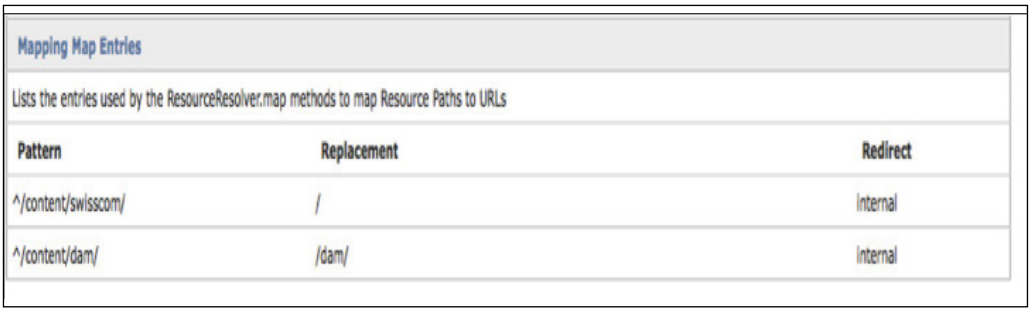
```
/etc/map
     +-- http
          +-- example.com.80
               +-- sling:redirect = "http://www.example.com/"
          +-- www.example.com.80
               +-- sling:internalRedirect = "/example"
          +-- any_example.com.80
               +-- sling:match = ".+\.example\.com\.80"
               +-- sling:redirect = "http://www.example.com/"
          +-- localhost_any
               +-- sling:match = "localhost\.\d*"
               +-- sling:internalRedirect = "/content"
               +-- cgi-bin
                    +-- sling:internalRedirect = "/scripts"
               +-- gateway
                    +-- sling:internalRedirect = "http://gbiv.com"
               +-- (stories)
                    +-- sling:internalRedirect = "/anecdotes/$1"
```

defines the following mapping entries:

Regular Expression	Redirect	Internal	Description
http/example.com.80	http://www.example.com	no	Redirect all requests to the Second Level Domain to www
http/www.example.com.80	/example	yes	Prefix the URI paths of the requests sent to this domain with the string `/example`
http/.+\.example\.com\.80	http://www.example.com	no	Redirect all requests to sub domains to www. The actual regular expression for the host.port segment is taken from the `sling:match` property.
http/localhost\.\d*	/content	yes	Prefix the URI paths with `/content` for requests to localhost, regardless of actual port the request was received on. This entry only applies if the URI path does not start with `/cgi-bin`, `gateway` or `stories` because there are longer match entries. The actual regular expression for the host.port segment is taken from the `sling:match` property.
http/localhost\.\d*/cgi-bin	/scripts	yes	Replace the `/cgi-bin` prefix in the URI path with `/scripts` for requests to localhost, regardless of actual port the request was received on.
http/localhost\.\d*/gateway	http://gbiv.com	yes	Replace the `/gateway` prefix in the URI path with http://gbiv.com for requests to localhost, regardless of actual port the request was received on.
http/localhost\.\d*/(stories)	/anecdotes/stories	yes	Prepend the URI paths starting with `/stories` with `/anecdotes` for requests to localhost, regardless of actual port the request was received on.

Mapping Nodes to Resources

JCR Nodes need to be mapped onto resources in order to be useful to Sling. The ResourceResolver.resolve() resolves the resource from the given URI. Use ResourceResolver.map() to get the resource path and create links.



Mapping Map Entries

Lists the entries used by the ResourceResolver.map methods to map Resource Paths to URLs

Pattern	Replacement	Redirect
^/content/swisscom/	/	internal
^/content/dam/	/dam/	internal

Adapting Resources

An adapter helps two incompatible interfaces to work together. The Adapter design pattern is used when you want two different classes with incompatible interfaces to work together. Sling offers an Adapter Pattern to conveniently translate objects that implement the Adaptable interface. This interface provides a generic *adaptTo()* method that will translate the object to the class type being passed as the argument.

The Resource and ResourceResolver interfaces are defined with a method adaptTo(), which adapts the object to other classes. The adaptTo pattern is used to adapt two different interfaces.
For example: Resource to Node.

```
Node node = resource.adaptTo(Node.class)
```

Using this mechanism, the JCR session of the resource resolver calls the adaptTo method with the javax.jcr.Session class object. Likewise the JCR node on which a resource is based can be retrieved by calling the *Resource.adapt*To method with the *javax.jcr.*Node class object.

.adaptTo() Use Cases

adaptTo() is now used for many use cases, for example:

- get implementation-specific objects
  - JCR-based implementation of generic Resource interface (to get access to underlying JCR Node)

- shortcut creation of objects that require internal context objects to be passed
  - JCR-based ResourceResolver holds a reference to the request's JCR Session, which is needed for many objects that will work based on that request session (e.g. PageManager or UserManager)

Examples - Resources can be adapted to:

Node	if this is a JCR-node-based resource or a JCR property referencing a node
Property	if this is a JCR-property-based resource
Item	if this is a JCR-based resource (node or property)
Map	returns a map of the properties, if this is a JCR-node-based resource (or other resource supporting value maps)
ValueMap	returns a convenient-to-use map of the properties, if this is a JCR-node-based resource (or other resource supporting value maps); can be done simpler by using ResourceUtil.getValueMap(Resource) (handles null case etc.)
PersistableValueMap	if this is a JCR-node-based resource and the user has permissions to modify properties on that node (Note: multiple persistable maps don't share their values)
InputStream	returns the binary content of a "file" resource (if this is a JCR-node-based resource and the node type is `nt:file` or `nt:resource`; if this is a bundle resource; file content if this is a file system resource) or the data of a binary JCR property resource
URL	returns an URL to the resource (repository URL of this node if this is a JCR-node-based resource; jar bundle URL if this is a bundle resource; file URL if this is a file system resource)
File	if this is a file system resource
SlingScript	if this resource is a script (eg. jsp file) for which a script engine is registered with sling
Servlet	if this resource is a script (eg. jsp file) for which a script engine is registered with sling or if this is a servlet resource
Authorizable (jackrabbit)	if this is a an authorizable resource (from the `AuthorizableResourceProvider` in org.apache.sling.jackrabbit.usermanager, under `/system/userManager`)

Resource Resolver can be adapted to:

Session	request's JCR session, if this is a JCR-based resource resolver (default)
PageManager	
ComponentManager	
Designer	
AssetManager	based on the JCR session, if this is a JCR-based resource resolver
TagManager	based on the JCR session, if this is a JCR-based resource resolver
UserManager	based on the JCR session, if this is a JCR-based resource resolver, and if the user has permissions to access the UserManager
Authorizable (cq-security)	the current user
User (cq-security)	
PrivilegeManager	
Preferences	preferences of the current user (based on JCR session if this is a JCR-based resource resolver)
PreferencesService	
PinManager	
QueryBuilder	
Ticket (contentbus)	adapts the underlying JCR session to a legacy cq4 ticket
BlogManager	based on the JCR session, if this is a JCR-based resource resolver
CalendarManager	based on the JCR session, if this is a JCR-based resource resolver

Additional Information

Full list at:

http://dev.day.com/docs/en/cq/current/developing/sling-adapters.html

This reference is important as you cannot get code completion for the adapter patterns.

Servlets

Servlets are modules of Java code that run in a server application to answer client requests. They are server-side constructs that provide component-based, platform-independent methods for building Web-based applications. A servlet can almost be thought of as an applet that runs on the server side--without a face.

Servlets make use of the Java standard extension classes in the packages *javax.servlet* (the basic Servlet framework) and *javax.servlet.http* (extensions of the Servlet framework for Servlets that answer HTTP requests). Since Servlets are written in the highly portable Java language and follow a standard framework, they provide a means to create sophisticated server extensions in a server and operating system independent way.

Although servlets are not limited to specific protocols, they are most commonly used with the HTTP protocols and the term "Servlet" is often used synonymously as "HTTP Servlet".

Servlets can access a library of HTTP-specific calls, receive all the benefits of the mature java language including portability, performance, reusability, and crash protection. Servlets are the popular choice for building interactive web applications. HTTP Servlets are typically used to:

- Provide dynamic content like getting the results of a database query
- Process and/or store the data submitted by the HTML client
- Manage information about the state of a stateless HTTP request

For example, an online shopping cart manages requests for multiple concurrent customers.

EXERCISE 3.1 Servlets

Goal

- Write a servlet that serves only GET requests
- Expose it at static URL
- Response shall contain the JCR repository properties as JSON

Steps

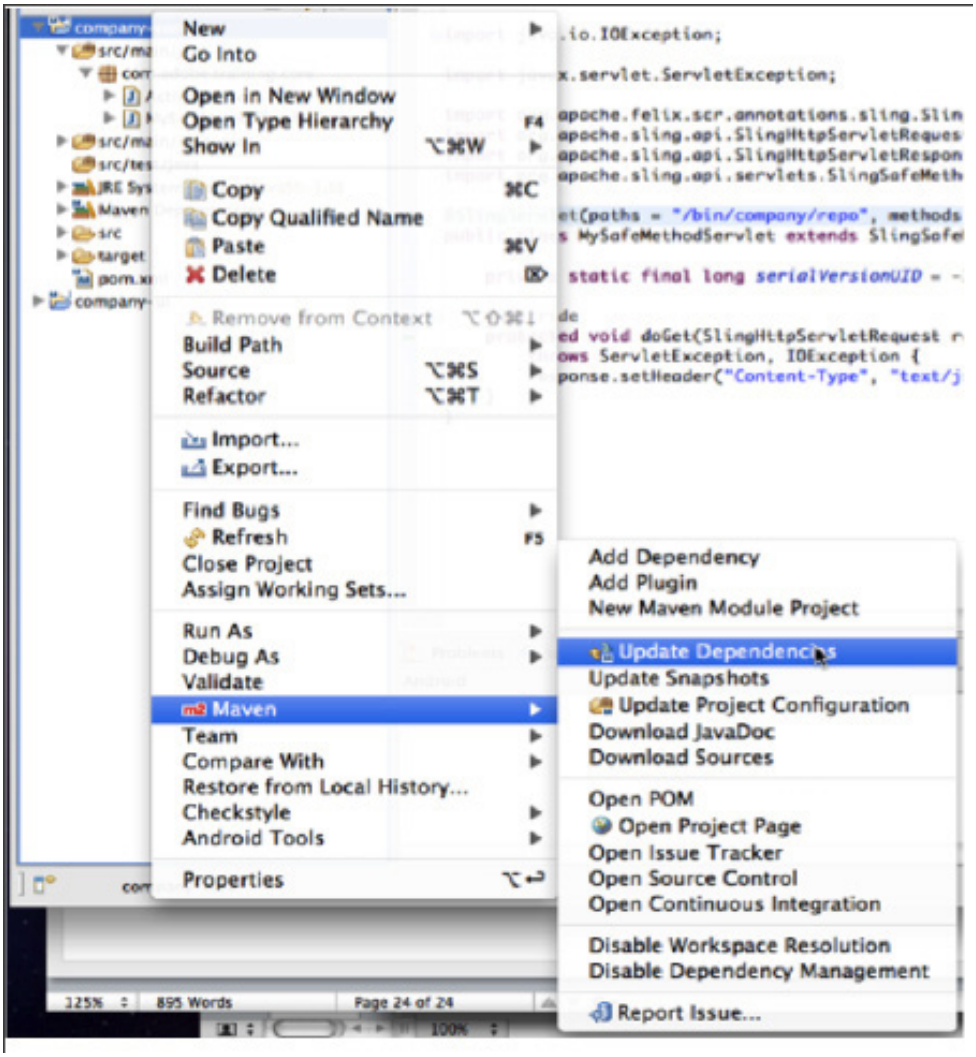
1. Add the necessary dependencies to the parent **pom.xml** In **pom.xml** , in the <dependency management> section, just before the <!-- Testing --> comment, add:

```
    <dependency>
        <groupId>org.apache.sling</groupId>
        <artifactId>org.apache.sling.api</artifactId>
        <version>2.3.0</version>
        <scope>provided</scope>
    </dependency>

<!-- servlet API -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.4</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>javax.servlet.jsp</groupId>
        <artifactId>jsp-api</artifactId>
        <version>2.1</version>
        <scope>provided</scope>
    </dependency>
```

In company-core/pom.xml make a reference to these dependencies, add:

```
    <!-- Apache Sling Dependencies -->
    <dependency>
            <groupId>org.apache.sling</groupId>
            <artifactId>org.apache.sling.api</artifactId>
    </dependency>

    <!-- Servlet API -->
    <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
        </dependency>
    <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>jsp-api</artifactId>
    </dependency>
```

In Eclipse to make sure that the Maven Eclipse picks up these changes, execute Update Dependencies



Alternatively you could have executed on the command line,

```
mvn eclipse:clean

mvn eclipse:eclipse
```

and refreshed the project in Eclipse.

mvn eclipse:clean removes the eclipse resources like the .settings-folder, the .project-file and the .classpath-file.

mvn eclipse:eclipse generates all eclipse resources needed to import the project into the eclipse IDE.

2. **Right click** on the company-core project and add a servlet class MySafeMethodServlet that extends Sling Safe Methods Servlet (so we need to implement only the GET method). The servlet path should be, /bin/company/repo.


```
package com.adobe.training.core;

import java.io.IOException;

import javax.servlet.ServletException;

import org.apache.felix.scr.annotations.sling.SlingServlet;
import org.apache.sling.api.SlingHttpServletRequest;
import org.apache.sling.api.SlingHttpServletResponse;

import org.apache.sling.api.servlets.SlingSafeMethodsServlet;

@SlingServlet(paths = "/bin/company/repo", methods = "GET")
public class MySafeMethodServlet extends
SlingSafeMethodsServlet {
        private static final long serialVersionUID =
-3960692666512058118L;
        @Override

        protected void doGet(SlingHttpServletRequest request,
SlingHttpServletResponse response)
        throws ServletException, IOException {
        response.setHeader("Content-Type", "application/json");
        response.getWriter().print("{\"coming\" : \"soon\"}");
        }
                }
```

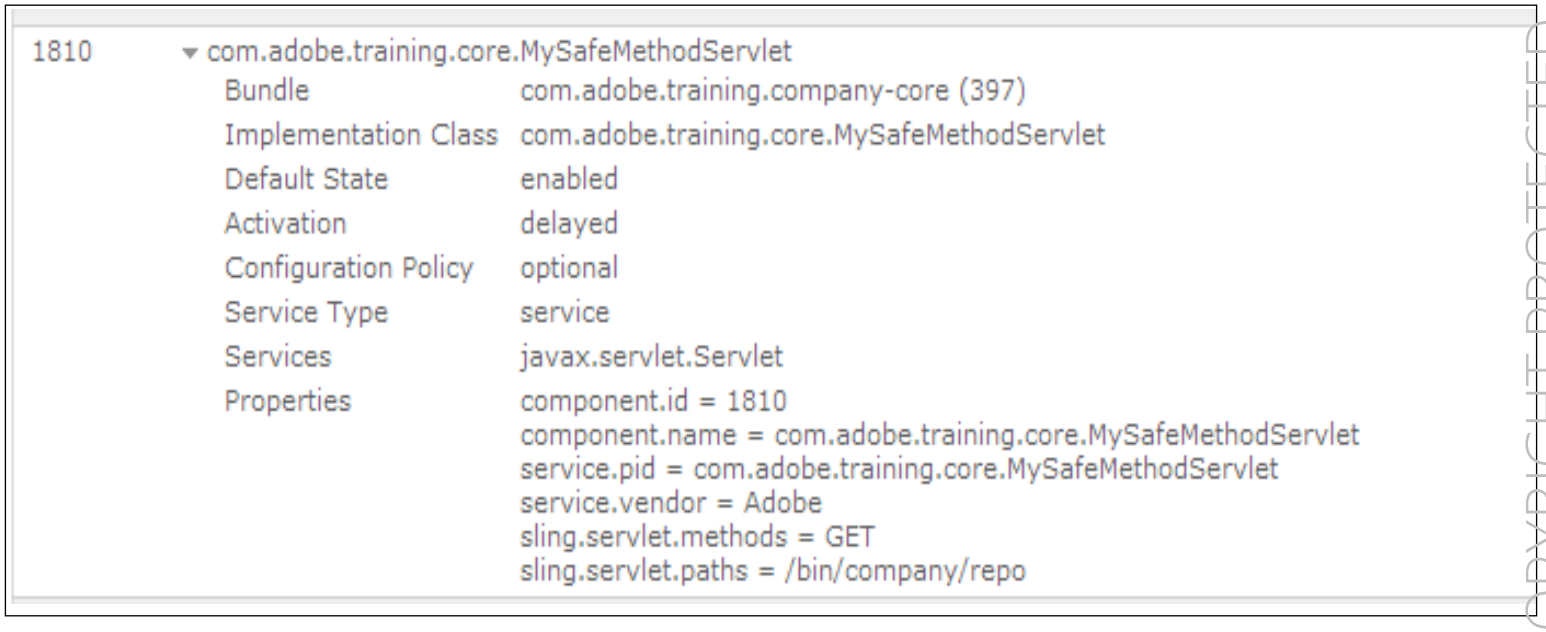
3. Deploy the code to your local AEM instance by executing in company-core directory:

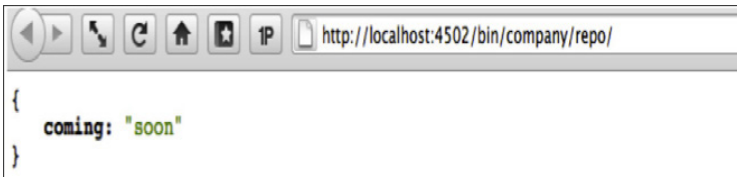
```
mvn clean install -P bundle
```

Check that the component is installed and works as expected:

NOTE: This Maven command will not work if you are running the parent pom.xml.

Note: Maven options: clean = remove class files -P bundle = create a bundle, i.e. the OSGi bundle class.
-P full = create a content package in CRX



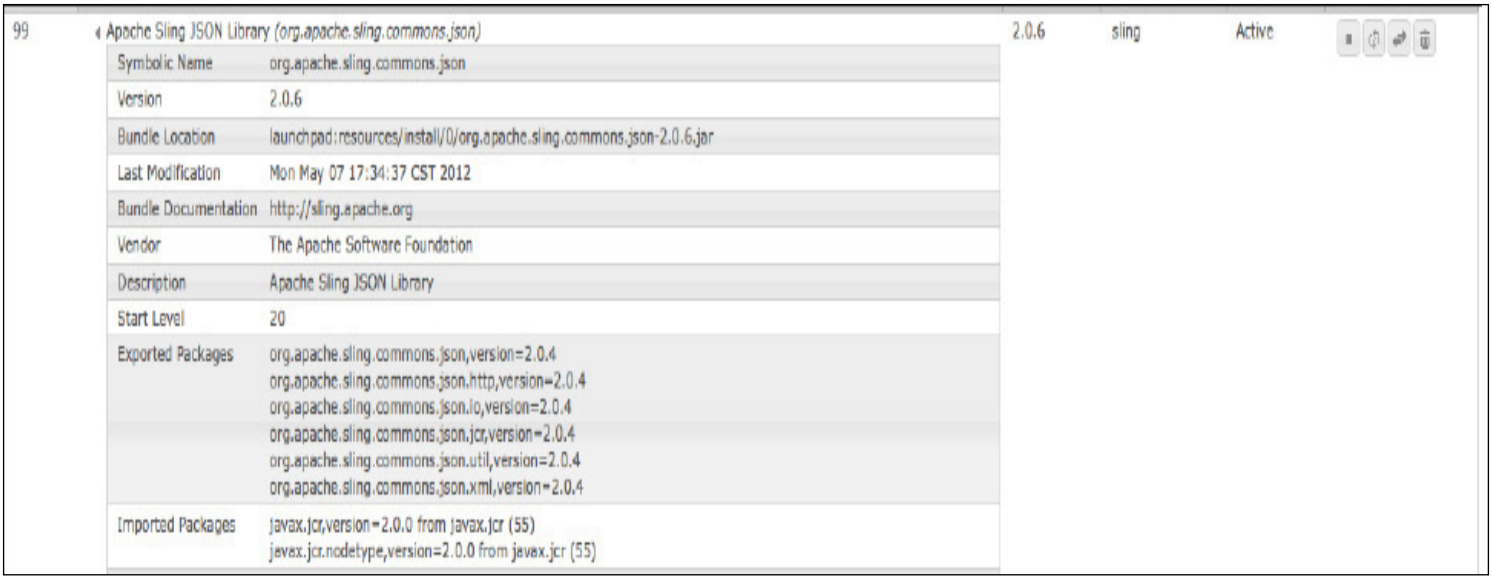


4. Find the JCR 2.0 dependency on mvnrepository.com and add it the the parent pom.xml and a reference in company-core/pom.xml.

NOTE: You must set the scope to "provided" in the parent pom.xml, i.e.

```
<dependency>
<groupId>javax.jcr</groupId>
<artifactId>jcr</artifactId>
<version>2.0</version>
<scope>provided</scope>
</dependency>
```

5. In the AEM Web Console find the Sling JSON bundle and determine the needed mvn dependency. Add it to the parent pom.xml and a reference in company-core/pom.xml.



99	Apache Sling JSON Library (org.apache.sling.commons.json)	2.0.6	sling	Active

Symbolic Name	org.apache.sling.commons.json
Version	2.0.6
Bundle Location	launchpad:resources/install/0/org.apache.sling.commons.json-2.0.6.jar
Last Modification	Mon May 07 17:34:37 CST 2012
Bundle Documentation	http://sling.apache.org
Vendor	The Apache Software Foundation
Description	Apache Sling JSON Library
Start Level	20
Exported Packages	org.apache.sling.commons.json,version=2.0.4 org.apache.sling.commons.json.http,version=2.0.4 org.apache.sling.commons.json.io,version=2.0.4 org.apache.sling.commons.json.jcr,version=2.0.4 org.apache.sling.commons.json.util,version=2.0.4 org.apache.sling.commons.json.xml,version=2.0.4
Imported Packages	javax.jcr,version=2.0.0 from javax.jcr (55) javax.jcr.nodetype,version=2.0.0 from javax.jcr (55)

```
<dependency>
        <groupId>org.apache.sling</groupId>
        <artifactId>org.apache.sling.commons.json</artifactId>
        <version>2.0.6</version>
        <scope>provided</scope>
</dependency
```

6. In Eclipse update the dependencies like you did previously. From now on, do this always when you change the POMs

7. Add the bolded code to Use the JSON library to render all repository properties into JSON:

```
package com.adobe.training.core;

import java.io.IOException;

import javax.jcr.Repository;
import javax.servlet.ServletException;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.sling.SlingServlet;
import org.apache.sling.api.SlingHttpServletRequest;
import org.apache.sling.api.SlingHttpServletResponse;
import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
import org.apache.sling.commons.json.JSONException;
import org.apache.sling.commons.json.JSONObject;
        @SlingServlet(paths = "/bin/company/repo", methods = "GET")
public class MySafeMethodServlet extends
```

```
SlingSafeMethodsServlet {
        private static final long serialVersionUID =-3960692666512058118L;

        @Reference
        private Repository repository;

        @Override
protected void doGet(SlingHttpServletRequest request,
SlingHttpServletResponse response)
        throws ServletException, IOException {
        response.setHeader("Content-Type", "application/json");
        //response.getWriter().print("{\"coming\" : \"soon\"}");

        String[] keys = repository.getDescriptorKeys();
        JSONObject jsonObject = new JSONObject();
        for (int i = 0; i< keys.length; i++) {
                try{
                  jsonObject.put(keys[i], repository.getDescriptor(keys[i]));
                }
                catch (JSONException e) {
                        e.printStackTrace();
                }
        }
        response.getWriter().print(jsonObject.toString());
  }
}
```

8. Deploy the bundle into the repository using mvn clean install -P full

9. Examine the results:


```
{"query.xpath.doc.order":"false","option.retention.supported":"false","node.type.management.same.name.siblings.supported":"f
alse","jcr.specification.version":"2.0","crx.cluster.master":"true","option.privilege.management.supported":"true","write.su
pported":"true","level.2.supported":"true","option.xml.export.supported":"true","option.versioning.supported":"true","node.t
ype.management.update.in.use.suported":"false","option.journaled.observation.supported":"false","option.baselines.supported"
:"false","option.lifecycle.supported":"false","jcr.repository.version":"0.20.0-
r1591473","option.shareable.nodes.supported":"false","query.xpath.pos.index":"false","option.xml.import.supported":"true","n
ode.type.management.orderable.child.nodes.supported":"true","node.type.management.overrides.supported":"true","option.transa
ctions.supported":"false","option.workspace.management.supported":"false","node.type.management.inheritance":"node.type.mana
gement.inheritance.single","option.simple.versioning.supported":"false","option.update.mixin.node.types.supported":"true","n
ode.type.management.autocreated.definitions.supported":"true","node.type.management.residual.definitions.supported":"true","
level.1.supported":"true","node.type.management.primary.item.name.supported":"true","option.unfiled.content.supported":"fals
e","option.principal.management.supported":"true","node.type.management.multivalued.properties.supported":"true","jcr.specif
ication.name":"Content Repository for Java Technology API","jcr.repository.name":"Apache Jackrabbit
Oak","option.access.control.supported":"true","option.locking.supported":"true","query.joins":"query.joins.none","option.act
ivities.supported":"false","crx.cluster.id":"a091c351-4c7b-406d-9895-
3b9816abc770","node.type.management.multiple.binary.properties.supported":"true","crx.repository.systemid":"62d07dcc-c2b8-
4210-82ec-
fbfc8c28d5c8","query.stored.queries.supported":"false","option.node.and.property.with.same.name.supported":"true","identifie
r.stability":"identifier.stability.method.duration","jcr.repository.vendor.url":"http://www.apache.org/","node.type.manageme
nt.value.constraints.supported":"true","option.update.primary.node.type.supported":"true","option.user.management.supported"
:"true","option.node.type.management.supported":"true","option.query.sql.supported":"false","jcr.repository.vendor":"The
Apache Software Foundation","query.full.text.search.supported":"false","option.observation.supported":"true"}
```

04

Sling Events

Apache Sling provides support for eventing, handling jobs, and scheduling. Sling's event mechanism leverages the OSGi Event Admin Specification. The OSGi API for events is very simple and lightweight. Sending an event just involves generating the event object and calling the event admin. Receiving an event requires implementation of a single interface and a declaration, through properties, which topics one is interested in. Each event is associated with an event topic, and event topics are hierarchically organized.

The OSGi specification for event handling uses a publish/subscribe mechanism based on these hierarchical topics. There is a loose coupling of the services, based on the whiteboard pattern. When the publisher has an event object to deliver, it calls all event listeners in the registry.

Various types of events can be handled. Predefined events include ***Framework Events***, for example a Bundle event indicating a change in a bundle's lifecycle, ***Service Events*** which indicate a service lifecycle change, and ***Configuration Events*** which indicate that a configuration has been updated or deleted. There are also:

- JCR observation events
- Application generated events
- Events from messaging systems (~JMS)
- "External events"

The event mechanism is an event mechanism which should not be confused with a *messaging* mechanism. Events are received by the event mechanism and distributed to registered listeners. Concepts like durable listeners, guarantee of processing etc. are not part of the event mechanism itself.

Listening to OSGi Events

The event listening mechanism must:

- Implement the EventHandler interface.
- Subscribe by service registration using the Whiteboard pattern, i.e. polling for events.
- Select the event with service properties, based on the event topic and a filter.

The property event.topics is set to the name of the specific topic which is to be listened for:

@scr.property name="event.topics" valueRef="<some topic>"
e.g.
@scr.property name="event.topics"
valueRef="ReplicationAction.EVENT_TOPIC"

or

@scr.property name="event.topics"
valueRef="org.apache.sling.api.SlingConstants.TOPIC_RESOURCE_ADDED"

Notice that the first example uses a AEM topic from com.day.cq.replication. ReplicationAction, whereas the second example is taken from the SlingConstants. You can refer to the org.apache.sling.api.SlingConstants class in the Javadocs to find out about other events available in Sling.

The annotation @Service is set to value = EventHandler.class to indicate that this is an event handler service. The code outline looks like this:

```
@Component
@Property(name = "event.topics", value =
        ReplicationAction.EVENT_TOPIC)
@Service(value = EventHandler.class)
public class MyEventListener implements JobProcessor,
        EventHandler {

        public void handleEvent(Event event) {
          if (EventUtil.isLocal(event)) {
                JobUtil.processJob(event, this);
          }
        }
```

Publishing Events

The steps to publish an event are:

- Get the EventAdmin service
- Create the event object (with a topic, and properties)
- Send the event (synchronously or asynchronously)

Job Events

Job events are a special category of event that were introduced into Sling. Job Events are guaranteed to be processed. Therefore someone has to do something with the event. Job Events are typically used in situations such as sending notification messages or post-processing images, and these events typically must be handled once and once only.

A job event is an event with the topic org/apache/sling/event/job and in this case, the real topic of the event is stored in the event.job.topic property.

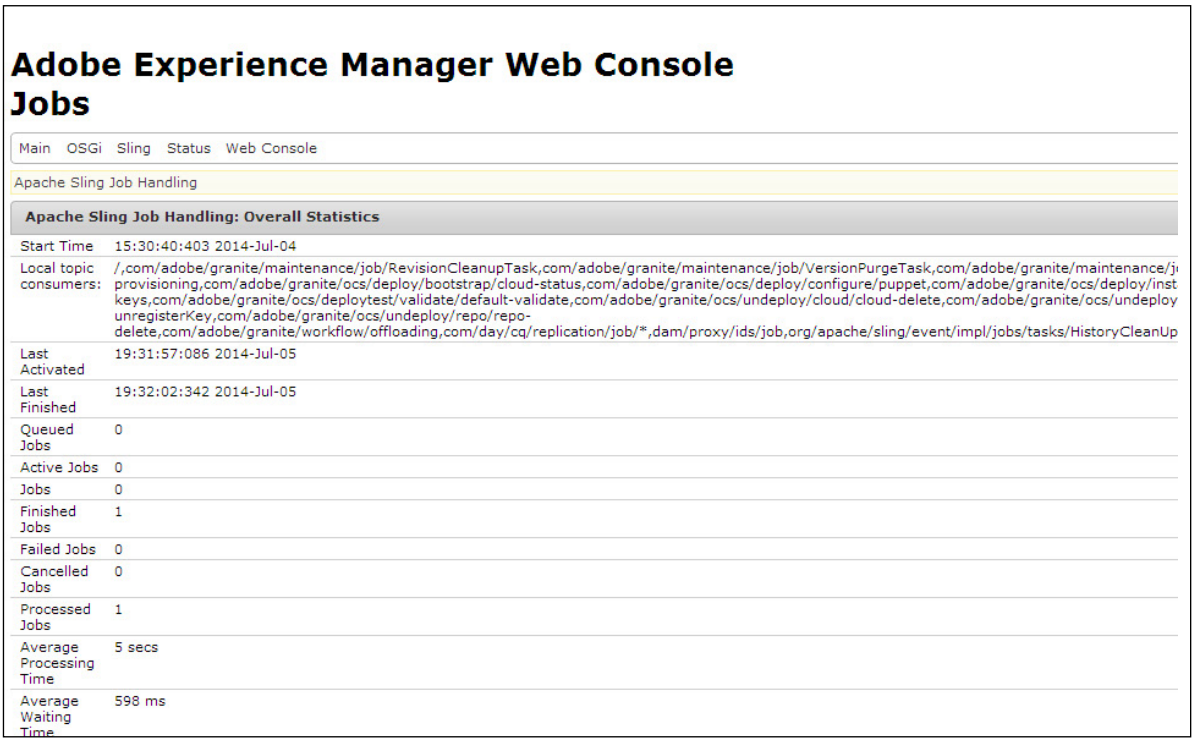
Sending Job Events

To send a job event the service needs to implement:

- the `org.osgi.service.event.EventHandler` interface.
- the `org.apache.sling.event.JobProcessor` interface.

Admin User Interface

Two tabs of the Adobe AEM Web Console contain information relating to events which may be helpful for debugging. Event statistics are shown under Sling > Jobs:



Adobe Experience Manager Web Console
Jobs

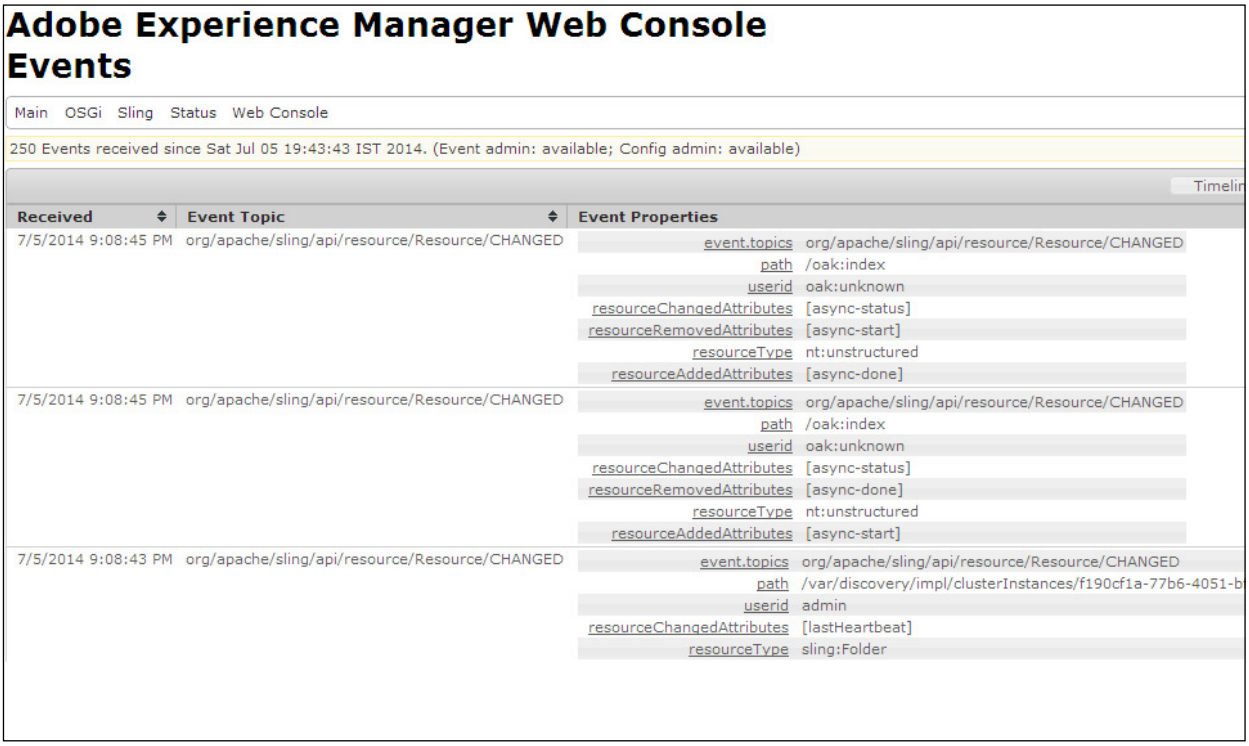
Main OSGi Sling Status Web Console

Apache Sling Job Handling

Apache Sling Job Handling: Overall Statistics

Start Time	15:30:40:403 2014-Jul-04
Local topic consumers:	/,com/adobe/granite/maintenance/job/RevisionCleanupTask,com/adobe/granite/maintenance/job/VersionPurgeTask,com/adobe/granite/maintenance/j provisioning,com/adobe/granite/ocs/deploy/bootstrap/cloud-status,com/adobe/granite/ocs/deploy/configure/puppet,com/adobe/granite/ocs/deploy/inst keys,com/adobe/granite/ocs/deploytest/validate/default-validate,com/adobe/granite/ocs/undeploy/cloud/cloud-delete,com/adobe/granite/ocs/undeploy unregisterKey,com/adobe/granite/ocs/undeploy/repo/repo- delete,com/adobe/granite/workflow/offloading,com/day/cq/replication/job/*,dam/proxy/ids/job,org/apache/sling/event/impl/jobs/tasks/HistoryCleanUp
Last Activated	19:31:57:086 2014-Jul-05
Last Finished	19:32:02:342 2014-Jul-05
Queued Jobs	0
Active Jobs	0
Jobs	0
Finished Jobs	1
Failed Jobs	0
Cancelled Jobs	0
Processed Jobs	1
Average Processing Time	5 secs
Average Waiting Time	598 ms

...and details of individual Events under OSGi > Events:



EXERCISE 4.1 - Event Handling

Goal

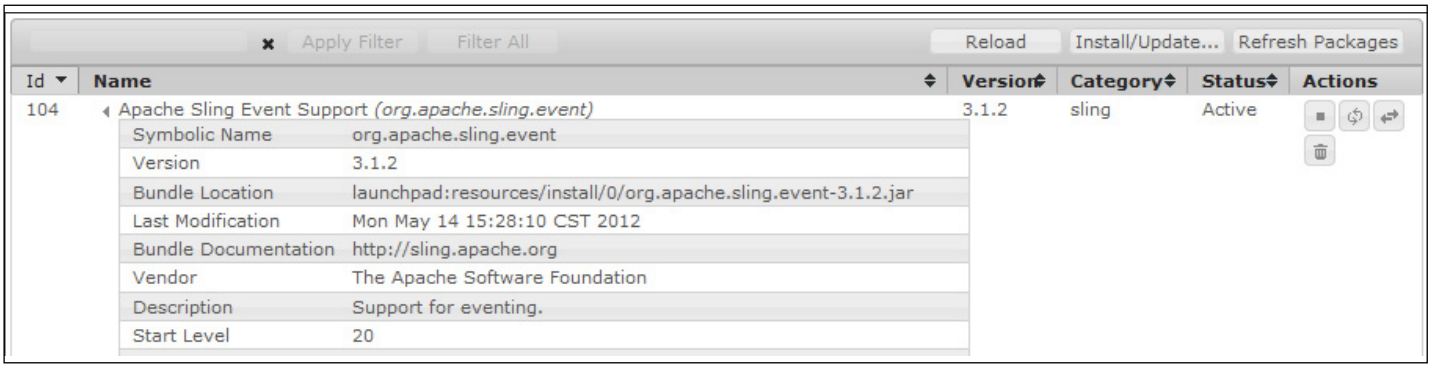
To demonstrate the implementation of an event handler, you are going to write an audit log for cq:Page replication events, which will listen for `ReplicationAction. EVENT _ TOPIC` events and log the page's title.

Steps

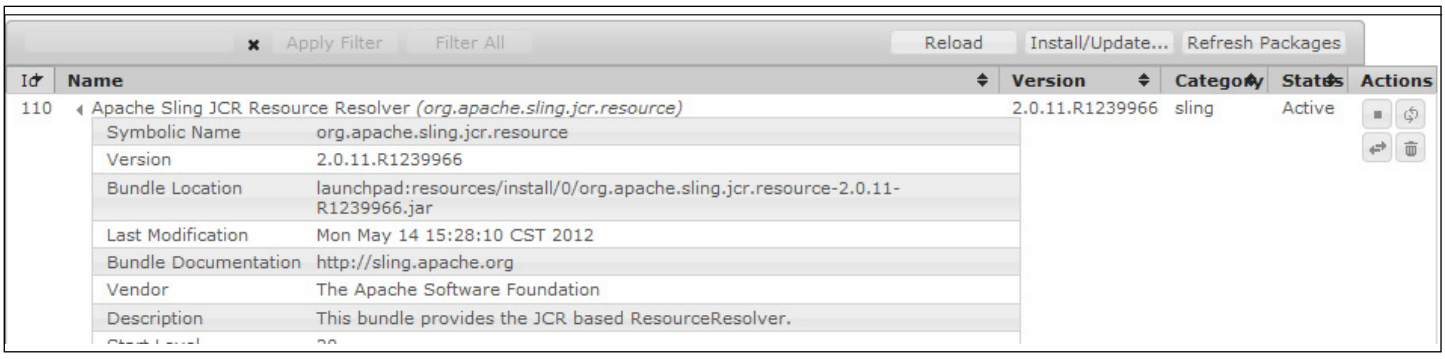
1. Start a publish instance on port 4503 and test that you can activate pages using the standard AEM Site Admin Interface.

2. For this exercise you will need the bundles com.day.cq.cq-replication, com.day.cq.wcm.cq-wcm-api, com.day.cq.cq-commons, org. apache.sling.jcr.resource and org.apache.sling.event. These have already been added to the pom.xml files for your convenience and to save time. If you needed other bundles, you would find them in the Adobe AEM Web Console and add them as dependencies to your pom.xml files.


```
<dependency>
        <groupId>com.day.cq.wcm</groupId>
        <artifactId>cq-wcm-api</artifactId>
        <version>5.6.2</version>
        <scope>provided</scope>
</dependency
```

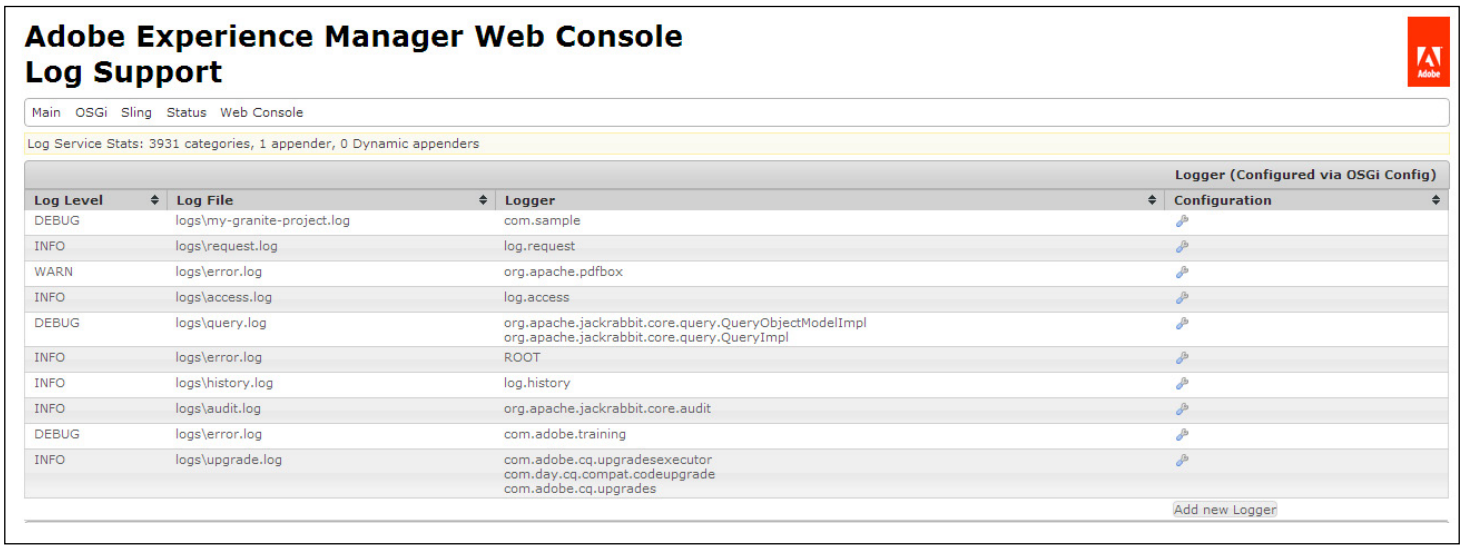



```
<dependency>
        <groupId>org.apache.sling</groupId>
        <artifactId>org.apache.sling.event</artifactId>
        <version>3.1.4</version>
        <scope>provided</scope>
</dependency
```



```
<dependency>
        <groupId>org.apache.sling</groupId>
        <artifactId>org.apache.sling.jcr.resource</artifactId>
        <version>2.2.2</version>
        <scope>provided</scope>
</dependency
```

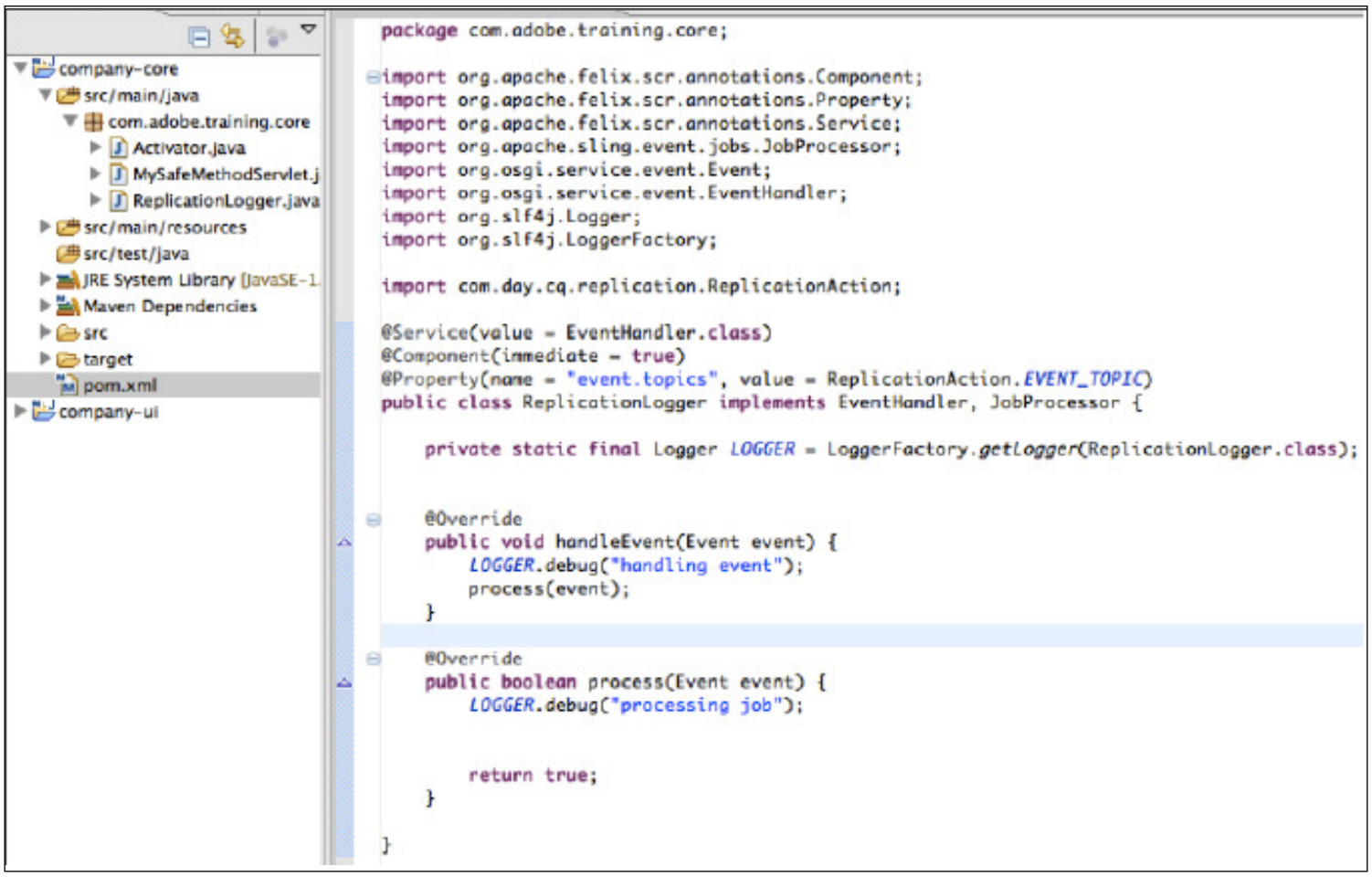
3. On the author instance, set the log level for `com.adobe.training` to debug. This can be done in the Adobe AEM Web Console (though Adobe recommends that permanent configuration changes are normally made directly in the repository - see http://docs.adobe.com/docs/en/aem/6-0/deploy/configuring/configuring-osgi.html for more details). Open the **Configuration** tab, and create a new **Logging Logger** factory configuration, then apply the Logger name com.adobe.training and the Log Level of Debug. After refreshing, the new logger details should appear under the Sling Log Support tab. The new **Logging Logger** will direct its output to the `error.log` file as we haven't changed that setting, but only `com.adobe.training` messages will be logged at the debug level:



4. Add the basic Listener class in a file named ReplicationLogger.java as shown below, and deploy the bundle. Notice the **ReplicationAction.EVENT_TOPIC** property which is being listened for, and the `@Component(immediate = true)` statement which is needed to ensure that the component com.adobe.training.core.ReplicationLogger is active immediately - otherwise it would only be installed when used, and since it is supposed to be listening for events, this would not work.

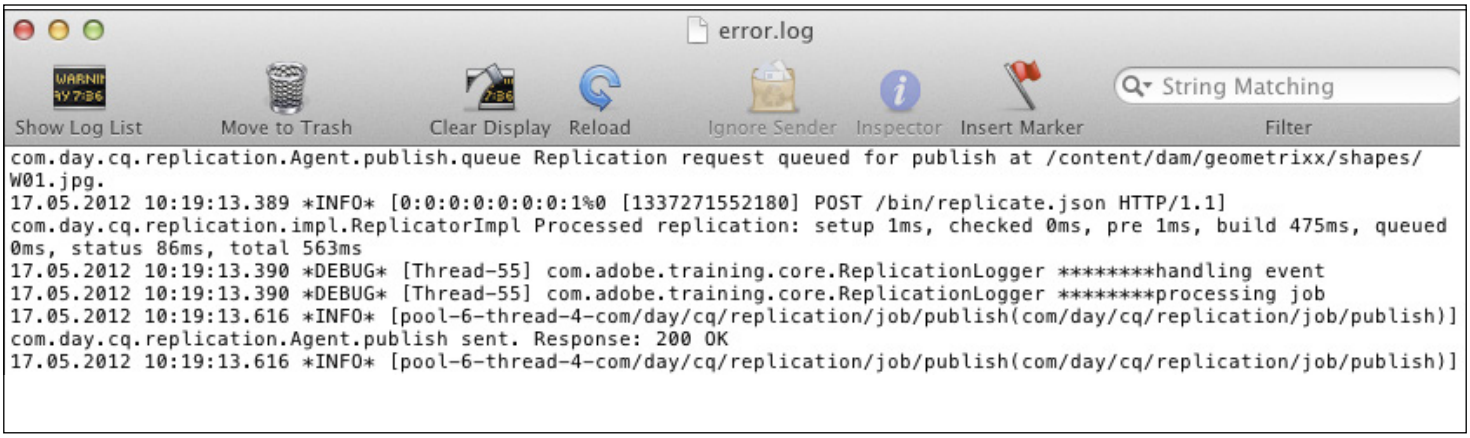
ReplicationLogger.java

```
package com.adobe.training.core;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Property;
import org.apache.felix.scr.annotations.Service;
import org.apache.sling.event.jobs.JobProcessor;
import org.osgi.service.event.Event;
import org.osgi.service.event.EventHandler;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.day.cq.replication.ReplicationAction;

@Service(value = EventHandler.class)
@Component(immediate = true)
@Property(name = "event.topics", value =
        ReplicationAction.EVENT_TOPIC)
public class ReplicationLogger implements EventHandler,
        JobProcessor {

private static final Logger LOGGER =
        LoggerFactory.getLogger(ReplicationLogger.class);

@Override
public void handleEvent(Event event) {
        LOGGER.debug("********handling event");
        process (event);
}

@Override
public boolean process(Event event) {
        LOGGER.debug("********processing job");

        return true;
    }
}
```



5. Rebuild and deploy the bundle

6. Activate a page in the Site Admin and inspect the log for the debug messages you put into the code:



7. Next you will add the code to log only page activation events. The `process()` method is extended to get the ReplicationAction from the event, and check whether it is the **ACTIVATE** type. If so, the page object is obtained, and the title is extracted to display in the log.

```
package com.adobe.training.core;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Property;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.apache.sling.api.resource.LoginException;
import org.apache.sling.api.resource.ResourceResolver;
import org.apache.sling.event.jobs.JobProcessor;
import org.apache.sling.jcr.resource.
JcrResourceResolverFactory;
import org.osgi.service.event.Event;
import org.osgi.service.event.EventHandler;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.day.cq.replication.ReplicationAction;
import com.day.cq.replication.ReplicationActionType;
import com.day.cq.wcm.api.Page;
import com.day.cq.wcm.api.PageManager;

@SuppressWarnings("deprecation")
@Service(value = EventHandler.class)
@Component(immediate = true)
@Property(name = "event.topics", value =
        ReplicationAction.EVENT_TOPIC)
public class ReplicationLogger implements EventHandler,
        JobProcessor {

private static final Logger LOGGER =
        LoggerFactory.getLogger(ReplicationLogger.class);

@Reference
private JcrResourceResolverFactory
        jcrResourceResolverFactory;

@Override
public void handleEvent(Event event) {
        LOGGER.debug("********handling event");
        process (event);
}
```

```
        @Override
public boolean process(Event event) {
        LOGGER.debug("********processing job");
        ReplicationAction action = ReplicationAction.fromEvent
          (event);

        ResourceResolver resourceResolver = null;
        if (action.getType().equals
          ReplicationActionType.ACTIVATE)) {
          try {
                resourceResolver = jcrResourceResolverFactory.
        getAdministrativeResourceResolver(null);
                final PageManager pm = resourceResolver.
        adaptTo(PageManager.class);
                final Page page = pm.getContainingPage
                        (action.getPath());
                if(page != null) {
                        LOGGER.debug
        ("********activation of page {}", page.getTitle());
                }

        }
        catch (LoginException e) {
                e.printStackTrace();
        }
        finally {
                if(resourceResolver != null &&
                        resourceResolver.isLive()) {
                resourceResolver.close();
                }
          }
        }
        return true;
  }
}
```

8. Again, activate a page in the Site Admin and inspect the log. You should see logged messages giving the titles of pages as they are activated.

NOTE: The resource resolver and how it is used with adaptTo to obtain a PageManager.

For didactic reason the example above used the deprecated JcrResourceResolverFactory. The correct way to retrieve a ResourceResolver from a JCR Session is:

```
javax.jcr.Session adminSession = repository.
loginAdministrative(null);
Map<String, Object> authInfo = new HashMap<String,
Object>();
authInfo.put(JcrResourceConstants.AUTHENTICATION_INFO_SESSION,
adminSession);
ResourceResolver resolver =
resourceResolverFactory.getResourceResolver(authInfo);
```

Congratulations! You have created an event listener that listens for page replication events, and logs the details when one occurs. This could easily be expanded to perform other operations triggered by some event on the server. Next we will look at scheduling events, so that you can make something happen when you want it to happen.

05

Sling Scheduling

The Sling Commons Scheduling mechanism is used to start jobs periodically using a cron definition. This can specify a complex time definition such as "at 6.32 pm on every Saturday and Sunday", or "at 2.30 am on the first Friday of every month". It makes use of the open source Quartz library. Cron expressions are explained here:

http://www.docjar.com/docs/api/org/quartz/CronExpression.html and
http://docs.oracle.com/cd/E12058_01/doc/doc.1014/e12030/cron_expressions.htm

The schedule can also be a simple period of time (in seconds) between each execution, or on a specific date and time.

A service which implements either java.lang.Runnable or org.quartz.job is started if it either contains a configuration property `scheduler.expression` (which contains a Cron expression) or scheduler. period (which contains a simple numerical period in seconds). It is also possible to use quartz methods to set more complex triggering arrangements, and to specify specific dates and times.

The job is started with the PID of the service. If the service has no PID, the configuration property `scheduler.name` must be set.

OSGi Service Fired By Quartz

To make use of the quartz library using the scheduler's whiteboard pattern, the following outline code is needed:

```
package <package name>;

@Component
@Service (interface="java.lang.Runnable")
@Property (name="scheduler.expression" value="0 0/10 * * * ?",
        type="String")

public class MyScheduledTask implements Runnable {

        public void run() {
                //place events to run here.
        }
}
```

- An `@Component` annotation is used to register the component. This will need to include immediate=true to activate the component immediately.
- The `@Service(interface="java.lang.Runnable")` annotation makes it possible to schedule this service.
- `@Property(name="scheduler.expression", value="0 0/10 * * * ?", type="String")` is where we write the quartz expression for the scheduled time interval.
- The class must also implement Runnable, and contain a run() method.

Scheduling At Periodic Times

Instead of specifying a quartz expression for the time interval, in simpler cases we can also just specify a time period, and the job will run repeatedly at this interval in seconds. For example, for a ten second interval:

```
@Property(name="scheduler.period", value="10", type="Long")
```

Preventing Concurrent Execution

If we do not want concurrent execution of the job, it can be prevented using the scheduler.concurrent parameter:

```
@Property(name="scheduler.concurrent",value="false",type="Boolean",
private="true")
```

Quartz Trigger Syntax

The Cron trigger expression comprises a series of parameters separated by white space, arranged as follows:

<Seconds> <Minutes> <hours> <Day of Month> <Month> <Day of Week> <Year>

It can include special characters such as:

- The '*' character to specify all values.
- The '?' character is allowed for the day-of-month and day-of-week fields. It is used to specify 'no specific value'. This is useful when you need to specify something in one of the two fields, but not the other.
- The '-' character is used to specify ranges. For example "10-12" in the hour field means "the hours 10, 11 and 12".
- The ',' character is used to specify additional values. For example "MON,WED,FRI" in the day-of-week field means "the days Monday, Wednesday, and Friday".

The '/' character is used to specify increments. For example "0/15" in the seconds field means "the seconds 0, 15, 30, and 45".

For example, "0 0 12 * * ?" means Fire at 12pm (noon) every day
For more details see:

http://www.docjar.com/docs/api/org/quartz/CronTrigger.html

Programmatically Scheduling Jobs

To programmatically schedule a job, you need a Runnable class, and can then add schedule times using appropriate methods:

```
@Reference
private Scheduler scheduler;
this.scheduler.addJob("myJob", job, null, "0 15 10 ? * MONFRI", true);

// periodic:
this.scheduler.addPeriodicJob("myJob", job, null, 3*60, true);

// one time
this.scheduler.fireJobAt("myJob", job, null, fireDate);
```

For more information on scheduling, see:

http://sling.apache.org/site/scheduler-service-commons-scheduler.html

EXERCISE 5.1 - Scheduling jobs

Goal

- Write a job that periodically deletes temporary nodes in the repository
- The configuration shall be stored in a configuration node which can be seen and edited in the Apache Felix console.

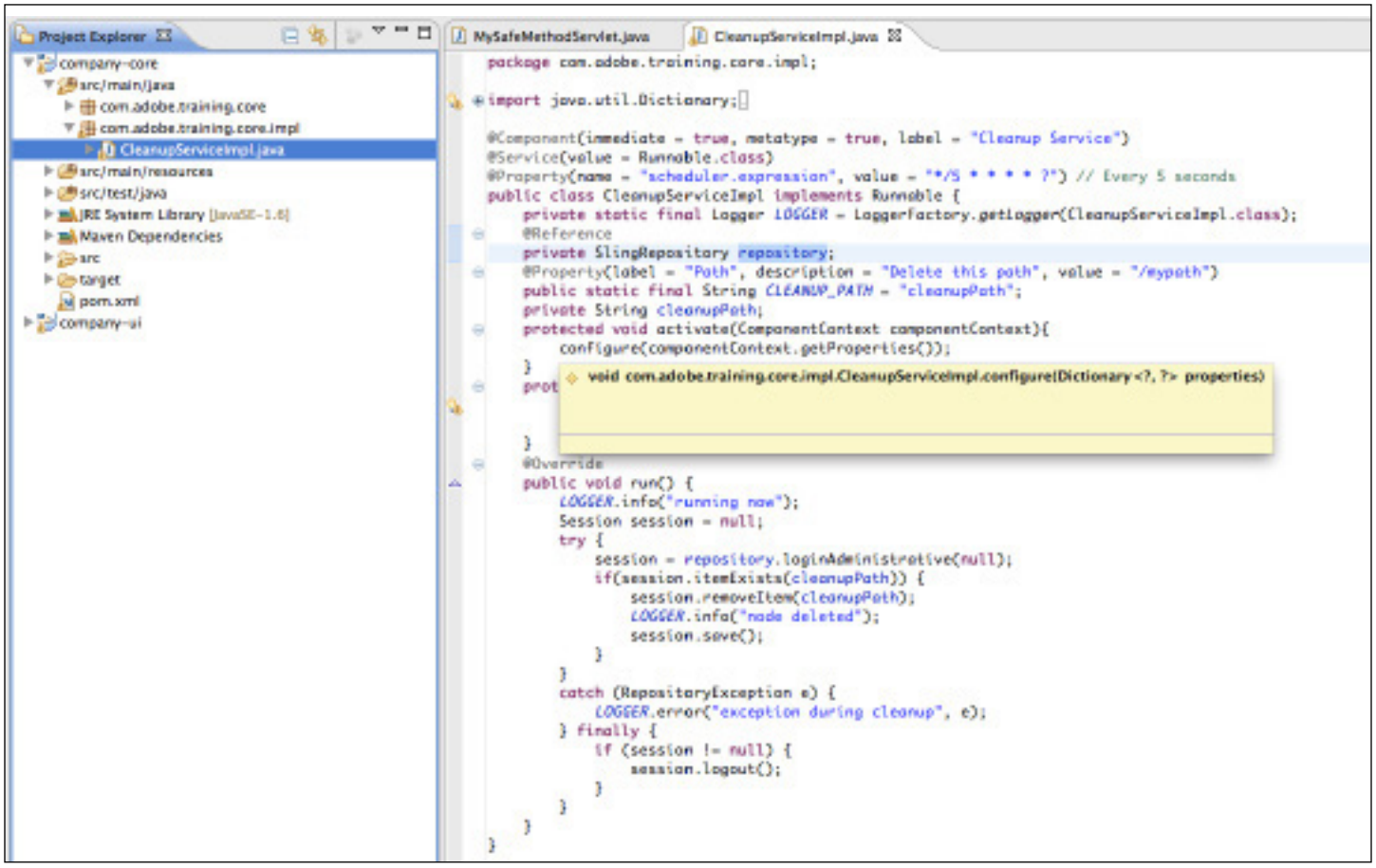
Steps

1. The basic setup of the exercise (with non-configurable properties), requires dependency org.apache.sling.jcr.api to be added to the pom.xml files. This has been done for you in the pom.xml files provided, to save time.

2. The code to be written into CleanupServiceImpl.java is shown below:

```
package com.adobe.training.core.impl;

import java.util.Dictionary;

import javax.jcr.RepositoryException;
import javax.jcr.Session;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Property;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.apache.sling.commons.osgi.OsgiUtil;
import org.apache.sling.jcr.api.SlingRepository;
import org.osgi.service.component.ComponentContext;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Component(immediate = true, metatype = true,
        label = "Cleanup Service")
@Service(value = Runnable.class)
@Property(name = "scheduler.expression",
        value = "*/5 * * * * ?") // Every 5 seconds
public class CleanupServiceImpl implements Runnable {

private static final Logger LOGGER =
        LoggerFactory.getLogger(CleanupServiceImpl.class);

@Reference
private SlingRepository repository;
```

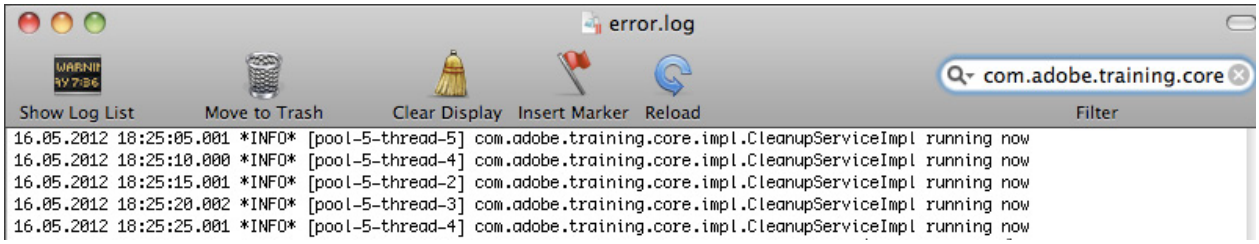
```
@Property(label = "Path", description = "Delete this path",
        value = "/mypath")
public static final String CLEANUP_PATH = "cleanupPath";
private String cleanupPath;

protected void activate(ComponentContext componentContext){
        configure(componentContext.getProperties());
}

protected void configure(Dictionary<?, ?> properties) {
        this.cleanupPath = OsgiUtil.toString(properties.get
          (CLEANUP_PATH), null);
        LOGGER.info("configure: cleanupPath='{}'",
          this.cleanupPath);
}
@Override
public void run() {
        LOGGER.info("running now");
        Session session = null;
        try {
          session = repository.loginAdministrative(null);
          if(session.itemExists(cleanupPath)) {
                session.removeItem(cleanupPath);
                LOGGER.info("node deleted");
                session.save();
           }
         }
         catch (RepositoryException e) {
            LOGGER.error("exception during cleanup", e);
          } finally {
            if (session != null) {
                session.logout();
           }
        }
    }
}
```



To note:

- metatype = true enables configuration in AEM Web Console.
- SlingRepository can be used for administrative logins, using loginAdministrative(null). Never forget to logout afterwards (in finally).
- configure will be called when the config changes. This is used to obtain the new value for the cleanup path when it is changed.

3. Deploy and inspect the log output:



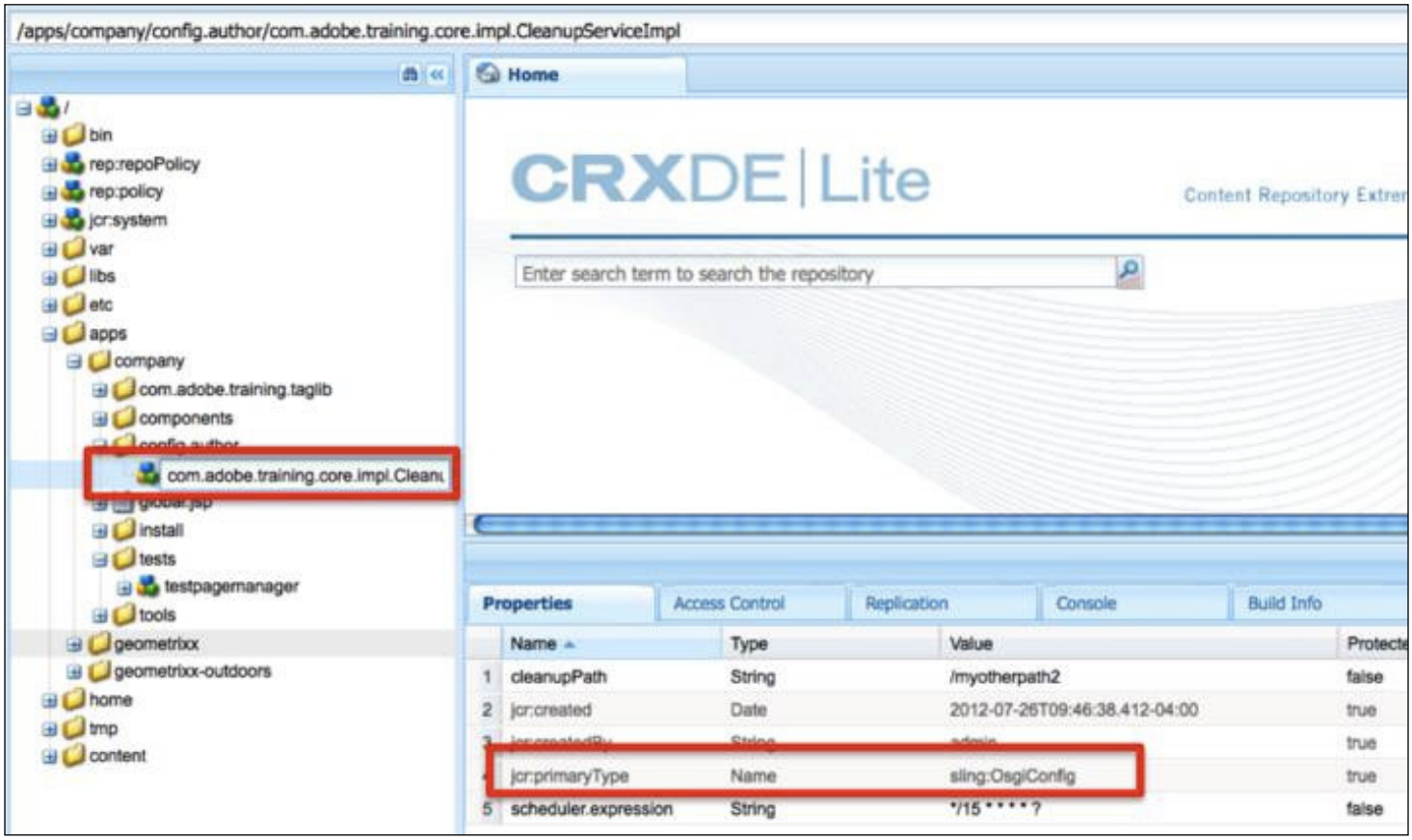
4. In CRXDE Lite create a node at the repository root called `/mypath`, save the changes, then see it get deleted by the cleanup service running every 5 seconds (you will need to refresh the web browser window for CRXDE Lite to see the node disappear).

5. **In the AEM web console components tab, click** on the wrench to the right of the com.adobe.training.core.impl.CleanupServiceImpl entry to open the Configuration pop up window. Configure a different path:

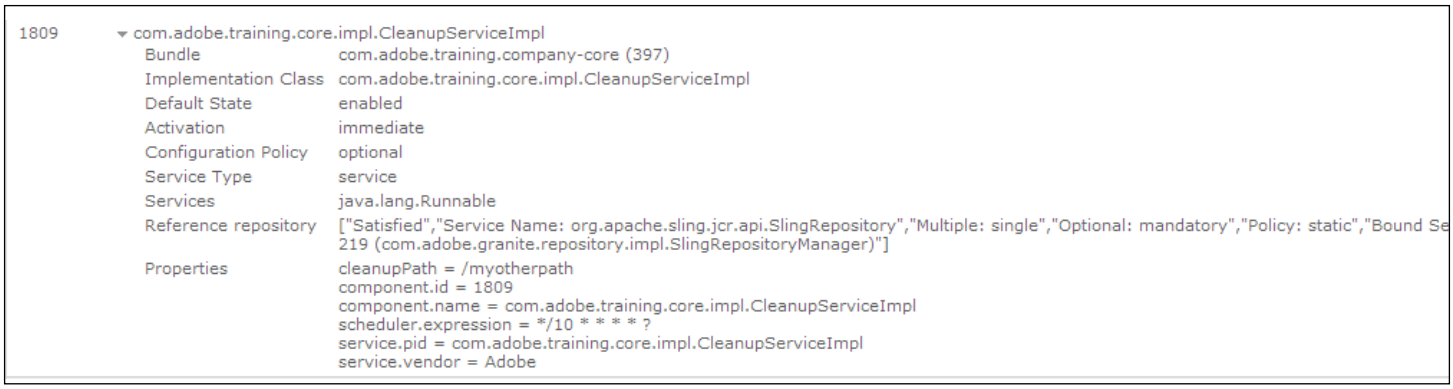
6. Now create a node with the new path name and confirm that this is now deleted.

7. Finally we will set the configuration in the repository. In CRXDE Lite create a folder (type nt:folder) /apps/company/config.author, and in the new folder, create a node of type `sling:OsgiConfig` with the name `com.adobe.training.core.impl.CleanupServiceImpl`. Add properties to this node as follows:

   - Name = `cleanupPath`, Type = String, Value = `/myotherpath`
   - Name = `scheduler.expression`, Type = String, Value = */10 * * * * ?



8. Confirm in the AEM Web Console that the configuration has changed:



9. Use vlt to serialize the config into your file system as a .content.xml file. Change to directory company-ui/src/main/content/jcr_root/ then execute the command:

   vlt update

   This will update folders and files configured in the `company-ui/src/main/content/META-INF/vault/filter.xml file`. Note that the credentials are not needed here as this information is already saved in your home directory inside `~/.vault/auth.xml`.

10. Now examine the contents of the `company-ui/src/main/content/jcr_root/apps/company` folder. You should see the newly created `config` folder, and inside this, the file `com.adobe.training.core.impl.CleanupServiceImpl.xml` Examine the contents of this xml file, and you will see the saved configuration properties that you created earlier.

NOTE: vlt update may be abbreviated to vlt up. Most vlt commands have an abbreviated version - see http://dev.day.com/docs/ en/crx/current/how_to/how_to_use_the_vlttool. html for more details on vlt commands.

Congratulations! You have successfully created a scheduled job, with settings that may be configured or viewed using Apache Felix, or through repository configurations. You have also seen how to export that information to your local file system.

06

JCR Basics, Content Modeling, Indexing, Search

Repository Basics

JCR API is defined in JSR-170 and JSR-283. Apache Jackrabbit is the reference implementation of these JSRs. Adobe CRX is the Adobe commercial extension of Jackrabbit.

JCR Review

The JCR provides a generic application data store for both structured and unstructured content. File systems provide excellent storage for unstructured, hierarchical content. Databases provide excellent storage for structured data due to transactional services and referential integrity functionality. The JCR provides the best of both data storage architectures, plus observation, versioning, and full text search.

The JCR presents a hierarchical content model where nodes and properties are defined as "Items".

JCR Features

- Query (XPath, SQL, JQL)
- Export/Import (XML)
- Referential Integrity
- Authentication
- Access Control
- Versioning
- Observation
- Locking and Transactions (JTA)

Repository Model

Inside a JCR repository, content is organized into one or more workspaces, each of which holds a hierarchical structure of nodes and properties. Beginning with the root node at the top, the hierarchy descends much like the directory structure of a file system: Each node can have zero or more child nodes and zero or more properties.

Properties cannot have children but do have values. The values of properties are where the actual pieces of data are stored. These can be of different types: strings, dates, numbers, binaries and so forth. The structure of nodes above the properties serves to organize this data according to whatever principles are employed by the application using the repository.

An additional advantage of the JCR is support for namespaces. Namespaces prevent naming collisions among items and node types that come from different sources and application domains. JCR namespaces are defined with a prefix, delimited by a single : (colon) character, for example - jcr:title.

Node Types

Node types define...

- The structure of application content
- A set of properties and their types
- A set of child nodes and their types

More about Nodes

- Node types may extend other node types (can have Super Types)
- Nodes are instances of a node type
- Nodes must have one primary node type
- Nodes may have one or more mixin node types
- Mixin node types mandate extra characteristics/functionality for a particular node in addition to those enforced by its primary node type

The CRX comes with some pre-defined node types.

- For reflection of repository-level functionality, e.g.
  - defining storage of versions ("nt:version")
  - defining storage of node types ("jcr:nodeType")
- For application-level common types, e.g.
  - simulating a conventional "file" and "folder" structure ("nt:file" & "nt:folder")
  - allowing storage of unstructured content ("nt:unstructured")

To see the node types registered in your CRX instance you can point your browser to the CRX main console:

http://<your-crx>/crx/explorer/index.jsp

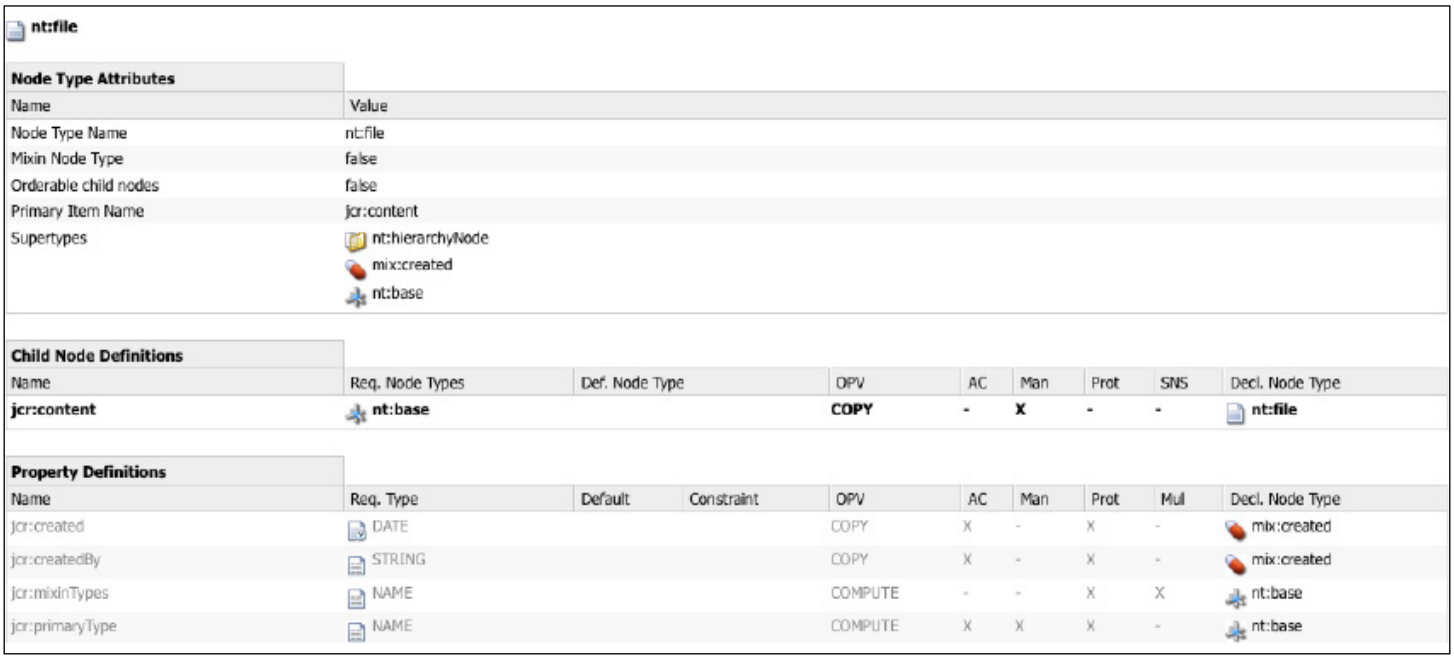
For example, in the default installation on your local machine this would be:

http://localhost:4502/crx/explorer/index.jsp

Node Type Definitions

- Node types are stored in the form of Node Type Definitions
- Those are reflected in the API as *javax.jcr.nodetype.NodeType*
- Call *javax.jcr.Node.getPrimaryNodeType()* to get the node type definition of a node
- A Node Type Definition consists of a set of mandatory attributes Node Type Definition in the CRX UI

Node Type Definition in the CRX UI:



nt:file

Node Type Attributes

Name	Value
Node Type Name	nt:file
Mixin Node Type	false
Orderable child nodes	false
Primary Item Name	jcr:content
Supertypes	nt:hierarchyNode mix:created nt:base

Child Node Definitions

Name	Req. Node Types	Def. Node Type	OPV	AC	Man	Prot	SNS	Decl. Node Type
jcr:content	**nt:base**		**COPY**	-	**X**	-	-	**nt:file**

Property Definitions

Name	Req. Type	Default	Constraint	OPV	AC	Man	Prot	Mul	Decl. Node Type
jcr:created	DATE			COPY	X	-	X	-	mix:created
jcr:createdBy	STRING			COPY	X	-	X	-	mix:created
jcr:mixinTypes	NAME			COMPUTE	-	-	X	X	nt:base
jcr:primaryType	NAME			COMPUTE	X	X	X	-	nt:base

Node Type Inheritance

- A node may have one or more Super Types
- Seen from the supertype, the inheriting node type is a subtype
- A subtype inherits:
  - The property definitions
  - The child node definitions
  - Other attributes (such as "isMixin")
- Supertypes are discoverable via the JCR API

Content Modeling: David's Model

1. Data First. Structure Later. Maybe.
2. Drive the content hierarchy, don't let it happen.
3. Workspaces are for clone(), merge() and update()
4. Beware of Same Name Siblings
5. References considered harmful

6. Files are Files are Files

7. ID's are evil

Data First. Structure Later. Maybe.

This topic is directed, for the most part, at those applications that are defining their own node types. Typically, AEM developers will use the node types defined for the AEM application.

However, if you create a new application that sits on the JCR, the following will be good advice:

- Learn to love nt:unstructured (& friends) in development
- Structure is expensive
    - for many use cases it is entirely unnecessary to explicitly declare structure (node types) to the underlying storage

NOTE: Structure means "node types", not hierarchy.

Drive the content hierarchy, don't let it happen.

"The content hierarchy is a very valuable asset. So don't just let it happen, design it. If you don't have a "good", human-readable name for a node, that's probably something that you should reconsider. Arbitrary numbers are hardly ever a 'good name'." -- David Neuscheler

The content hierarchy drives

- ACLs
- URLs
- Caching (see Dispatcher later)
- Search and content organization

Workspaces are for clone(), merge() and update()

When creating applications that use the CRX, put your use of workspaces to the following test:

- If you have a considerable overlap of "corresponding" nodes (essentially the nodes with the same UUID) in multiple workspaces you have probably put workspaces to good use.
- If there is no overlap of nodes with the same UUID you are probably abusing workspaces.

Beware of Same Name Siblings

The use of Same Name Siblings (SNS) was introduced into the JCR specification to allow compatibility with data structures that are designed and expressed through XML.

For import of XML or interaction with existing XML SNS may be necessary and useful but they should not be necessary in well formed data models.

References are considered harmful

References imply referential integrity. References are costly from both:

- the repository point of view, as it must manage the references
- the content point of view, as it impacts flexibility

It is much better to model those references as "weak-references" or simply use a path to refer to another object.

Files are Files are Files

If a content model exposes something that even remotely "smells" like a file or a folder, try to use nt:file or nt:folder.

IDs are evil

In relational databases IDs are a necessary means to express relations, so people tend to use them in content models as well.

If your content model is full of properties that end in "Id" you probably are not
leveraging the hierarchy properly
To learn more about David's model, go to:

http://dev.day.com/docs/en/cq/current/howto/model_data.html

Repository Internals

BlobStore (Jackrabbit 3.x) and DataStore (Jackrabbit 2.x)

The data store holds large binaries. On write, these are streamed directly to the data store and only an identifier referencing the binary is written. By providing this level of indirection, the data store ensures that large binaries are only stored once, even is they appear in multiple locations within the content in the PM store. In effect the data store is an implementation detail of the PM store. Like the PM, the data store can be configured to store its data in a file system (the default) or in a database.

The Oak BlobStore is an enhanced version of the Jackrabbit 2.x DataStore. The BlobStore can be used to store large binary values. For example, large binaries such as files, special treatment can improve performance and reduce disk usage. Other than the JCR 2.0 DataStore, the BlobStore splits binaries into blocks of 2 MB. This addresses the following issues in the JCR2.0 DataStore:

DataStore	BlobStore
A temporary file is created when adding a large binary, even if the binary already exists.	With Oak BlobStores, binaries are split into blocks of 2 MB. These blocks are processed in memory, so that temp files are never needed.
Sharding is slow and complicated because the hash needs to be calculated first, before the binary is stored in the target shard (the FileDataStore still doesn't support sharding the directory currently). Sharding is used to store data over multiple servers. Refer to the Oak chapter for more details.	Sharding for Oak BlobStores is trivial because each block is processed separately.
File handles or database streams are kept open until the consumer is done reading, which complicates the code, and we could potentially get too many open files or run out of db connections when the consumer doesn't close the stream.	File handles don't need to be kept open because the Oak BlobStore's blocks are processed in memory.

Currently, in the BlobStore, binaries that are similar and are stored separately except if some of the 2 MB blocks match. However, the algorithm in the BlobStore would allow re-using all matching parts, because in the BlobStore, concatenating blob ids means concatenating the data. All current Oak BlobStore implementations use the SHA-256 algorithm to compute hashes.

Note: Jackraabit 2.0 DataStores are supported in Oak.

Microkernels

In Oak, Microkernels replaces the Persistence Manager. The Oak Microkernel API provides an abstraction layer for the actual storage of the content. The Microkernel API is completely based on Strings. It uses the JSOP format, which is a lightweight HTTP protocol for manipulating JSON-based object models.

Currently, Oak has two main Microkernel implementations: DocumentMK and SegmentMK.

The MicroKernel Design Goals and Principles:

- Manage huge trees of nodes and properties efficiently.
- MVCC-based concurrency control (writers don't interfere with readers, snapshot isolation)
- GIT/SVN-inspired DAG-based versioning model

- Highly scalable concurrent read & write operations
- Session-less API (there's no concept of sessions; an implementation doesn't need to track/manage session state)
- Easy to remote
- Efficient support for large number of child nodes
- Integrated API for efficiently storing/retrieving large binaries
- Human-readable data serialization (JSON)

For more information on Microkernels, read the Oak chapter.

Journal

Journals are used if you are using SegmentMK for storing as the Microkernel. Journals are special, atomically updated documents that record the state of the repository as a sequence of references to successive root node records. As the system grows there is a need of more journals in it. They are linked to each other in a tree hierarchy. Commits can be done on every journal but in the end all commits will be merged back to the root journal. Temporary branches are also recorded as journals. For every branch there is one journal document. Journals can also represent root node references and in this case they are used as the starting point for garbage collection.

Query Index

Oak does not index content by default as does Jackrabbit 2. You need to create custom indexes when necessary, much like in traditional RDBMSs. If there is no index for a specific query, then the repository will be traversed. That is, the query will still work but probably be very slow.

When Oak is used with AEM6, the major indexes required for AEM are created automatically.

What it means for developers:

- When querying the repository with the JCR API's QueryManager, you are very likely to require an index for that query in order for the query to perform well.
- Check the log for the message " using traversal index". If you find this message, consider creating an index in the backend.

Internally, the query engine uses a cost based query optimizer that asks all the available query indexes for the estimated cost to process the query. It then uses the index with the lowest cost.
By default, the following indexes are available:

- A property index for each indexed property.
- A full-text index which is based on Apache Lucene / Solr.
- A node type index (which is based on an property index for the properties jcr:primaryType and jcr:mixins).

- A traversal index that iterates over a subtree.

If no index can efficiently process the filter condition, the nodes in the repository are traversed at the given subtree.
Usually, data is read from the index and repository while traversing over the query result. There are exceptions however, where all data is read in memory when the query is executed: when using a full-text index, and when using an "order by" clause.

For more information, refer to: http://jackrabbit.apache.org/oak/docs/query.html

Repository Configuration

In JCR 2.0, the configuration for the repository is defined in the repository.xml file. This repository.xml file contains global settings and a workspace configuration template. The template is instantiated to a workspace.

In Oak, this can be done using the ConfigMgr. (http://localhost:4502/system/console/configMgr)
Following configuration can be done:

- External Login module: Oak has a new concept of external login modules. JAAS login modules can be deployed as OSGi bundles with startup level 15. The external login module can be activated by Apache Jackrabbit Oak External Login Module. Configuration of the synchronization of the user data with the external login module is configured in section, Apache Jackrabbit Oak Default Sync Handler.

- LDAP Configuration: With this, you can configure LDAP. This configuration is available in the system console section, Apache Jackrabbit Oak LDAP Identity Provider.

Basic Content Access

The JCR provides very fast access by both path and ID. The underlying storage addressed by ID, but path traversal is also required for ACL checks. The caches optimized for a reasonably sized active working set. The typical web access pattern:

- a handful of key resources

- a long tail of less frequently accessed content

- few writes

Writing can result in a performance hit especially, when updating nodes with lots of child nodes.

Batch Processing

Batch processing presents two separate issues: read and write. When reading lots of content:

- Implement tree-traversal as the best approach, but this approach will flood the caches
- Schedule for off-peak times
- Add explicit delay (used by the garbage collectors)
- Use a dedicated cluster node for batch processing

When writing lots of content (including deleting large subtrees)

- Remember - the entire transient space is kept in memory and committed atomically
- Split the operation to smaller pieces
- Save after every ~1k nodes
- Leverage the data store if possible

CRX Search Features not specified by the JCR

- The following features are not specified by the JCR specification, but have been added to the CRX:
- Extract text from binary content
- Get a text excerpt with highlighted words that matched the query (ExcerptProvider)
- Search for a term and its synonyms: SynonymSearch
- Search for similar nodes: SimilaritySearch
- Define index aggregates, rules and scores: IndexingConfiguration
- Check spelling of a fulltext query statement: SpellChecker

Query Syntax

As specified, JSR 170: SQL and XPath: both syntaxes have same feature set. Since JSR 283: The Abstract Query Model (AQM) defines the structure and semantics of a query. The specification defines 2 language bindings for AQM:

- JCR-SQL2 (Grammar: http://www.h2database.com/jcr/grammar.html)
- JCR-JQOM

Basic AQM Concepts

A query has one or more selectors. When the query is evaluated, each selector independently selects a subset of the nodes in the workspace based on node type.

Joins transform the multiple sets of nodes selected by each selector into a single set. The *join* type can be inner, left-outer, or right-outer.

AQM Concepts - Constraints

A query can specify a constraint to filter the set of node-tuples. The constraints may be any combination of:

- Absolute or relative path:

  e.g., nodes that are children of /pictures

- Name of the node

- Value of a property:

  e.g., nodes whose jcr:created property is after 2007-03-14T00:00:00.000Z

- Length of a property:

  e.g., nodes whose jcr:data property is longer than 100 KB

- Existence of a property:

  e.g., nodes with a jcr:description property

- Full-text search:

  e.g., nodes which have a property that contains the phrase "beautiful sunset"

Search Basics

Defining and executing a JCR-level search requires the following logic:

1. Get the QueryManager for the Session/Workspace:

   ```
   QueryManager qm = session.getWorkspace().getQueryManager();
   ```

2. Create the query statement:

   Query q = `qm.createQuery`("select * from moviedb:movie order by Title",Query.SQL);

3. Execute the query:

QueryResult res = `q.execute();`

4. Iterate through the results:

```
NodeIterator nit = res.getNodes();
```

Query Examples - SQL2

Find all nt:folder nodes

```
SELECT * FROM [nt:folder]
```

NOTE: In reality, you would do this type of filter operation within the application, not with a search query.

Find all files under /var. Exclude files under /var/classes.

```
SELECT * FROM [nt:file] AS files
WHERE ISDESCENDANTNODE(files, [/var])
AND (NOT ISDESCENDANTNODE(files, [/var/classes]))
```

Find all files under /var (but not under /var/classes) created by existing users. Order results in ascending order by jcr:createdBy and jcr:created.

```
SELECT * FROM [nt:file] AS file
        INNER JOIN [rep:User] AS user ON file.[jcr:createdBy] =
user.[rep:principalName]
        WHERE ISDESCENDANTNODE(file, [/var])
                AND (NOT ISDESCENDANTNODE(file, [/var/classes]))
        ORDER BY file.[jcr:createdBy], file.[jcr:created]
```

Java Query Object Model (JQOM)

A mapping of AQM to Java API. JQOM expresses a query as a tree of Java objects. The package ***javax.jcr.query.qom*** defines the API.

A query is built using QueryObjectModelFactory and its createQuery method. For example:

```
QueryManager qm = session.getWorkspace().getQueryManager();
QueryObjectModelFactory qf = qm.getQOMFactory();
QueryObjectModel query = qf.createQuery( ... );
```

JQOM Examples

Find all nt:folder nodes.

```
QueryObjectModelFactory qf = qm.getQOMFactory();
Source source = qf.selector("nt:folder", "ntfolder");
QueryObjectModel query = qf.createQuery(source, null, null,null);
```

Find all files under /var. Exclude files under /var/classes.

```
QueryObjectModelFactory qf = qm.getQOMFactory();
Source source = qf.selector("nt:file", "files");
Constraint pathConstraint = qf.and(qf.descendantNode("files",
"/var"), qf.not(qf.descendantNode("files", "/var/classes")));
QueryObjectModel query = qf.createQuery(source,
pathConstraint, null,null);
```

Find all files under /var (but not under /var/classes) created by existing users. Order results in ascending order by jcr:createdBy and jcr:created.

```
QueryObjectModelFactory qf = qm.getQOMFactory();
Source source = qf.join(qf.selector("nt:file", "file"),
qf.selector("rep:User", "user"),
QueryObjectModelFactory.JCR_JOIN_TYPE_INNER,qf.
equiJoinCondition("file","jcr:createdBy", "user","rep:principalName"));
Constraint pathConstraint = qf.and(qf.descendantNode("file", "/var"),
qf.not(qf.
descendantNode("file", "/var/classes")));
Ordering orderings[] = {qf.ascending(qf.propertyValue("file",
"jcr:createdBy")),
qf.ascending(qf.propertyValue("file", "jcr:created")) };
QueryObjectModel query = qf.createQuery(source, pathConstraint,
orderings,
null);
```

Search Performance

Which JCR search methodologies are the fastest?

- Constraints on properties, node types, full text
- Typically O(n) where n is the number of results, vs. the total number of nodes

Which JCR search methodologies are pretty fast?

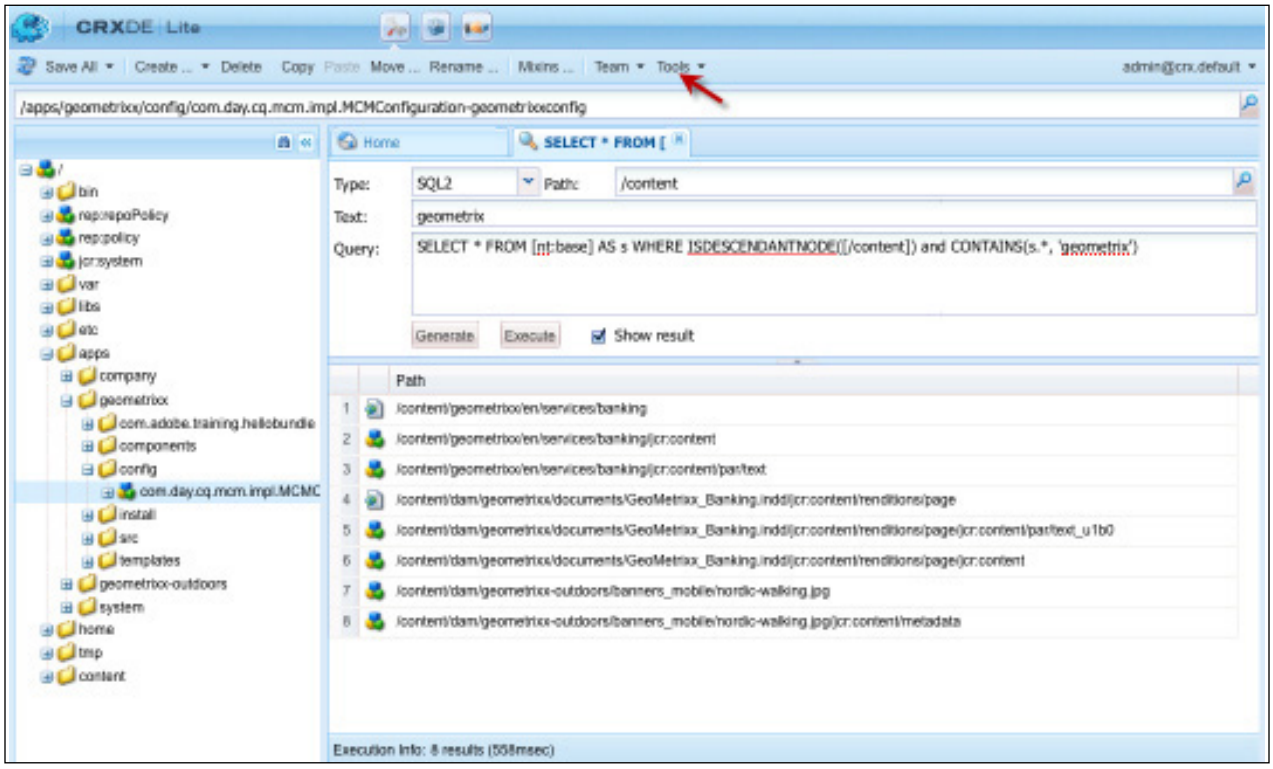
- Path constraints

Which JCR methodologies need some planning?

- Constraints on the child axis
- Sorting, limit/offset
- Joins

Testing Queries

You can test your queries using CRXDE Lite. Access the Query Tool in the Tools menu from the top toolbar in CRXDE Lite.



EXERCISE 6.1 - Debugging and Logging Queries

The CRX provides a mechanism to log query execution information. It can be done in the Adobe AEM Web Console.

Steps

1. Open the Adobe AEM Web Console. (http://localhost:4502/system/console)
2. Open the Configuration tab
3. Create a new Apache Sling Logging Logger configuration, then apply the following values:

Log Level: Debug

Log File: logs/query.log

Logger: org.apache.jackrabbit.oak.jcr.query

4. Click on the "+" to add a new logger class

   Logger: org.apache.jackrabbit.oak.jcr.query.QueryResultImpl

5. Then click Save. The new Logging Logger will direct its output to a new log file named *query.log.*

6. Update the ROOT logger configuration as follows:

   Log File: logs/query.log

   Number of Log files: 5

   Log File Threshold: '.'yyyy-MM-dd

7. Then click **Save**. The new Logging Writer will control the number of files that exist for the query.log before they begin to roll (overwrite).

EXERCISE 6.2 – Search

Now that we are logging the query classes, it is time to create a servlet that implements a Sling selector. The servlet will perform a full text search starting with the specified node. The result set will be a set of pages, but we will specify node type nt:unstructured, because we know that the jcr:content child of each page is of type nt:unstructured.

Steps

1. In the company-core project, enter the following code to create a servlet, named SearchServlet, that responds to resource type *geometrixx/components/ homepage* and selector "search".

```
package com.adobe.training.core;

import java.io.IOException; 34

import javax.jcr.Node;
import javax.jcr.NodeIterator;
import javax.jcr.RepositoryException;
import javax.jcr.ValueFactory;
import javax.jcr.query.Query;
import javax.jcr.query.QueryManager;
import javax.jcr.query.qom.Constraint;
import javax.jcr.query.qom.QueryObjectModel;
import javax.jcr.query.qom.QueryObjectModelFactory;
import javax.jcr.query.qom.Selector;
```

```
import javax.servlet.ServletException;

import org.apache.felix.scr.annotations.sling.SlingServlet;
import org.apache.sling.api.SlingHttpServletRequest;
import org.apache.sling.api.SlingHttpServletResponse;
import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
import org.apache.sling.commons.json.JSONArray;
import org.apache.sling.commons.json.JSONObject;

import com.day.cq.wcm.api.PageManager;

@SlingServlet(resourceTypes = "geometrixx/components/homepage", selectors = public class
SearchServlet extends SlingSafeMethodsServlet {

 /**
 *
 */
 private static final long serialVersionUID = 3169795937693969416L;
```

2. Now add the following code to the Search Servlet to implement the GET method that writes search results into JSONArray.

```
@Override
public final void doGet(final SlingHttpServletRequest request, final SlingHttpServletResponse 3 throws ServletException, IOException {
  response.setHeader("Content-Type", "application/json");
  JSONObject jsonObject = new JSONObject();
  JSONArray resultArray = new JSONArray();

  try {
    // this is the current node that is requested, in case of a page that is the jcr:content node
  Node currentNode = request.getResource().adaptTo(Node.class);
   PageManager pageManager = request.getResource().getResourceResolver();
   // node that is the cq:Page containing the requested node
   Node queryRoot = pageManager.getContainingPage(currentNode.getPath()).adaptTo(Node.class);

   String queryTerm = request.getParameter("q");
   if (queryTerm != null) {
     NodeIterator searchResults = performSearch(queryRoot, queryTerm);
     while (searchResults.hasNext()) resultArray.put(searchResults.nextNode().getPath()) ;
     jsonObject.put("results", resultArray);
   }
  }
  catch (Exception e) {
   e.printStackTrace();
  }
// response.getWriter().print("SQL VERSION");
  response.getWriter().print( jsonObject.toString());
 }
```

3. Implement actual search method.

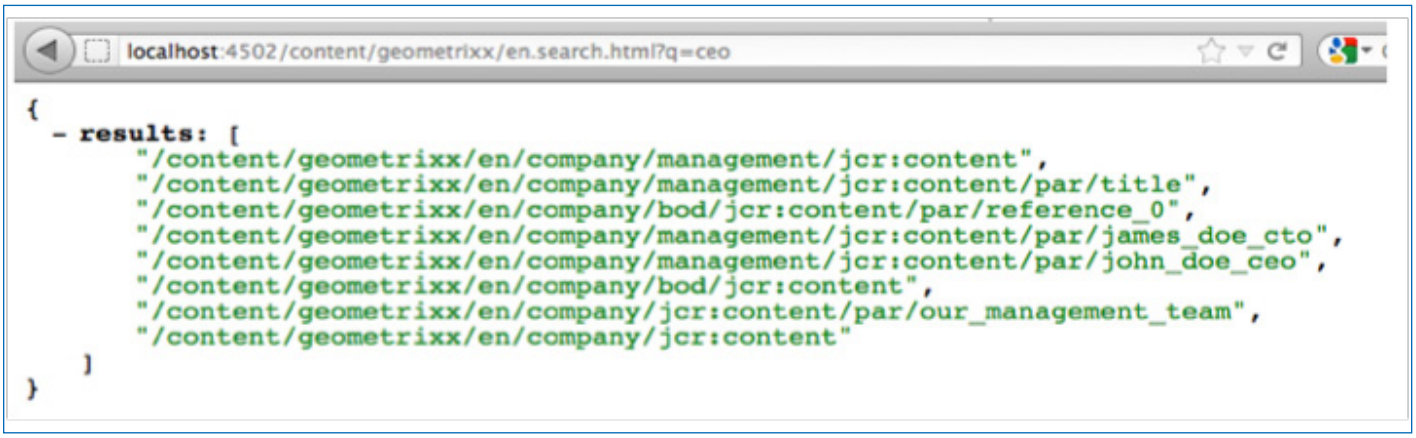
```
private NodeIterator performSearch(Node queryRoot, String queryTerm) throws RepositoryException {

        // JQOM infrastructure
        QueryObjectModelFactory qf = queryRoot.getSession().getWorkspace().getQueryManager().getQOMFactory();
        ValueFactory vf = queryRoot.getSession().getValueFactory();

        final String SELECTOR_NAME = "all results";
        final String SELECTOR_NT_UNSTRUCTURED = "nt:unstructured";
        // select all unstructured nodes
        Selector selector = qf.selector(SELECTOR_NT_UNSTRUCTURED, SELECTOR_NAME);

        // full text constraint
        Constraint constraint = qf.fullTextSearch(SELECTOR_NAME, null, qf.literal(vf.createValue(queryTerm)));
        // path constraint
        constraint = qf.and(constraint, qf.descendantNode(SELECTOR_NAME, queryRoot.getPath()));

        // execute the query without explicit order and columns
        QueryObjectModel query = qf.createQuery(selector, constraint, null, null);
        return query.execute().getNodes();

}
```

Note: Repository path "content/geometrixx/en/jcr:content" contains sling:resourceType "geometrixx/components/homepage" which matches the resource type of this servlet. The selector "search" causes the servlet to be chosen by sling. In fact any URI that matches a repository path that has this sling resource type would give the same result, as long as the selector is set to search.

4. Build and deploy the bundle.

5. Results are available at, for example

   http://localhost:4502/content/geometrixx/en.search.html?q=ceo



6. Implement same functionality using SQL2

```
private NodeIterator performSearchWithSQL(Node queryRoot, String queryTerm) throws
RepositoryException {

        QueryManager qm = queryRoot.getSession().getWorkspace().getQueryManager();

        Query query = qm.createQuery("SELECT * FROM [nt:unstructured] AS node WHERE
ISDESCENDANTNODE([" + queryRoot.getPath() + "]) and CONTAINS(node.*, '" + queryTerm + "')",
Query.JCR_SQL2);

        return query.execute().getNodes();
}
```

Congratulations! You have written a servlet that implements a Sling selector, performs a full text search, and returns the query results as JSON.

07

JCR Versioning, Observation

Observation

A repository may support *observation*, which enables an application to receive notification of persistent changes to a workspace. JCR defines a general event model and specific APIs for asynchronous and journaled observation. A repository may support asynchronous observation, journaled observation or both. Whether an implementation supports asynchronous or journaled observation can be determined by querying the repository descriptor table with the keys.

`Repository.OPTION _ OBSERVATION _ SUPPORTED`

or

`Repository.OPTION _ JOURNALED _ OBSERVATION _ SUPPORTED`

A return value of true indicates support.(see *Servlets, getting the repository descriptor via a DOM object*).

Event Model

A persisted change to a workspace is represented by a set of one or more *events*. Each event reports a single simple change to the structure of the persistent workspace in terms of an item added, changed, moved or removed. The six standard event types are (see JCR API):

NODE_ADDED, NODE_MOVED, NODE_REMOVED, PROPERTY_ADDED, PROPERTY_REMOVED and PROPERTY_CHANGED.

A seventh event type, PERSIST, may also appear in certain circumstances (see *Event Bundling in Journaled Observation*).

Scope of Event Reporting

The scope of event reporting is implementation-dependent. An implementation should make a *best-effort* attempt to report all events, but may exclude events if reporting them would be impractical given implementation or resource limitations. For example, on an import, move or remove of a subgraph containing a large number of items, an implementation may choose to report only events associated with the root node of the affected graph and not those for every subitem in the structure.

The Event Object

Each event generated by the repository is represented by an Event object.

Event Types

The type of an Event is retrieved through `int Event.getType()` which returns one of the int constants found in the Event interface: NODE_ADDED, NODE_MOVED, NODE_REMOVED, PROPERTY_ADDED, PROPERTY_REMOVED, PROPERTY_CHANGED or PERSIST.

Event Information

Each Event is associated with a path, an identifier and an information map, the interpretation of which depend upon the event type.

The event path is retrieved through:

```
String Event.getPath(),
```

the identifier through:

```
StringEvent.getIdentifier()
```

and the information map through:

```
java.util.Map Event.getInfo()
```

If the event is a NODE_ADDED or NODE_REMOVED then,

- Event.getPath() returns the absolute path of the node that was added or removed.
- Event.getIdentifier() returns the identifier of the node that was added or removed.
- Event.getInfo()returnsanemptyMapobject.

If the event is NODE_MOVED then,

- Event.getPath() returns the absolute path of the *destination* of the move.

- Event.getIdentifier() returns the identifier of the node that was moved.
- Event.getInfo() returns a Map containing parameters information from the method that caused the event.

If the event is a PROPERTY_ADDED, PROPERTY_CHANGED or PROPERTY_REMOVED then,

- Event.getPath() returns the absolute path of the property that was added, changed or removed.
- Event.getIdentifier() returns the identifier of the parent node of the property that was added, changed or removed.
- Event.getInfo()returnsanemptyMapobject.

If the event is a PERSIST (see *§12.6.3 Event Bundling in Journaled Observation*) then Event.getPath() and Event.getIdentifier() return null and Event.getInfo() returns an empty Map.

Externally Caused NODE_MOVED Event

In a repository that reports events caused by mechanisms external to JCR, the keys and values found in the information map returned on a NODE_MOVED are implementation-dependent.

User ID

An Event also records the identity of the Session that caused it:

- String Event.getUserID() returns the user ID of the Session, which is the same value that is returned by Session.getUserID()

User Data

An Event may also contain arbitrary string data specific to the session that caused the event. A session may set its current user data using:

```
void ObservationManager.setUserData(Sting userData)
```

Typically a session will set this value in order to provide information about its current state or activity. A process responding to these events will then be able to access this information through:

```
String Event.getUserData()
```

and use the retrieved data to provide additional context for the event, beyond that provided by the identify of the causing session alone.

Event Date

An event also records the time of the change that caused it. This acquired through:

```
Long Event.getDate()
```

The date is represented as a millisecond value that is an offset from the epoch January 1, 1970 00:00:00.000 GMT (Gregorian).

Event Bundling

A repository that supports observation *may* support event bundling under asynchronous observation, journaled observation, or both.

In such a repository, events are produced in bundles where each corresponds to a single atomic change to a persistent workspace and contains only events caused by that change.

Event Ordering

In both asynchronous and journaled observation the order of events within a bundle and the order of event bundles is not guaranteed to correspond to the order of the operations that produced them.

Asynchronous Observation

Asynchronous observation enables an application to respond to changes made in a workspace as they occur.

An application connects with the asynchronous observation mechanism by registering an event listener with the workspace. Listeners apply *per workspace*, not repository-wide; they only receive events for the workspace in which they are registered. An event listener is an application-specific class implementing the EventListener interface that responds to the stream of events to which it has been subscribed.

This observation mechanism is *asynchronous* in that the operation that causes an event to be dispatched does not wait for a response to the event from the listener; execution continues normally on the thread that performed the operation.

Observation Manager

Registration of event listeners is done through the ObservationManager object acquired from the Workspace through:

```
ObservationManager Workspace.getObservationManager().
```

Adding an Event Listener

An event listener is added to a workspace with:

```
void ObservationManager. addEventListener(
        EventListener listener,
        int eventTypes,
        String absPath,
        boolean isDeep,
        String[] uuid,
        String[] nodeTypeName,
        boolean noLocal)
```

The EventListener object passed is provided by the application. As defined by the EventListener interface, this class must provide an implementation of the onEvent method:

```
void EventListener.onEvent(EventIterator events)
```

When an event occurs that falls within the scope of the listener, the repository calls the onEvent method invoking the application-specific logic that processes the event.

Event Filtering

In the upcomming JSR-333 specification, the event filtering is done through an EventFilter object passed to the ObservationManager at registering time.

The EventFilterObject provides the eventTypes, absPath, isDeep, Identifiers, NodeTypes and Nolocal values as parameters of the class.

In JSR-283 you have to deal "manually" with those objects and not through the EventFilter.

Which events a listener receives are determined as follows.

Access Privileges

An event listener will only receive events for which its Session (the Session associated with the ObservationManager through which the listener was added) has sufficient access privileges.

Event Types

An event listener will only receive events of the types specified by the eventTypes parameter of the addEventListener method. The eventTypes parameter is an int composed of the bitwise AND of the desired event type constants.

Local and Nonlocal

If the noLocal parameter is true, then events generated by the Session through which the listener was registered are ignored.

Node Characteristics

Node characteristic restrictions on an event are stated in terms of the associated parent node of the event. The *associated parent node* of an event is the *parent* node of the item at (or formerly at) the path returned by `Event.getPath()`.

Location

If isDeep is false, only events whose associated parent node is at absPath will be received.
If isDeep is true, only events whose associated parent node is at or below absPath will be received.

It is permissible to register a listener for a path where no node currently exists.

Identifier

Only events whose associated parent node has one of the identifiers in the uuid String array will be received. If this parameter is null then no identifier-related restriction is placed on events received.

NOTE: Specifying an empty array instead of null, results in no nodes being listened to.

Node Type

Only events whose associated parent node is of one of the node types String array will be received. If this parameter is null then no node type- related restriction is placed on events received.

Re-registration of Event Listeners

The filters of an already-registered EventListener can be changed at runtime by re-registering the same EventListener Java object with a new set of filter arguments. The implementation must ensure that no events are lost during the changeover.

Event Iterator

In asynchronous observation the EventIterator holds an event bundle or a single event, if bundles are not supported. EventIterator inherits the methods of RangeIterator and adds an Event-specific next method:

```
Event EventIterator.nextEvent()
```

Listing Event Listeners

Retrieved via:

```
EventListenerIterator ObservationManager.
getRegisteredEventListeners()
```

EventListenerIterator

Methods that return a set of EventListener objects (such as ObservationManager. getRegisteredEventListeners) do so using an EventListenerIterator. The EventListenerIterator class inherits the methods of RangeIterator and adds an EventListener-specific next method:

```
EventListener EventListenerIterator.nextEventListener()
```

Removing Event Listeners

Through: `void ObservationManager.removeEventListener(EventListener listener)`

User Data

Through: `void ObservationManager.setUserData(StringuserData)`

Journaled Observation

Journaled observation allows an application to periodically connect to the repository and receive a report of changes that have occurred since some specified point in the past (for example, since the last connection). Whether a repository records a per-workspace event journal is up to the implementation's configuration.

Event Journal

The EventJournal of a workspace instance is acquired by calling either

```
EventJournal ObservationManager.getEventJournal()
```

Or

```
EventJournal getEventJournal(
        int eventTypes,
        String absPath,
        boolean isDeep,
        String[] uuid,
        String[] nodeTypeName,
        Boolean noLocal).
```

Events reported by this EventJournal instance will be filtered according to the current session's access rights, any additional restrictions specified through implementation-specific configuration and, in the case of the second signature, by the parameters of the methods. These parameters are interpreted in the same way as in the method addEventListener.

An EventJournal is an extension of EventIterator that provides the additional method skipTo(Calendar date).

```
void EventJournal.skipTo(Calendar date)
```

Journaling Configuration

An implementation is free to limit the scope of journaling both in terms of coverage (that is, which parts of a workspace may be observed and which events are reported) and in terms of time and storage space. For example, a repository can limit the size of a journal log by stopping recording after it has reached a certain size, or by recording only the tail of the log (deleting the earliest event when a new one arrives). Any such mechanisms are assumed to be within the scope of implementation configuration.

Event Bundling in Journaled Observation

In journaled observation dispatching is done by the implementation writing to the event journal.

If event bundling is supported a PERSIST event is dispatched when a persistent change is made to workspace bracketing the set of events associated with that change. This exposes event bundle boundaries in the event journal.

Note that a PERSIST event will never appear within an EventIterator since, in asynchronous observation, the iterator itself serves to define the event bundle. In repositories that do not support event bundling, PERSIST events do not appear in the event journal.

Importing Content

Whether events are generated for each node and property addition that occurs when content is imported into a workspace is left up to the implementation.

Exceptions

The method EventListener.onEvent does not specify a throws clause. This does not prevent a listener from throwing a RuntimeException, although any listener that does should be considered to be in error.

Event topics used on AEM-level

In AEM, event topics include Replication and PageEvents, e.g.

- com.day.cq.wcm.api.PageEvent
- com.day.cq.replication.ReplicationAction

Those classes use internally the observation mechanism specified above.

EXERCISE 7.1 - Observation Listener

Goal

We are going to write an observation listener that checks new or modified properties named "jcr:title". (Note that the title of a page is stored in the property jcr:title under the jcr:content node).

If the property does not have an exclamation mark ("!") at the end, the listener shall add one

Steps

1. Create the TitlePropertyListener component and the OSGi infrastructure. The component shall register itself as the listener on activation (and make sure it unregisters on deactivation so that you do not get a memory leak).

```
package com.adobe.training,core;

import javax.jcr.Property;
import javax.jcr.RepositoryException;
import javax.jcr.Session;
import javax.jcr.observation.Event
import javax.jcr.observation.EventIterator;
import javax.jcr.observation.EventListener;
import javax.jcr.observation.ObservationManager;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.sling.jcr.api.SlingRepository;
import org.osgi.service.component.ComponentContext;
import org.slf4j.Logger;
import org.slf4j.LoggerFactor;

@Component
public class TitlePropertyListener implements EventListener {
        private final Logger LOGGER = LoggerFactory.getLogger(TitlePropertyListener.class);
```

```
        @Reference
        private SlingRepository repository;
        private Session session;
        private ObservationManager observationManager;
        protected void activate(ComponentContext context) throws Exception {
                session = repository.loginadministrative(null);
                observationManager = session getWorkspace().getObservationManager();

                observationManager.addEventListener(this, Event.PROPERTY_ADDED | Event.
PROPERTY_CHANGED,"/", true, null,
                        null, true);
                LOGGER.info('********added JCR event listener");
}
protected void deactivate(ComponentContext componentContext) {
                try {
                        if (observationManager !=null) {
                                observationManager.removeEventListener(this);
                                LOGGER info("********removed JCR event listener");
                        }
                }
                catch (RepositoryException re) {
                        LOGGER.error("********error removing the JCR event listener", re);
                }
                finally{
                        if (session != null){
                                session.logout();
                                session = null;
                        }
                }
```

Note: The method call addEventListener: Event types are bitwise OR'ed. The last parameter (noLocal) prevents changes done by this session to be sent to the listener (which would result in an endless loop in this case). You could also register for events sent by specific node types or at specific paths only. (see Event class on the JCR API).

2. Implement the onEvent method of the EventListener interface:

```
public void onEvent(EventIterator it) {
        while (it.hasNext()) {
          Event event = it.nextEvent();
                try {
                  LOGGER.info("********new property event: {}", event.getPath());
                  Property changedProperty = session.getProperty(event.getPath());
                  if (changedProperty.getName().equalsIgnoreCase("jcr:title")
                   && !changedProperty.getString().endsWith("!")) {
                  changedProperty.setValue(changedProperty.getString() + "!");
                  session.save();
                  }
          }
                catch (Exception e) {
                        LOGGER.error(e.getMessage(), e);
                }
        }
}
```

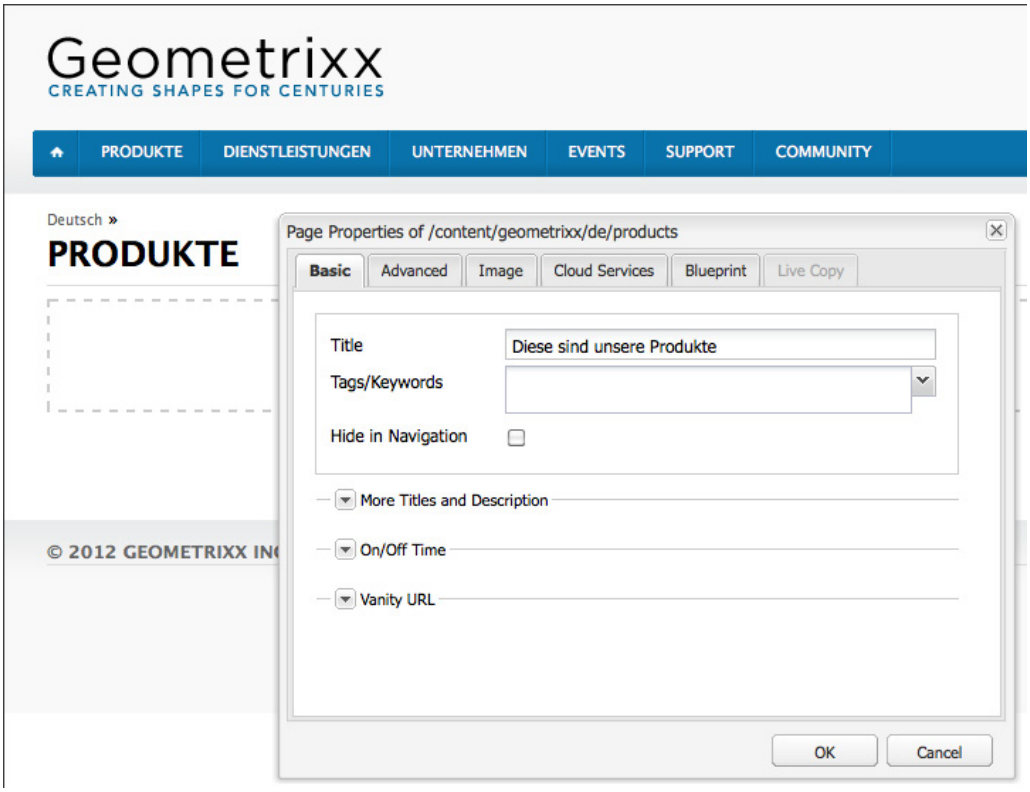
3. Deploy the component by executing:

```
mvn clean install -P bundle
```

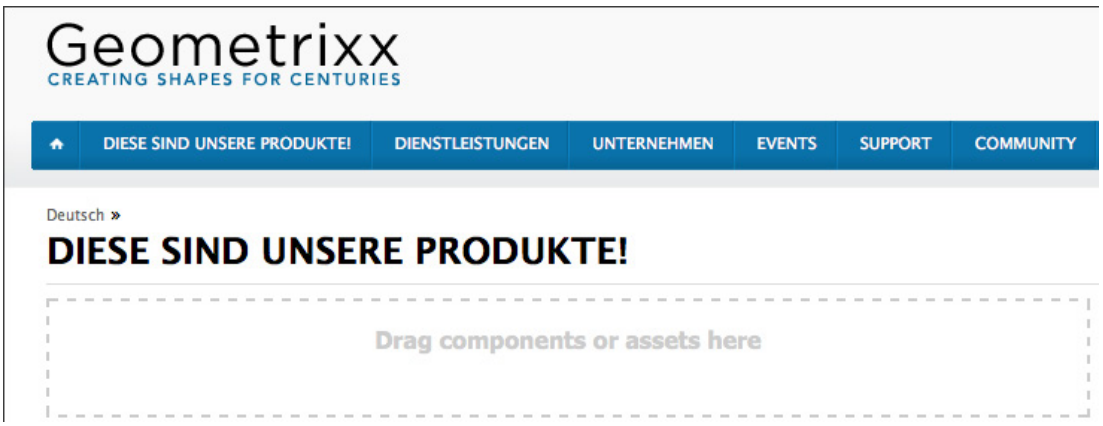
4. Check that the component is installed:

5. Change a property called "jcr:title" in CRXDE Lite. Refresh to see the added "!".

6. You can also change any page's title in site admin, and see that the page's title is added with an "!"



7. Click ok



08

Jackrabbit Oak

Jackrabbit Oak is an effort to implement a scalable and efficient hierarchical content repository for use as the foundation of modern, efficient web sites, and other demanding content applications.

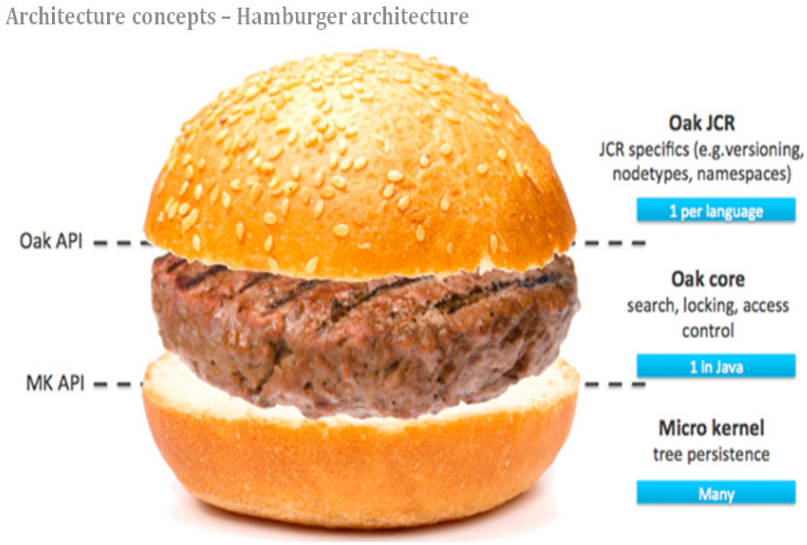
Jackrabbit Oak implements the Java Content Repository (JCR) spec. The most used parts of JSR-283 are implemented, but Jackrabbit Oak does not aim to be a reference implementation of JSR-283. AEM 6.0 is built on top of Jackrabbit Oak.

Following are the goals of Jackrabbit Oak:

- Scalability: Oak supports big repositories and distributed repository with many cluster nodes.
- Improved throughput: Parallel write operations are possible using Oak.
- Oak Supports large number of child nodes.

Oak Architecture (also known as Hamburger Architecture)

The repository implements standards like JCR, WebDAV, and CMIS, and are easily accessible from various platforms, especially from JavaScript clients running in modern browser environments. The implementation provides more out-of-the-box functionalities than typical NoSQL databases while achieving comparable levels of scalability and performance.



The Burger's Top Bun: Oak JCR

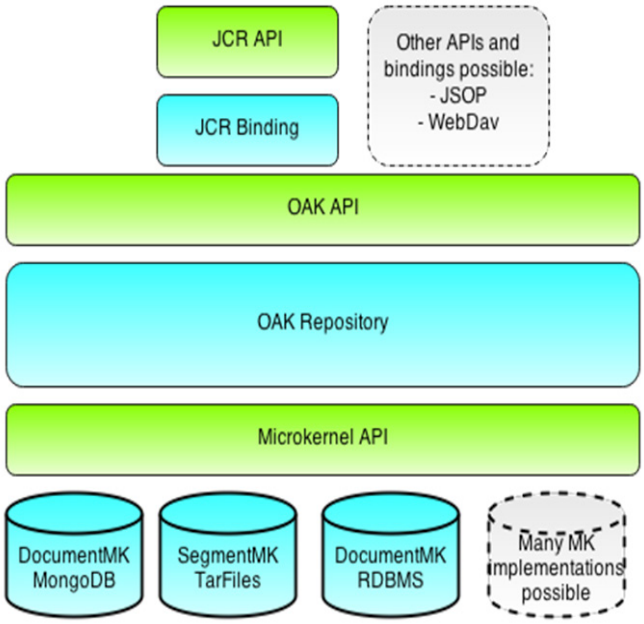
- Oak Implements the JCR API

The Burger's Patty: Oak Core

- Where, most of the heavy lifting takes place
- Adds to MK's tree model (ACLs, Search, and Indexing)
- Observation
- Exposes essentially a decorated tree model
- Mostly transforms JCR semantics into tree operations
- Also contains, "Commit hooks" that implement JCR constraints, e.g. node types
- It is now implemented for JCR. However, non-Java implementations are possible and are part of the concept

The Burger's Bottom Bun: Microkernel

- Implements a tree model (Nodes and Properties)
- Exposes Microkernel API



Microkernels

The Oak Microkernel API provides an abstraction layer for the actual storage of the content.
The Microkernel API is completely based on strings. It uses the JSOP format, which is a lightweight HTTP protocol for manipulating JSON-based object models.
Currently, Oak has two main Microkernel implementations: DocumentMK and SegmentMK.

The MicroKernel Design Goals and Principles:

- Manage huge trees of nodes and properties efficiently
- MVCC-based concurrency control (writers do not interfere with readers; snapshot isolation)
- GIT/SVN-inspired DAG-based versioning model
- Highly scalable concurrent read & write operations
- Session-less API (there is no concept of sessions; an implementation does not need to track/ manage session state)
- Easy to remote
- Efficient support for large number of child nodes
- Integrated API for efficiently storing or retrieving large binaries
- Human-readable data serialization (JSON)

The MicroKernel Data Model:

- Simple JSON-inspired data model: just nodes and properties
- A node consists of an unordered set of name to item mappings. Each property and child node is uniquely named and a single name can only refer to a property or a child node, not both at the same time.
- Properties are represented as name/value pairs
- Supported property types: string, number, Boolean, array
- A property value is stored and used as an opaque, unparsed character sequence

DocumentMK

The Oak Document Microkernel manages JCR content by storing each content node in a separate document.

Implementation

DocumentMK is a concept and is not something that is directly implemented. Currently, there are two relevant implementations of a DocumentMK:

- MongoMK: Based on MongoDB, this is currently the only DocumentMK implementation supported by AEM6
- RDBMK: Based on a relational database, currently not supported for AEM6

The discussion in the following sections is based on the MongoMK implementation because it is currently more mature than the RDBMK. While the concepts should apply for all implementations of DocumentMK, it is possible that some points are implemented differently in RDBMK.

Documents

Documents in a DocumentMK are usually stored as JSON. A sample JSON-representation of a document in DocumentMK is:

```
{
    "_deleted" : {
        "r13f3875b5d1-0-1" : "false"
    },
    "_id" : "1:/node",
    "_lastRev" : {
        "r0-0-1" : "r13f38818ab6-0-1"
    },
    "_modified" : NumberLong(274208516),
    "_modCount" : NumberLong(2),
    "_revisions" : {
        "r13f3875b5d1-0-1" : "c",
        "r13f38818ab6-0-1" : "c"
    },
    "prop" : {
        "r13f38818ab6-0-1" : "\"foo\""
    }
}
```

Document nodes have two types of fields:

- Simple fields are stored as key/value pairs, in the example above, _id, _modified and _ modCount are simple fields.
- Versioned fields are kept in sub-documents where the key is a revision paired with the value of this revision.

DocumentMK documents will be treated in detail in section "Storage of Oak Documents in MongoDB."

Revisions

After each "commit" operation on the MicroKernel, a new revision is created.
In MongoMK, a revision is a string that consists of three pairs:

- A time stamp derived from the system time of the machine it was generated on
- A counter to distinguish revisions created with the same time stamp
- The cluster node id where the revision was created

Branches

MicroKernel implementation supports branches, which allows client to stage multiple commits and make them visible with a single merge call.

Previous Documents

DocumentMK adds data to a document with every modification but it never deletes any data (unless a cleanup is explicitly triggered). Old data is moved when there are 1000 commits to be moved or the document is bigger than 1 MB. Previous documents only contain immutable data, which means they only contain committed and merged revisions.

Background operations

Each DocumentMK instance connecting to same database in Mongo server performs certain background task.

- Renew Cluster Id Lease
- Background Document Split
- Background Writes
- Background Reads

Cluster node metadata

Cluster node metadata is stored in the clusterNodes collection. There is one entry for each cluster node that is running, and there are entries for cluster nodes that were ran. Old entries are kept so if a cluster node is started again, it gets the same cluster node id as before.

SegmentMK

SegmentMK is an Oak storage backend that stores content as various types of records within larger segments. One or more journals are used to track the latest state of the repository. These are the principles on which SegmentMK is designed:

- Immutability—the segments are immutable and because of that it is easy to cache frequently accessed segments. This feature simplifies backups.
- Compactness—the size optimized is in a way that it reduces the IO costs and it fits content in caches as much as possible.
- Locality—one segment usually stores related records. With this, the cache misses are avoided; for example, if a client wants to access more related nodes per session.

Implementation

The supported (in AEM6) persistence mechanism of the SegmentMK is currently the TarMK, which is based on tar files in the local files system.

Segments

One segment usually contains a continuous subset of a content tree; for example, node with its properties and closest child nodes. One segment is defined by UUID and can be up to 256 KB. Each segment keeps a list of the UUIDs of all other segments it references. The UUID contains data about the type of the segment like in this example:

`xxxxxxxx-xxxx-4xxx-Axxx-xxxxxxxxxxxx` : data segment UUID
`xxxxxxxx-xxxx-4xxx-Bxxx-xxxxxxxxxxxx` : bulk segment UUID

As you can see from the example, there are two types of segments—data and bulk segments. Data segments can contain any types of records and may refer to content in other segments. Bulk segments are only used for storing large binary values and they can contain only raw binary data, interpreted as a sequence of block records.

Journals

Journals are special, atomically updated documents that record the state of the repository as a sequence of references to successive root node records. As the system grows, there is a need of more journals in it. They are linked to each other in a tree hierarchy. Commits can be done on every journal, but in the end all commits are merged back to the root journal. Temporary branches are also recorded as journals. For every branch, there is one journal document. Journals can also represent root node references and in this case, they are used as the starting point for garbage collection.

Records

Records are part of a segment and there can be different types of them—blocks, lists, maps, values, templates, and nodes. Let us describe these types of records:

- Block records—Blocks are binary records of up to 4 kB and they don't contain any references to other records. That is why they can be easily identified.
- List records—List records represent a hierarchically stored immutable list and are used for storing arrays of values for multi-valued properties and sequences of blocks for large binary values.
- Map records—Map records represent a hierarchically stored immutable map. They store unordered sets of key-value pairs of record references and are used for nodes with a large number of properties or child nodes.

- Value records—Value records are byte arrays used for storing all names and values of the content tree. Depending on the data they hold, they can be small, medium, long, and external. The type of value record is encoded in the high-order bits of the first byte of the record.

- Template records—A template record describes the common structure of a family of related nodes. It contains sets of property name and type pairs. Additionally, it holds information whether the node has zero, one, or many child nodes.

- Node records—Node records hold the actual content structure of the repository. Node record consists of a template reference followed by property value references and child node entries.

Binary Data Stores

In Oak, the binary data can be stored independently from the content nodes.
The most important data store implementations are:

- Tar data store: binary data is stored in tar files on the file system

- MongoDB data store: binary data is stored in MongoDB

- S3 data store: binary data is stored in amazon cloud

- RDB data store: binary data is stored in a relational database

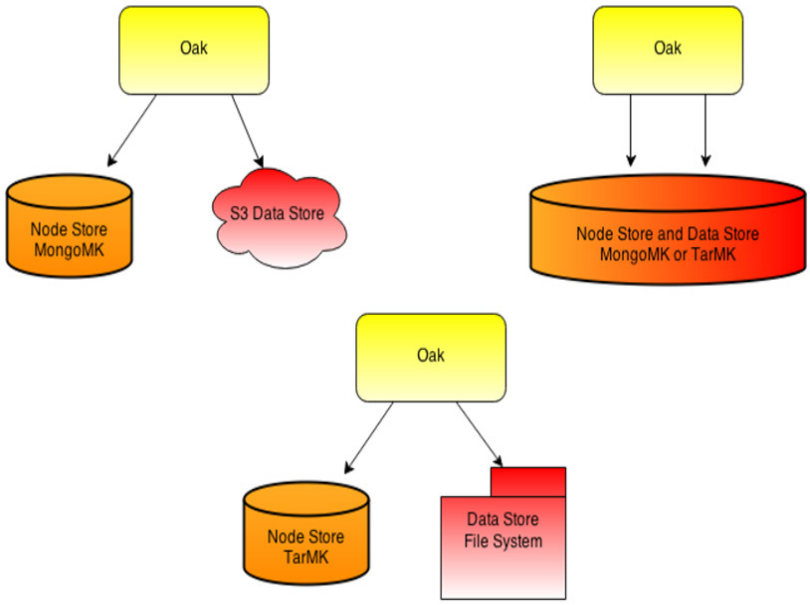
- File System: legacy implementation using the JCR2 DataStore

DataStores and MikroKernels

Not all combinations of data stores and node stores (Microkernels) are supported in AEM6. The following matrix shows the supported combinations:

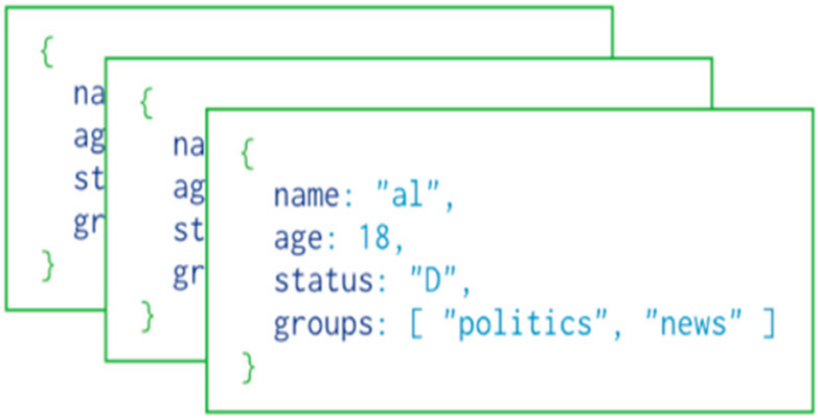
Node Store / Data Store	File System	S3	MongoDB	Tar	RDB
TarMK	OK	OK		OK	
MongoMK	OK	OK	OK		
RDB MK	OK				OK

The following image shows a selection of possible combinations of Microkernels and data stores. Note that the node and data store can be stored in the same backend, but this is not required.



Introduction to MongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling. MongoDB stores data in the form of documents, which are JSON-like field and value pairs. Formally, MongoDB documents are BSON documents, which is a binary representation of JSON with additional type information. The values of the fields in a document can be of any BSON data types, including other documents, arrays, and arrays of documents. The field names are strings and they can't contain null characters, dots, or dollar signs. The maximum BSON document size is 16 MB. MongoDB stores all documents in collections. A collection is a group of related documents that have a set of shared common indexes.



Basic operations in MongoDB

Read operations (queries)

Read operations are used for retrieving data, which is stored in the database. Queries can specify criteria for identifying documents and they can also use projections that limit the returned data. The most common reading method for MongoDB is `find()` method.

```
db.users.find(                          <-- collection
   { age: { $gt: 18 } },                <-- query criteria
   { name: 1, address: 1 }              <-- projection
).limit(5)                              <-- cursor modifier
```

In this example, it is assumed that, there is a collection in the database named users and this query will return "name", "address" and "_id" fields of at most 5 matching documents, which match the condition "age>18". The most important thing to know about projections is that the "_id" field is

always included in results unless explicitly excluded. Except from excluding the _id field in inclusive projections, you cannot mix exclusive and inclusive projections.

```
db.users.find( { age: { $gt: 18 } } ).sort( {age: 1 } )
```

The second example has the same criteria as the first but it shows how you can use sorting in the queries. Here, the results are sorted by "age" in ascending order.

The `find()` method in the mongo shell returns a cursor to the returning documents. The shell displays only the first 20 results. You can iterate over the next set of results with the command it. If you want to write an iterating function, you have to assign the cursor to a variable, and then create a loop:

```
var a = db.users.find()
while ( a.hasNext() )  printjson( a.next )
```

You can retrieve one document in two ways:

```
printjson( a [1] )
db.users.findOne()
```

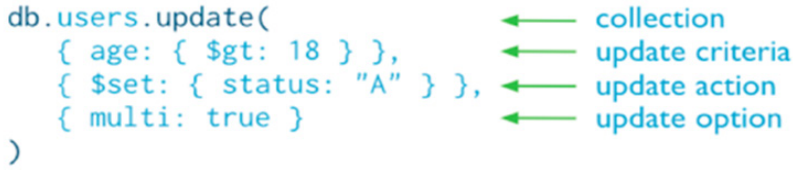
By default, the cursor is closed by the server if it is inactive for 10 minutes, or if the client has exhausted the cursor. To override this behavior, you can set noTimeout flag.

```
var myCursor = db.inventory.find().addOption(DBQuery.Option.noTimeout);
```

Write operations

Write operations are used for creating and modifying data in the MongoDB instance. They are atomic operations and on the level of a single document.
The basic write operation is the `insert()`method that is used for creating documents.

The previous example inserts a document in the users collection. If you don't specify _id field it will be automatically added. All MongoDB documents have a unique _id value. If you try to create a document with a duplicate _id value, mongod returns a duplicate key exception.
For modifying existing documents in a collection, you can use the `update()` or the `save()` method.



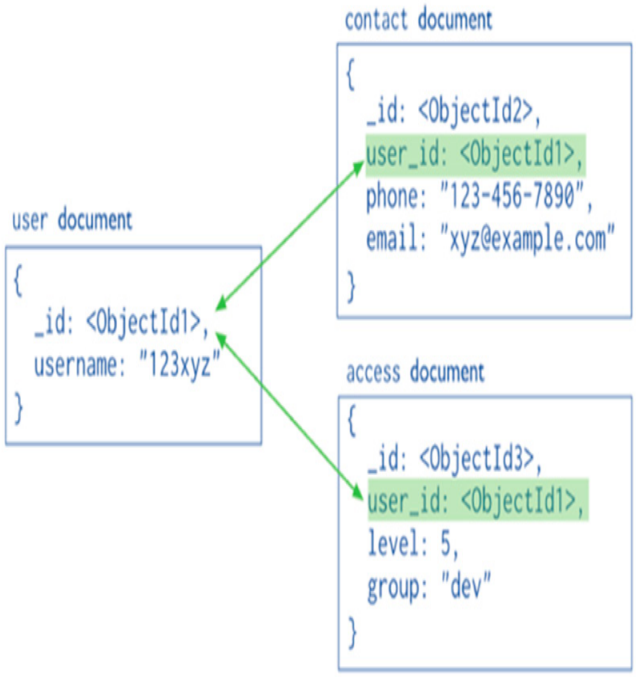
This operation updates all the users with age>18 and sets their status to "A". The multi option is set to true and that means, the method can update multiple documents. The default behavior of the `update()` method is updating a single document. The `save()` methods can only update single documents because it performs replacing. If there is no document with the same _id value, the `save()`method inserts a new document. If you need your update method to insert new document, you have to set the upsert update flag to true.

For deleting documents from a collection, you can use the remove() method.

This operation will remove all the documents that match the criteria. To limit this behavior to a single document, you have to set the justOne parameter to true.

Document models in MongoDB

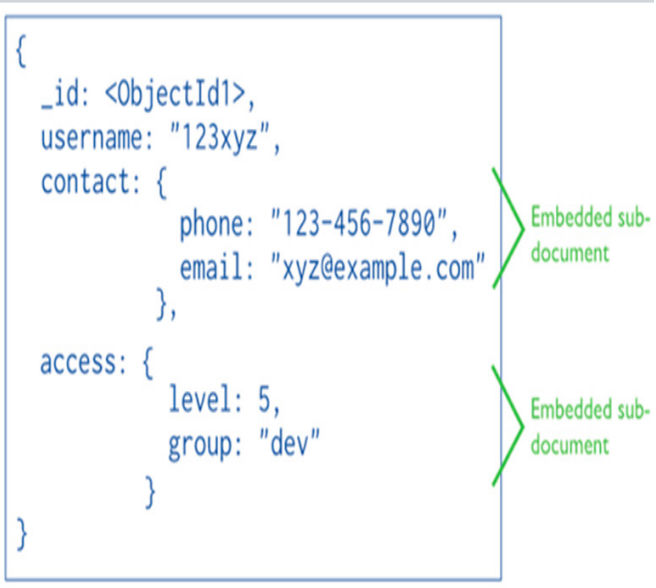
When designing data models, you should be aware of the needs of your application, the performance characteristics of the database engine, and the data retrieval patterns. The most important is the structure of the documents and the relationships between data. There are two tools for this representation—references and embedded documents.



References store the relationship between data by including links or references from one document to another. These are normalized data models. It is good to use normalized data models in these scenarios:

- When embedding would result in duplication of data but would not provide sufficient read performance advantages to outweigh the implications of the duplication
- To represent more complex many-to-many relationships
- To model large hierarchical data sets

In conclusion, references provide more flexibility than embedding, but they require more round trips to the server when executing a query.



Embedded documents store the relationships between data by creating one single document. These are renormalized data models. It is good to use this kind of models in these scenarios:

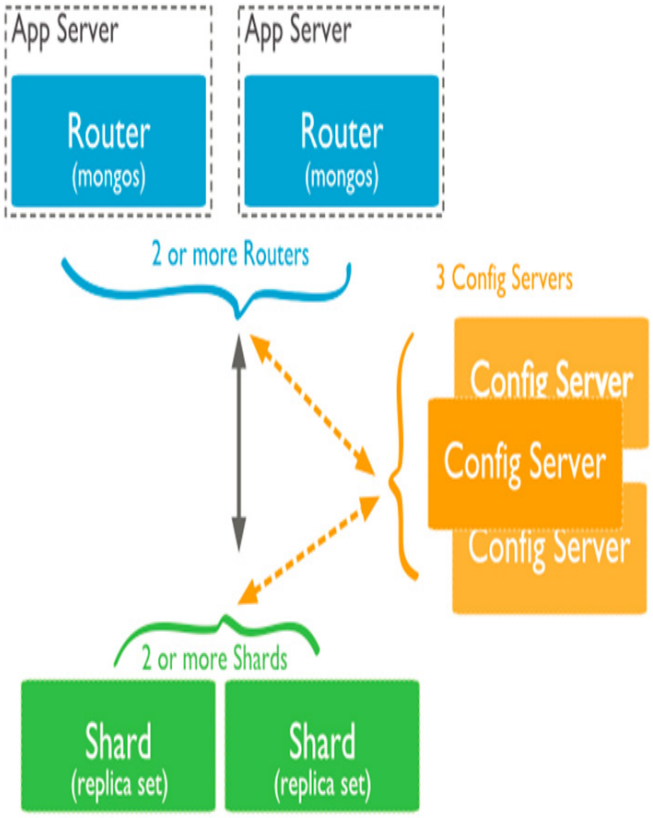
- You have "contains" relationships between entities
- You have one-to-many relationships between entities. In these relationships, the "many" or child documents always appear in the context of the "one" or parent document.

In conclusion, embedding is good for read operations and performing update on related data in a single atomic operation. However, related data can lead to document growth and data fragmentation.

Sharding

When working with databases, as time progresses, data in it increases and it may come to a point when a single machine is not sufficient to store all the data. This problem can be solved with two approaches. The first approach is vertical scaling, which adds more CPU and storage resources to increase capacity. Here, the horizontal scaling, also known as sharding is discussed.

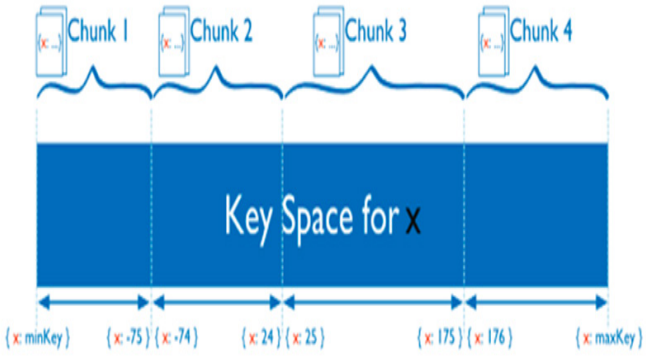
Sharding divides the data over multiple servers or shards. Each shard is an independent database and the data has to be distributed at collection level.



In many cases, this mongo database is referred as mongod. This is not the same as mongo instances, which are referred as mongos. The mongos are not mongo databases; they represent query routers that communicate with the applications and the mongo databases, and will be explained later on.

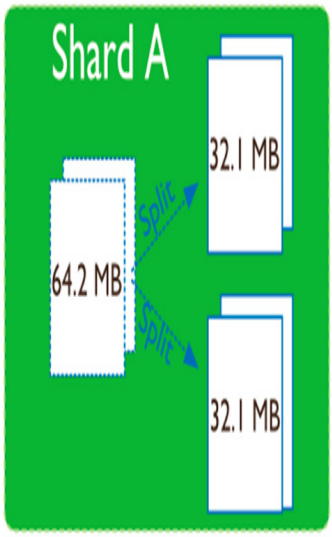
In the picture, you can see the components of the sharded cluster. This section begins with the part that contains a subset of the data and is called a shard. The shard can be a replica set or a single mongod. In production sharded cluster, each shard is a replica set. Every database has a primary shard that holds all the un-sharded collections in that database. The distribution of the collection's documents is determined with a shard key that exists in every document in the collection.

MongoDB partitions data in the collection using ranges of shard key values. Each chunk defines a non-overlapping range of shard values. MongoDB distributes the chunks and their documents, among the shards in the cluster.

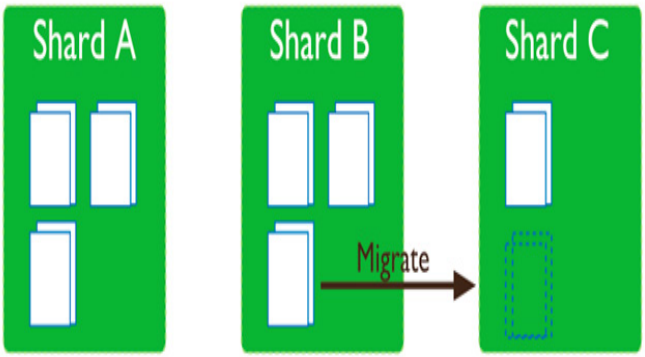


The example uses a numeric shard key and it is range-based sharding. You can also use hashed shard keys. As a hashed shard key, you can use the default _id field of the document. When using this approach, there is a more random distribution of a collection in a cluster.

Config servers are used for storing the metadata for a sharded cluster. This data contains the list of chunks on every shard and the ranges that define the chunks. MongoDB performs writing and reading operations to the config servers. If one or two configuration servers become unavailable, the cluster's metadata becomes read only. If all three servers are unavailable, you can use the cluster if you do not restart mongos instances. If mongo instances are restarted, the mongos will be unable to route reads and writes. Reading operations are performed in cases when a mongos starts, restarts, or after a chunk migration. Writing operations are performed in cases when a chunk grows beyond the specified chunk size and because of that it is split in half.



The new chunks may lead to an uneven distribution across the shards and in this case, the chunks are redistributed across the shards.



This is automatic migration of chunks. In limited cases, you can also use manual migration using the moveChunk command.

db.adminCommand({ moveChunk : "myapp.users", find : {username : "smith"},
to : "mongodb-shard3.example.net" })

This command moves the chunk that includes the shard key value "smith" to the shard named mongodb-shard3.example.net.

The default chunk size is 64 MB. When deciding on a chunk size, you must consider the fact that, if you use small chunks there is more even distribution of data but there will be more query routing operations; with large chunks, you have a fewer migrations but there is more uneven distribution of data.

Query routers or mongo instances interface with client applications and direct operations to the appropriate shard or shards. The query router processes and targets operations to shards, and then returns results to the clients. The mongos tracks what data is on which shard by caching the metadata from the configuration servers. A mongos instance routes a query to a cluster by determining the list of shards that must receive the query and establishing a cursor on all targeted shards. The operations in a sharded environment can be broadcast or targeted. In broadcast operation, mongo instances broadcast queries to all shards unless the mongos can determine which shards or subset of shards stores this data. Targeted operations are based on the shard key and they are targeted at a single shard or limited group of shards.

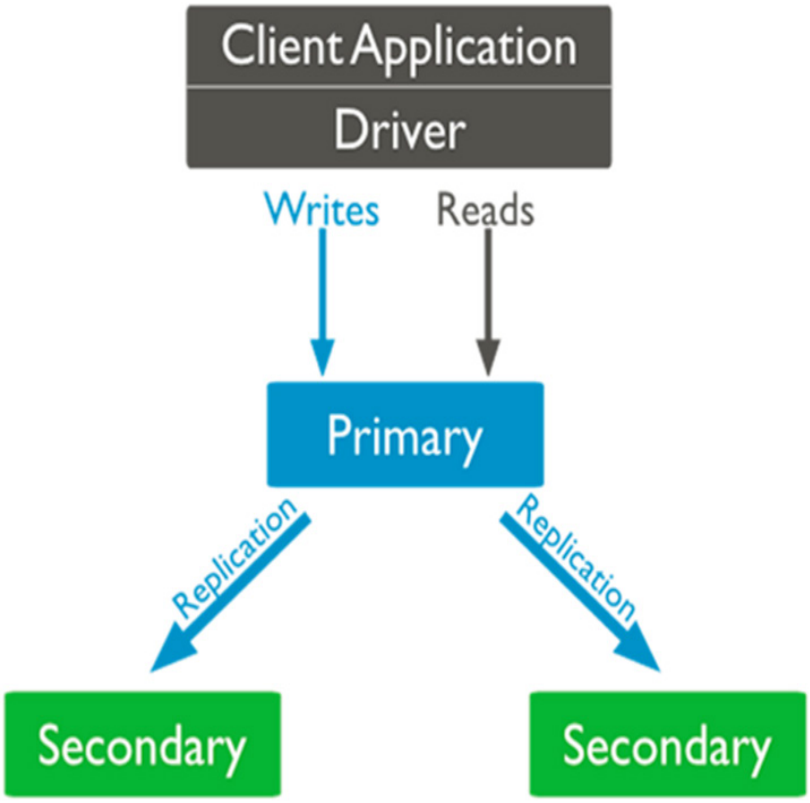
Reasons for not sharding MongoDB

It looks like sharding solves all your problems with large repositories. Well, that is not entirely true. Sharding enables you to shard the repository horizontally, but it also has severe drawbacks:

- When you decide to use sharding, you have to set up sharding keys and every decision you make about the database will be constrained by that choice.
- Deployment is complicated when you use sharding. You have to grab all snapshots, set up new configuration servers, edit the config server data with new shard names, and then start Mongo servers.
- Once you have your sharded database in production, you have to think about the fact that your sharded infrastructure has more places in it, and no change in any one component's mean-time-to-fail, leaving you more exposed
- In a sharded setup, the network connectivity between the Mongo router, config servers, and each shard influences overall performance.

Replica sets

Replica set is a group of mongod instances that host the same data set. One replica set consists of primary member, secondary members, and sometimes an arbiter.



Replica set can have only one primary. The primary accepts all write operations from clients. MongoDB applies write operations on the primary, and then records the operations on the primary's Oplog. Oplog(operations log) is a special capped collection that keeps a rolling record of all operations that modify the data stored in your database. The secondary replicate the primary's oplog and apply the operations to their data sets. Clients cannot write data to secondary but they can read data from them. If the primary becomes unavailable, the secondary can become a primary. In this case, the replica set holds an "election" and the secondary with the highest priority is set to be the new primary. By default, all members have a priority of 1, but you can adjust priorities so that only members in a specific data center can become primary.

A secondary can be configured with priority 0 and it will be prevented of becoming a primary, which means it will serve as a cold standby. This kind of members is called priority 0 set members. Secondary can also be configured to prevent applications from reading it, which allows it to run applications that require separation from normal traffic, like reporting and backups. This kind of members is called hidden members and they cannot vote on elections. The third kind of secondary members are delayed replica set members. They apply operations from the oplog on some configured delay. They are used for recovering information in case of an error and they must have priority 0 and also be hidden. In some replica sets, there is an extra mongod instance that is called arbiter. The arbiter does not have a data set. It exists only for voting on elections and it cannot become secondary or primary. You should add an arbiter only to sets with even members to prevent tied elections. On the other hand, a primary may become a secondary and also a secondary can become a primary during an election.

From the perspective of a client application, whether MongoDB instance is running as a single server or a replica set is transparent. By default, in MongoDB read operations return results from the primary member. However, it can be configured that this operations read data from secondary members. In this case, there is a risk that the query may return data that reflects a previous state. This is characterized as eventual consistency because eventually the secondary will have the same state as the primary. To guarantee consistency for reads from secondary members, you can configure the client and driver to ensure that write operations succeed on all members before completing successfully.

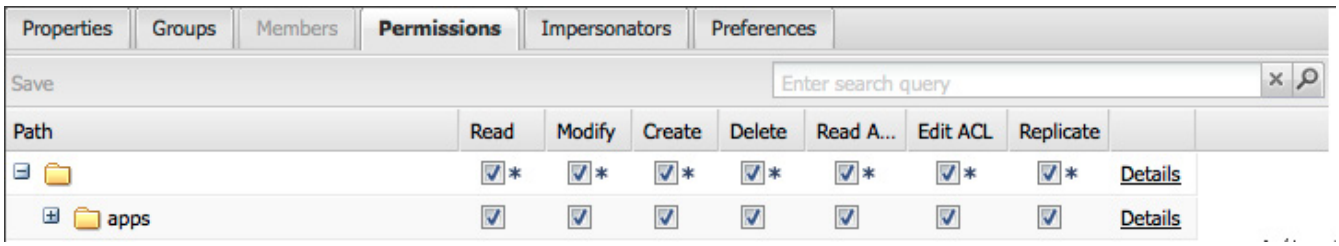
9

Users, Groups, Permissions

AEM uses ACLs to determine what actions a user or group and can take and where it can perform those actions.

Permissions and ACLs

Permissions define who is allowed to perform which actions on a resource. The permissions are the result of access control evaluations. You can change the permissions granted/denied to a given user by selecting or clearing the checkboxes for the individual AEM actions. A check mark indicates that an action is allowed. No checkmark indicates that an action is denied.



Where the checkmark is located in the grid also indicates what permissions users have in what locations within AEM (that is, which paths).

Action	Description
Allow (Check mark)	CQ WCM allows the user to perform the action on this page or on any child pages.
Deny (No checkmark)	CQ WCM does not allow the user to perform the action on this page nor on any child pages.

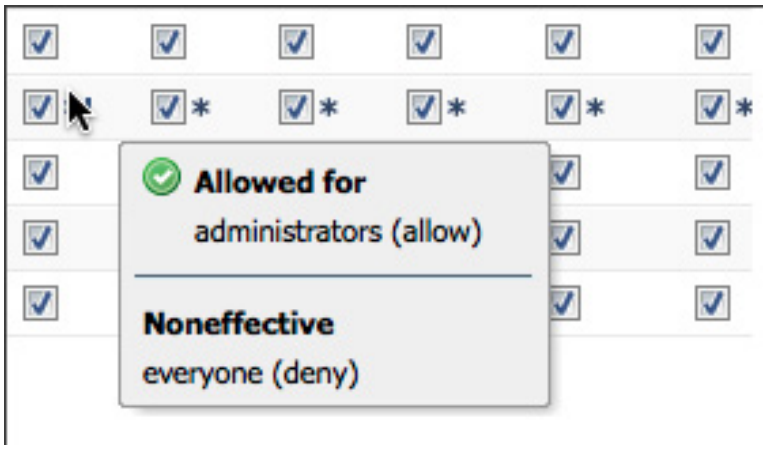
The permissions are also applied to any child pages. If a permission is not inherited from the parent node but has at least one local entry for it, then the following symbols are appended to the check box. A local entry is one that is created in the CRX 2.3 interface (Wildcard in the path of ACLs currently can only be created in CRX.)

For an action at a given path:

* (asterisk)	There is at least one local entry (either effective or ineffective). These wildcard ACLs are defined in CRX.
! (exclamation mark)	There is at least one entry that currently has no effect.

When you hover over the asterisk or exclamation mark, a tooltip provides more details about the declared entries. The tooltip is split into two parts:

Upper part	Lists the effective entries.
Lower part	Lists the noneffective entries that may have an effect somewhere else in the tree (as indicated by a special attribute present with the corresponding ACE limiting the scope of the entry). Alternatively, this is an entry whose effect has been revoked by another entry defined at the given path or at an ancestor node.



Actions

Actions can be performed on a page (resource). For each page in the hierarchy, you can specify which action the user is allowed to take on that page. Permissions enable you to allow or deny an action.

Action	Description
Read	The user is allowed to read the page and any child pages.
Modify	The user can modify existing content on the page and on any child pages. At the JCR level, users can modify a resource by modifying its properties, locking, versioning, nt-modifications, and they have complete write permission on nodes defining a icr:content child node, for example cq:Page, nt:file, cq:Asset.
Create	The user can: • create new paragraphs on the page or on any child page. • create a new page or child page. If **modify** is denied the subtrees below icr:content are specifically excluded because the creation of icr:content and its child nodes are consierd a page modification. This only applies to nodes defining a icr:content child node.
Delete	The user can: • delete existing paragraphs from the page or any child page. • delete a page or child page. If **modify** is denied any subtrees below icr:content are specifically excluded as removing icr:content and its child nodes is considered a page modification. This only applies to nodes defining a icr:content child node.
Read ACL	The user can read the access control list of the page or child pages.
Edit ACL	The user can modify the access control list of the page or any child pages.
Replicate	The user can replicate content to another environment (for example, the Publish environment). The privilege is also applied to any child pages.

Access Control Lists and how they are evaluated

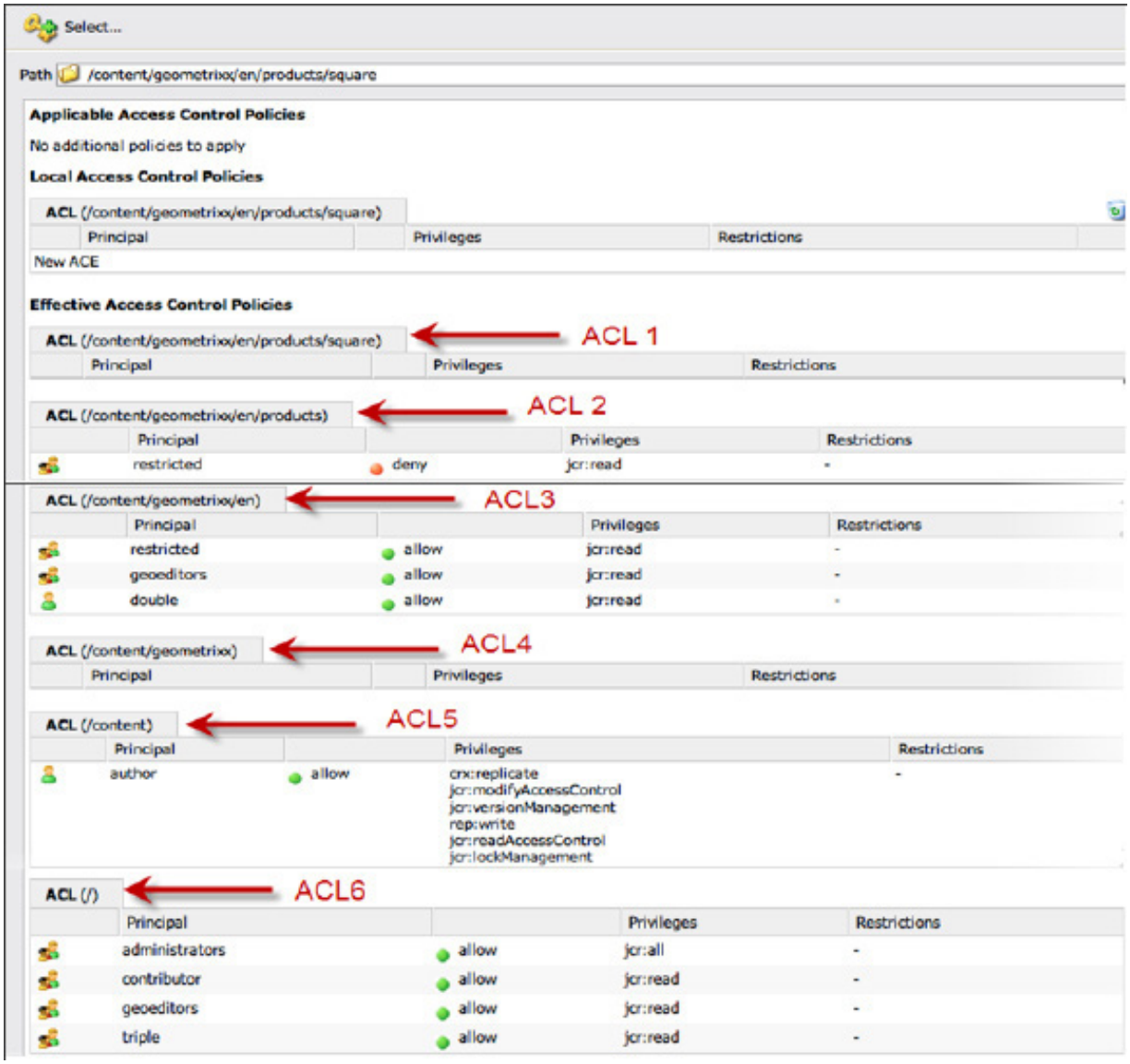
AEM Sites uses Access Control Lists (ACLs) to organize the permissions being applied to the various pages.

Access Control Lists are made up of the individual permissions and are used to determine the order in which these permissions are actually applied. The list is formed according to the hierarchy of the pages under consideration. This list is then scanned bottom-up until the first appropriate permission to apply to a page is found.

Assume that a user wants to access to the following page

/content/geometrixx/en/products/square

And the ACL list for that page is the following one



The ACL (or permission) that will be applied to the page is ACL 1, related to /content/geometrixx/en/products/square. In our case the ACL associated to this page is empty, so AEM will go one level up to the ACL 2.

ACL 2 will be applied if the user belongs to the group restricted and in this case, the user will have access denied for this page. If the user is not part of the restricted group then AEM will go up another level to ACL 3.

ACL 3 will be applied if the user belongs to the group geoeditors or double only, in this case the user will have the access granted to the page. If the user is part of the restricted group as we saw the ACL applied is ACL 2. If the user is not part of the 3 mentioned groups before AEM will go up one level to ACL 4.

ACL 4 being empty, if AEM reach this level it will go up again one level to ACL 5.

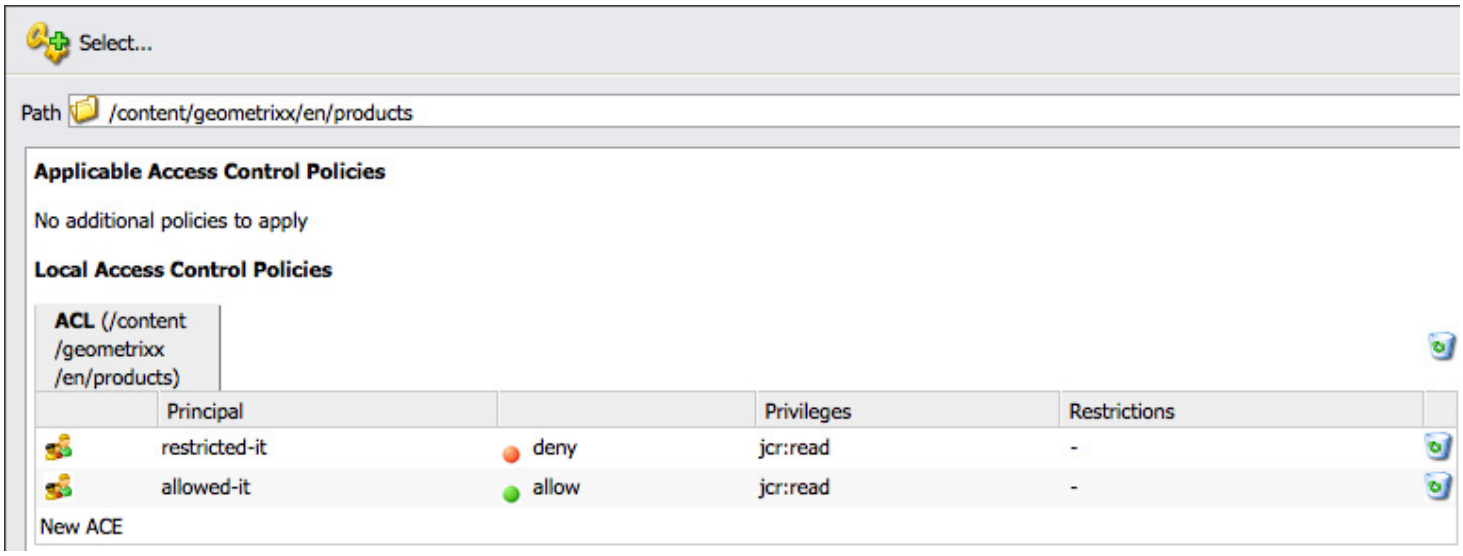
ACL 5 will be applied if the user belongs to the author group and then the user will have the access granted to the page. If the user is no part of the author group AEM will go up one level to ACL 6.

ACL 6 will be applied if the user belongs to the administrators, contributor or triple group, if this is the case the user will have the access granted to the page. If the user is part of the geoeditors group as we saw then AEM would have had already applied ACL 3 and granted access to the page. If the user is not part of any group he will have of course the access denied to the page.

Concurrent permission on ACLs

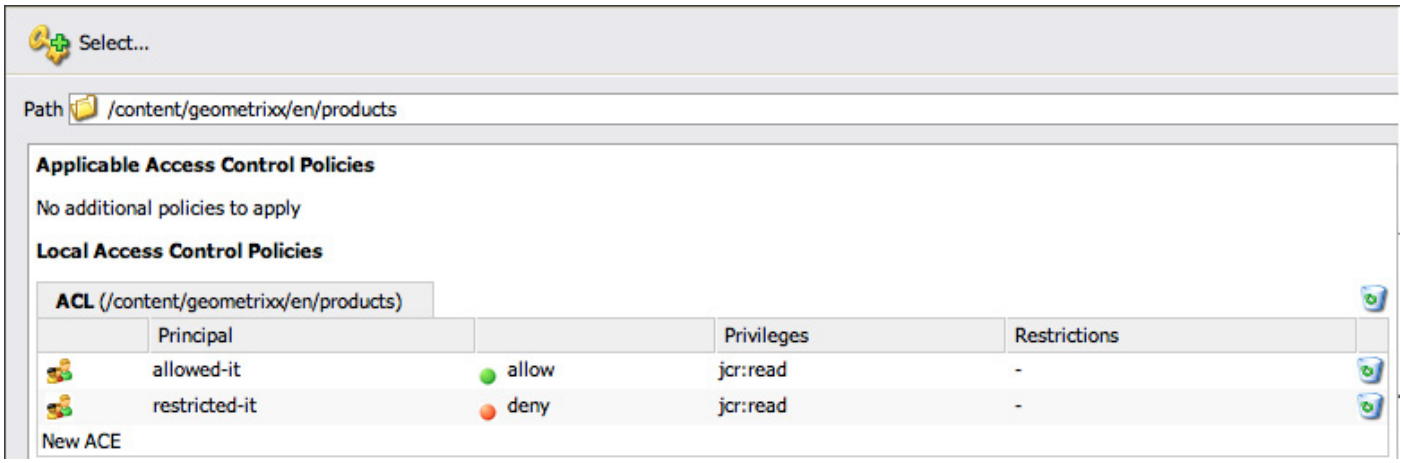
When two concurrent (and opposing) permissions are listed on the same ACL for the same resource, the ACL that is applied for such resource is the one at the bottom:

Suppose that we have the following ACL for the same resource under /content/geometrixx/en/products:



If a user is part of the two groups allowed-it and restricted-it, he will see the access to the page products denied (because the ACL deny in read access is the rule at the bottom).

Now if the order of the ACL is the opposite:



This time when a user is part of the two groups allowed-it and restricted-it, he will see the access granted to the page products (because the ACL allow in read access is the rule at the bottom).

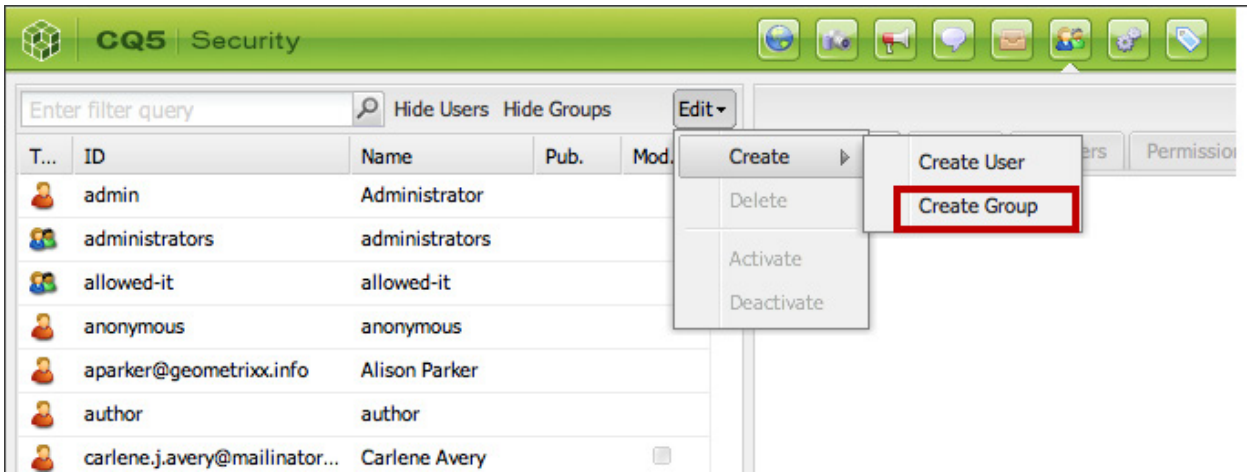
EXERCISE 9.1 - ACLs

Goal

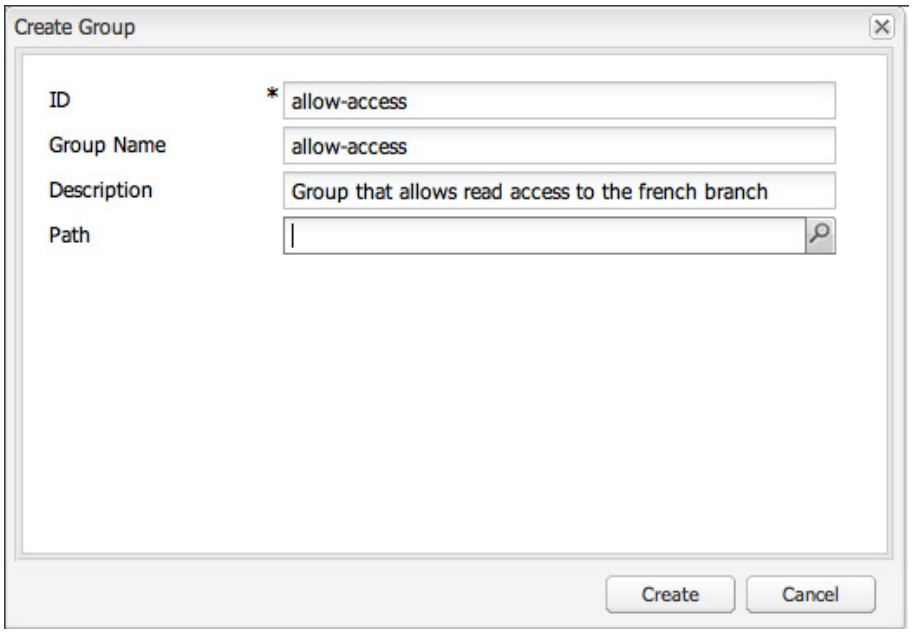
We are going to create a user which will be part of two groups with two different ACLs and we will modify them programmatically

Steps

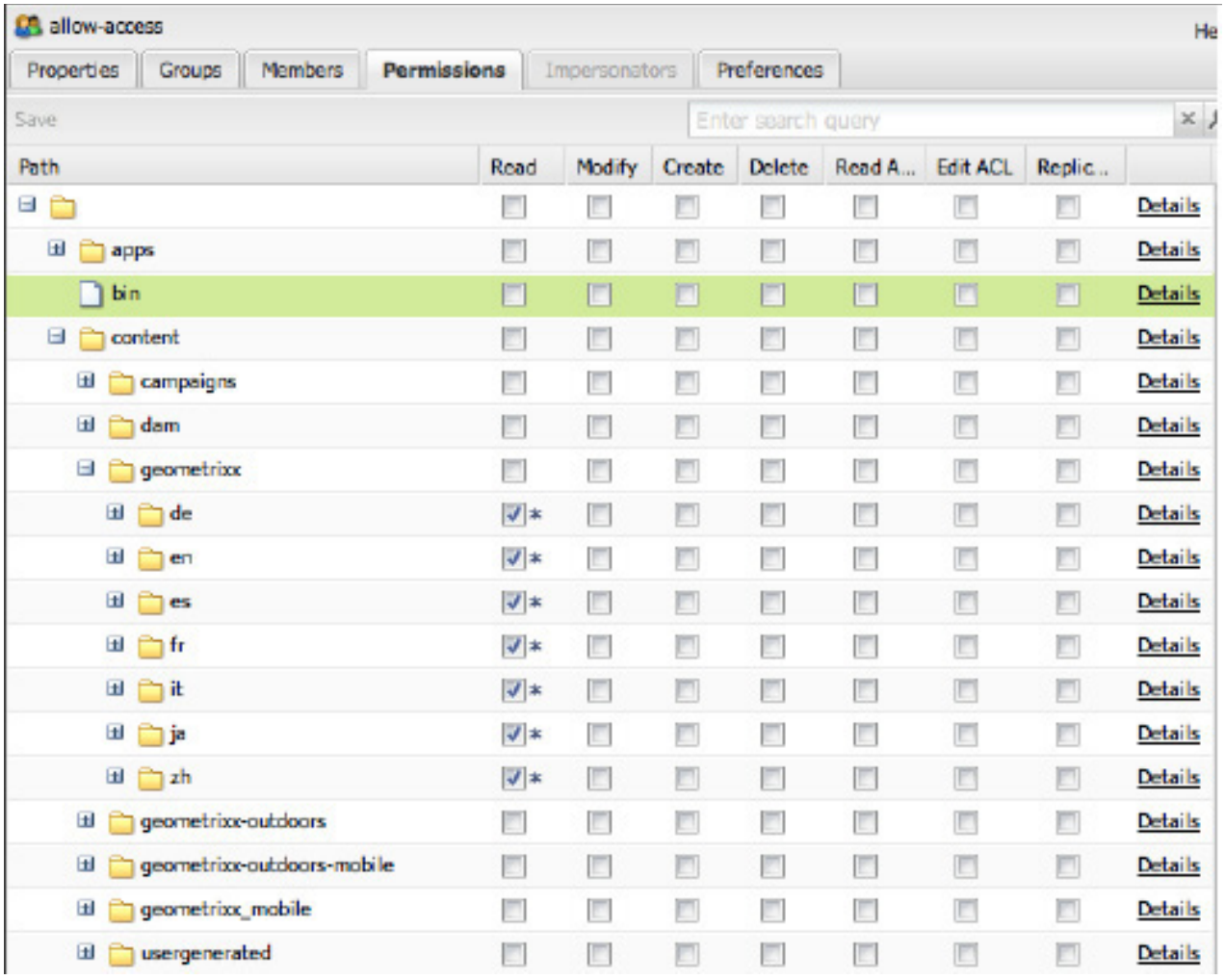
1. Create a group "allow-access" the group should have read access to /content/geometrixx/fr. Choose the user menu, then on the left pane click on the Edit button. A drop down menu will allow you to Create your group



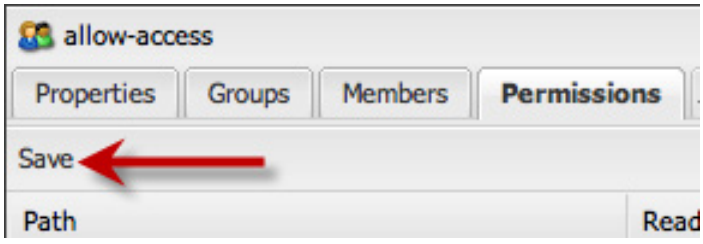
Create the group as following:



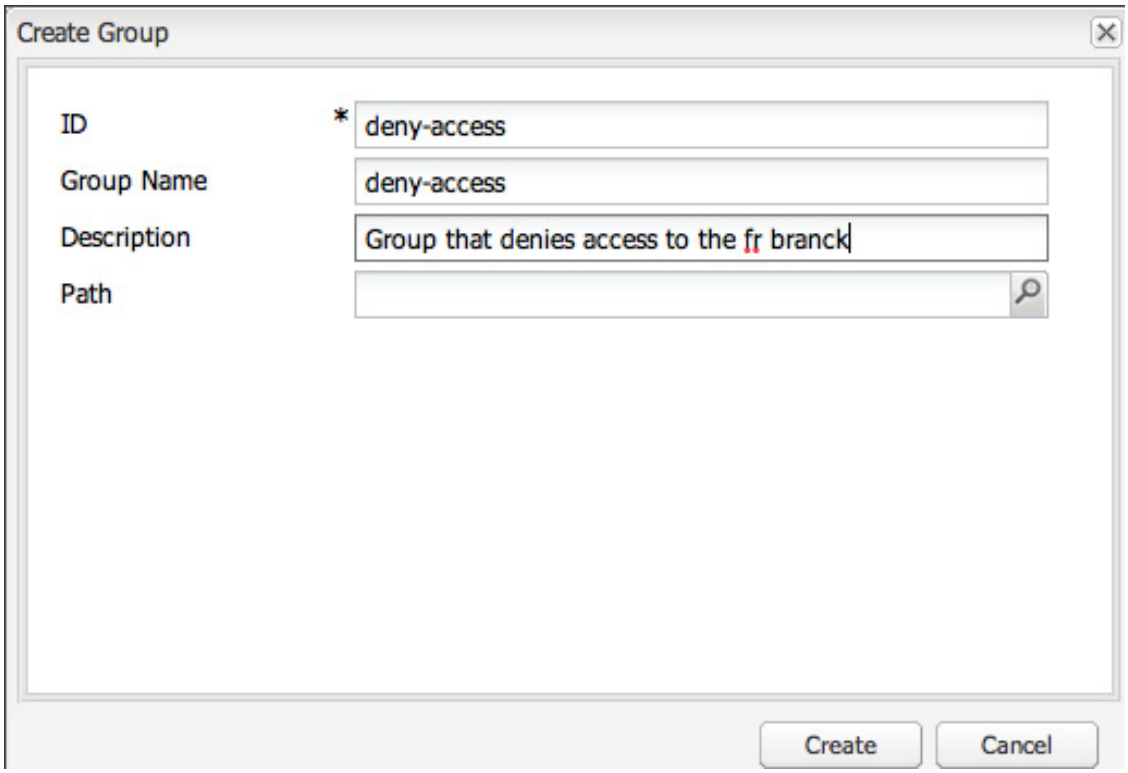
Open the newly created group and assign read rights to the all the languages branches, as following:



Don't forget to click on Save after you're done

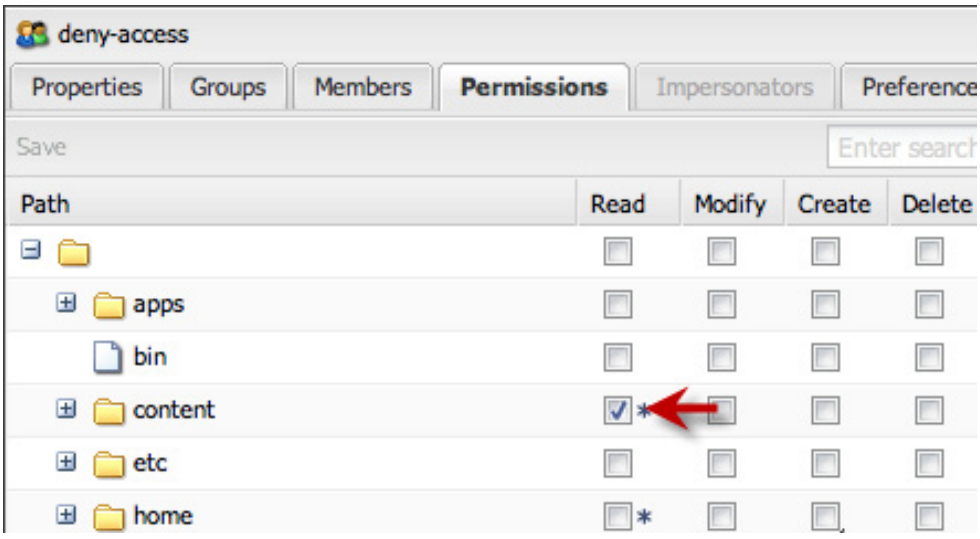


2. Create another group "deny-access", the group should not have read access to /content/geometrixx/fr.

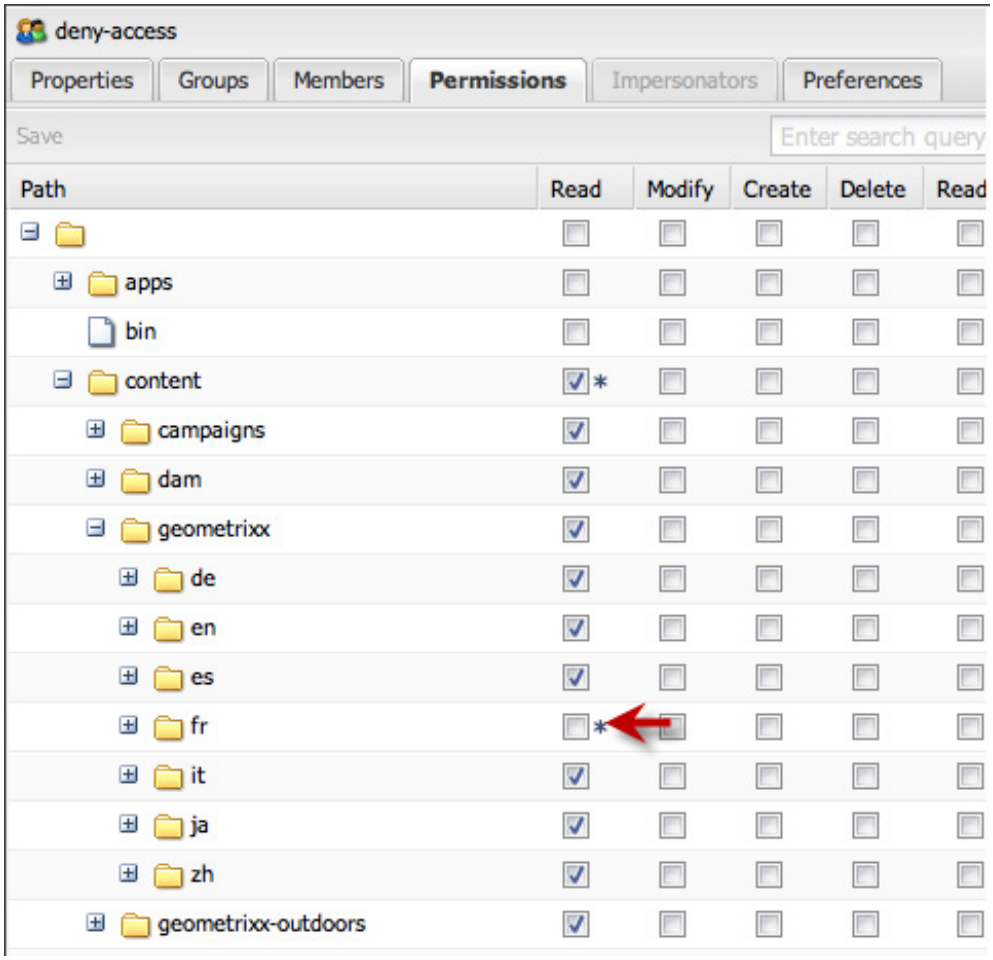


Open the newly created group and assign read rights to the all the languages branches, except for the French one as following.

For that first provide read access to the /content node

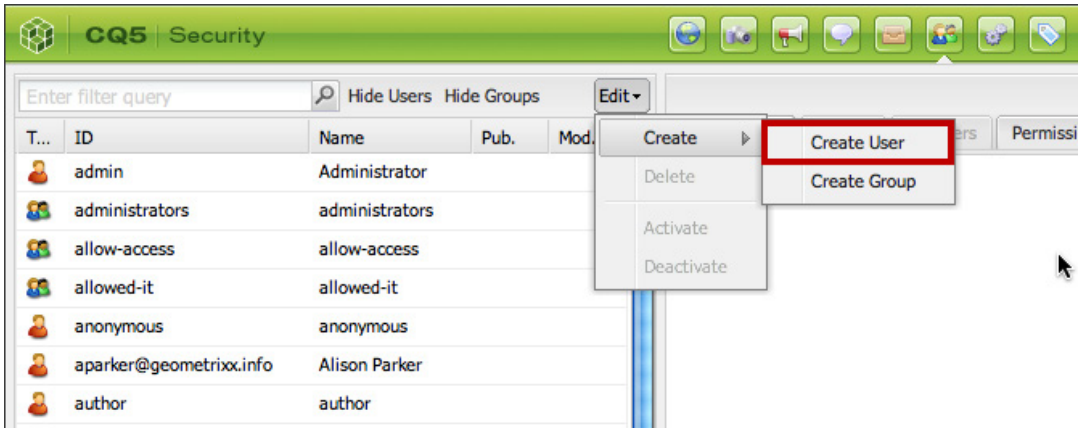


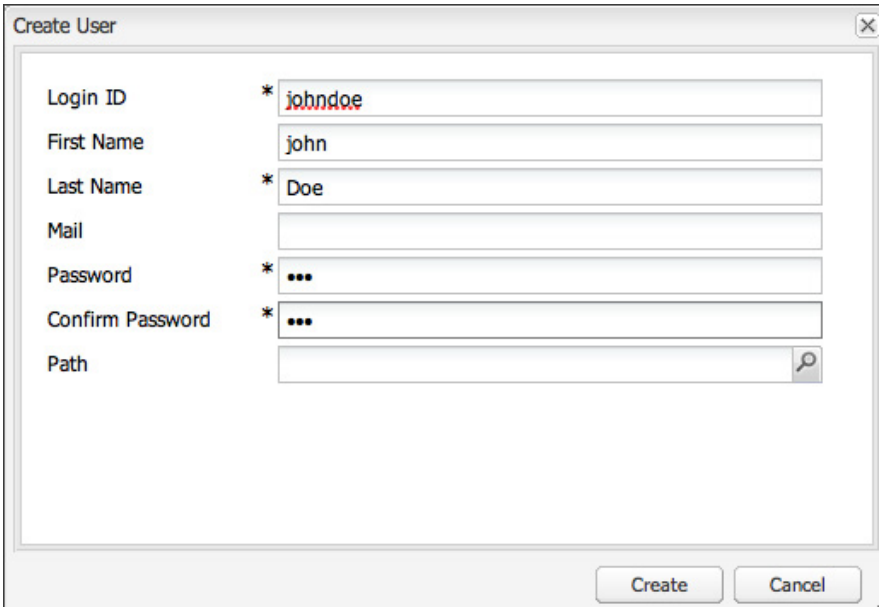
Save your change, refresh your page, expand the content node and deselect the fr node.



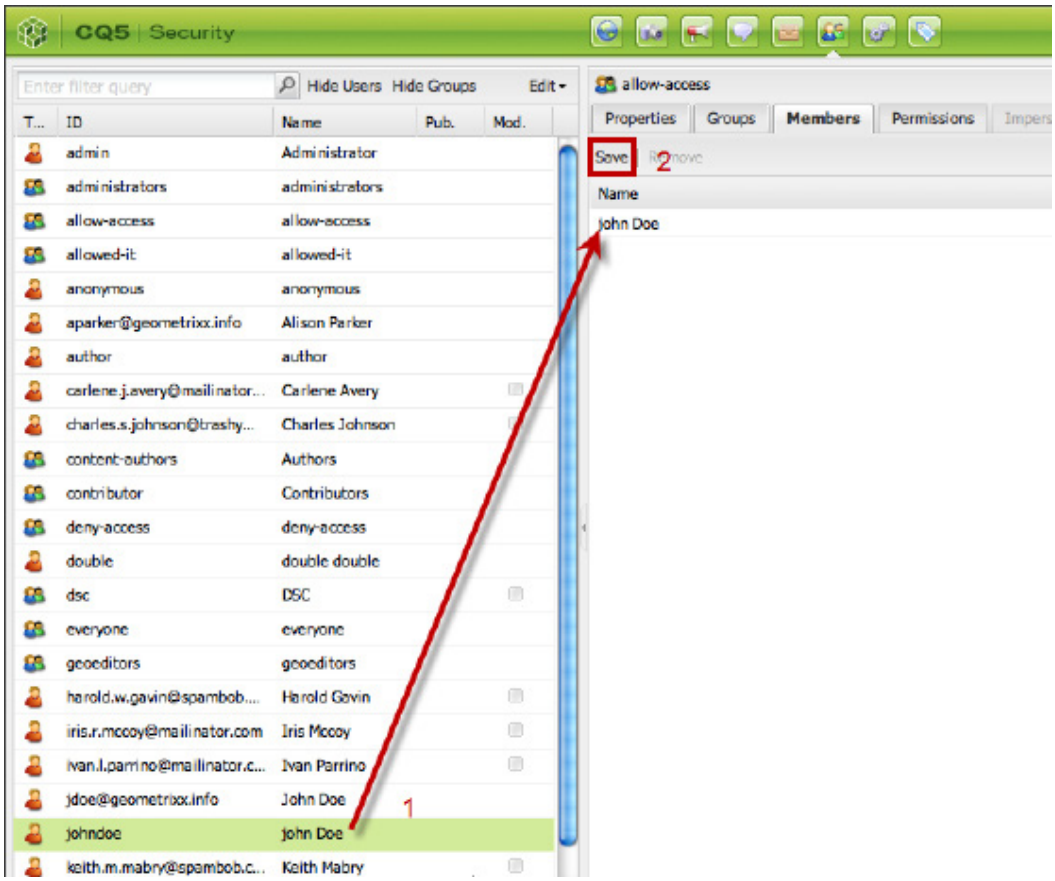
Don't forget to save the changes by clicking on Save!

3. Create the user John Doe, for that proceed the same way as you did when creating a new group, and choose "Create User"





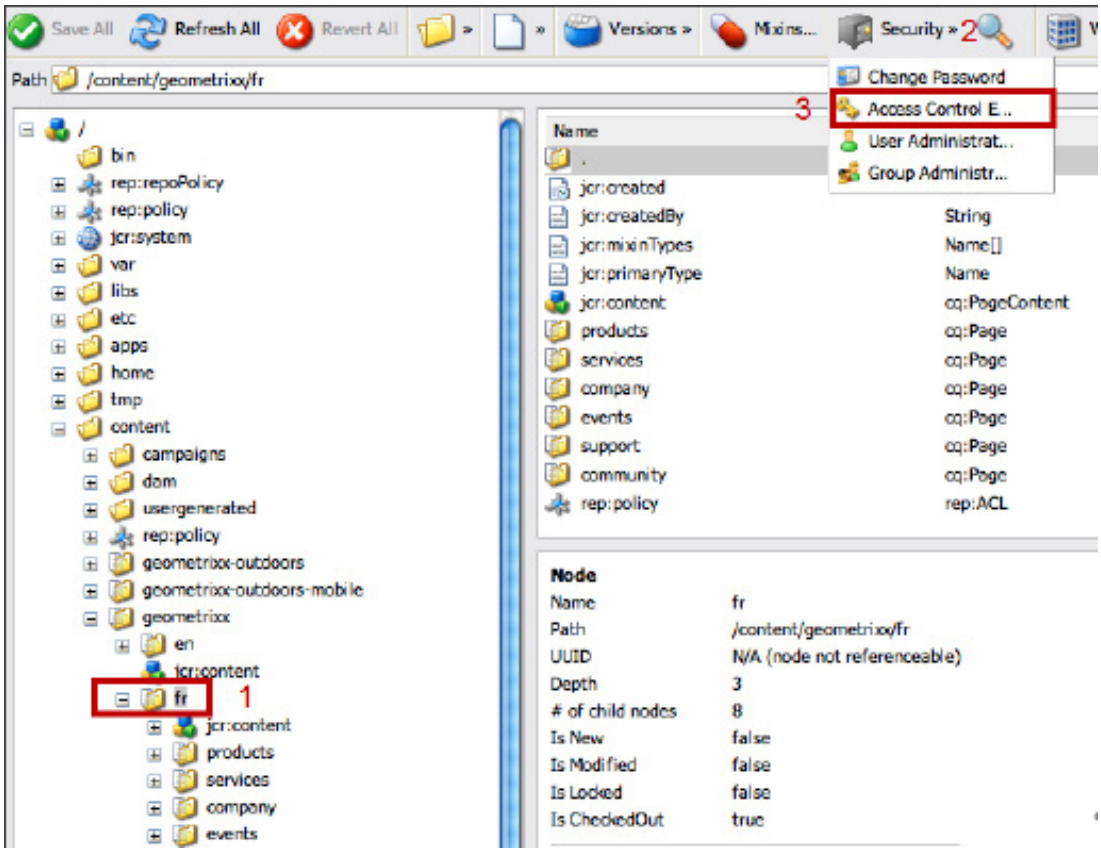
4. Add the user John Doe to the allow-access and deny-access group. For that you'll have to open each group, click on Members and Drag & Drop the user John Doe to the member list



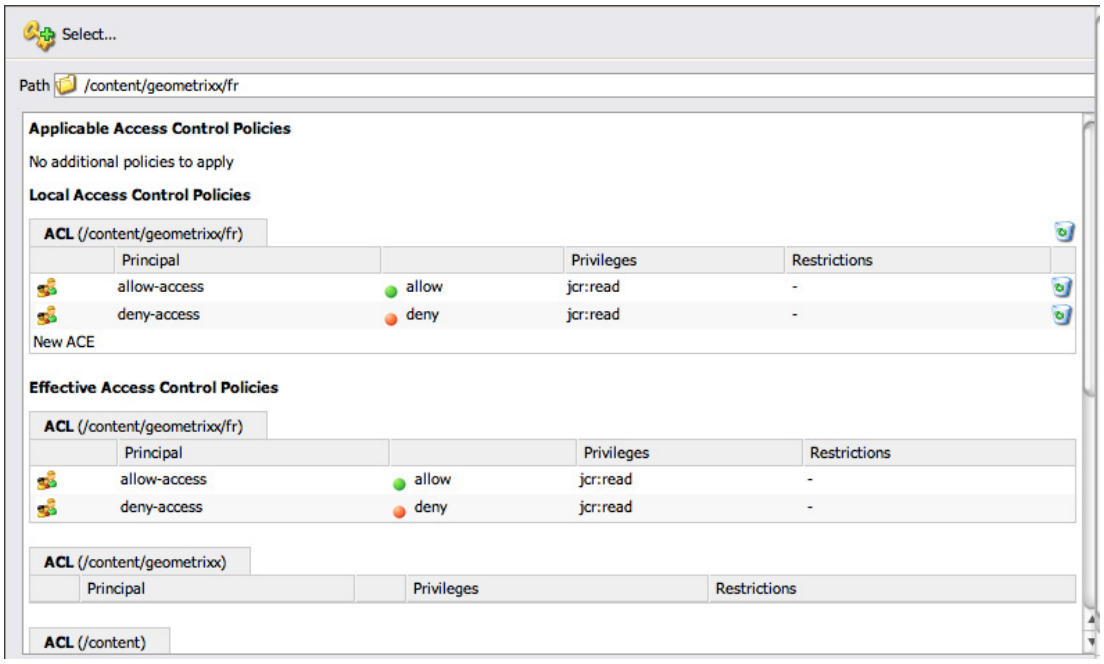
Don't forget to click on Save each time!

5. Add the user to the group contributors so that he has also access to elements like CSS-JS libraries as well as the design associated to the page.

6. Open CRX explorer and click on the French node, then in the Security menu choose Access Control Editor.



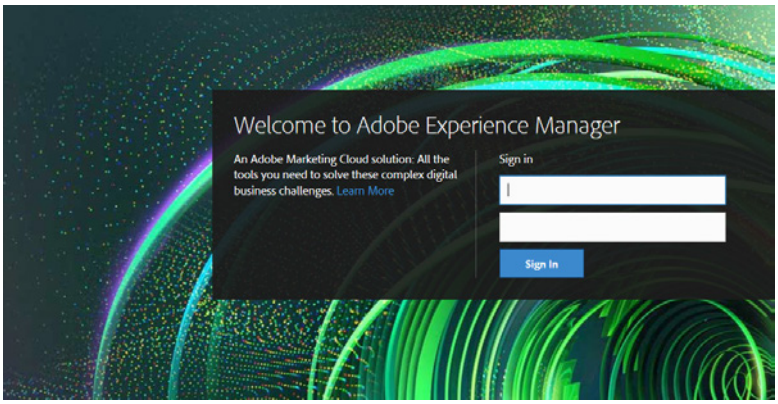
You should see the following information:



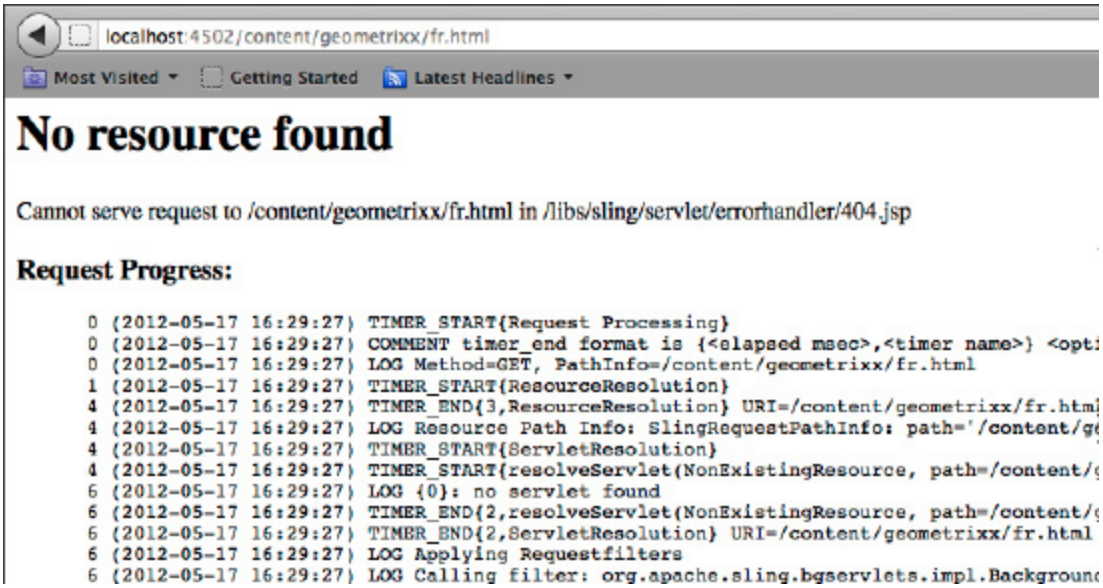
As you can see, the user John Doe doesn't have access to the French page as the deny rule is the one at the bottom of the ACL rules applied to the resource /content/geometrixx/fr and therefore takes precedence.

7. Open the page http://localhost:4502/content/geometrixx/fr.html. in you browser.

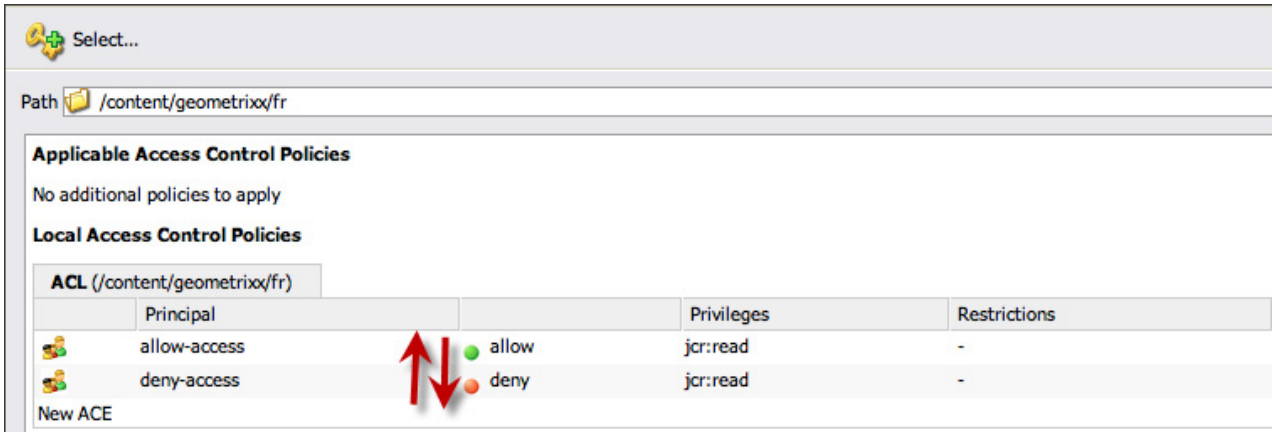
Log in as John Doe



As the user John Doe doesn't have access to the page he will see the following screen:



8. Open again the ACL editor for the French page. You will change the order of the ACL list by sliding the deny access ACL on top of the allow access ACL.

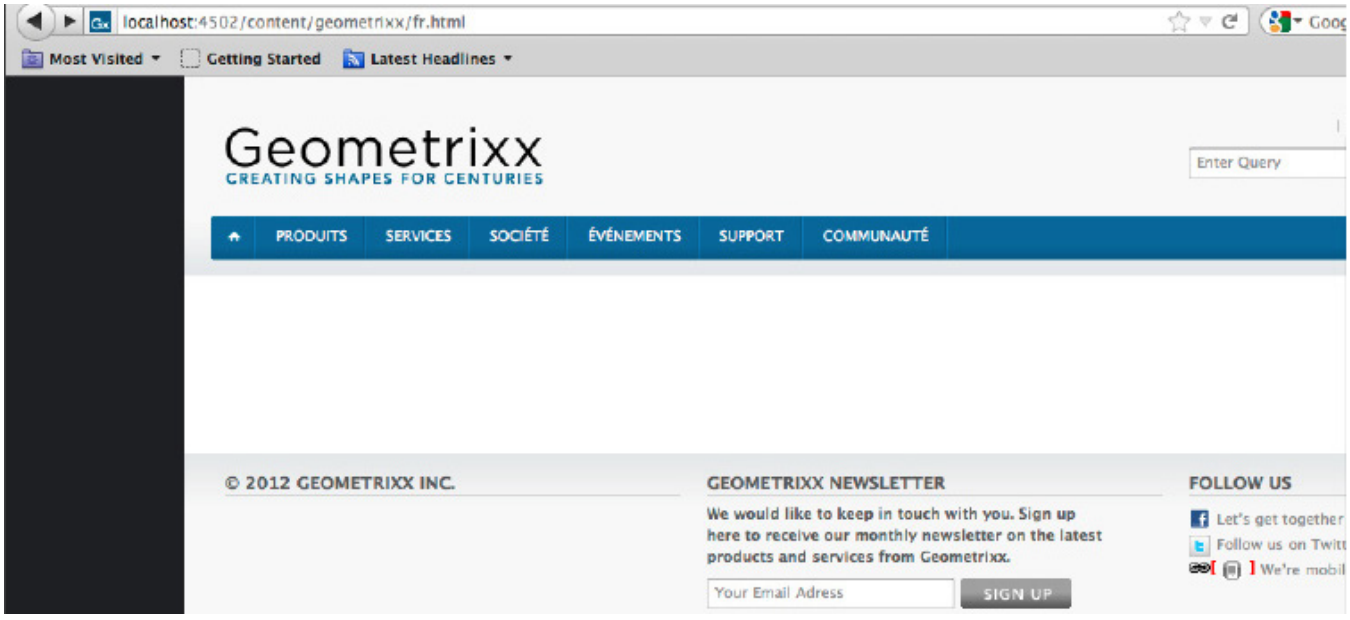


You should see the following:

Click on the Apply button then Ok.

9. John Doe now should have access to the French page as the ACL rule on the bottom is the one granting him read access to the node. Get disconnected from the previous session and open again the page http://localhost:4502/content/geometrixx/fr.html. Again use the user John Doe to log in.

This time you should be able to see the French page.



10. Now re-order the ACLs again so that the user does not have access to the French page.

EXERCISE 9.2 – Automating ACLs

1. Let's now write a service that grants the group, named deny-access, read access to /content/geometrixx/fr.

   Your pom.xml has already the following dependency

```
<dependency>
        <groupId>org.apache.jackrabbit</groupId>
        <artifactId>jackrabbit-api</artifactId>
        <version>2.2.0</version>
        <scope>provided</scope>
</dependency>
```

You cannot easily find this bundle in the AEM Web Console. This bundle is exported by the system bundle (bundle 0).

The service you implement will add the required permissions to the specified location upon activation of the OSGi component. Create the ModifyPermissions component, with the appropriate OSGi annotations.

```
package com.adobe.training.core;

import java.util.NoSuchElementException;
import javax.jcr.RepositoryException;
import javax.jcr.Session;
import javax.jcr.security.AccessControlList;
import javax.jcr.security.AccessControlManager;
import javax.jcr.security.AccessControlPolicyIterator;
import javax.jcr.security.Privilege;
import org.apache.sling.jcr.api.SlingRepository;
import org.apache.felix.scr.annotations.Activate;
import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.jackrabbit.api.security.JackrabbitAccessControlList;
import org.apache.jackrabbit.api.security.user.Authorizable;
import org.apache.jackrabbit.api.security.user.UserManager;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Component
public class ModifyPermissions {
        private static final String CONTENT_GEOMETRIXX_FR = "/content/geometrixx/fr";
        private static final Logger LOGGER=  LoggerFactory.getLogger(ModifyPermissions.class);

@Reference
        private SlingRepository repo;

        @Activate
        protected void activate(){
                LOGGER.info("ModifyPermissions activated");
                modifyPermissions();
        }

        private void modifyPermissions() {
                Session adminSession = null;
                try{
                        adminSession= repo.loginAdministrative(null);

                        UserManager userMgr=
((org.apache.jackrabbit.api.JackrabbitSession)adminSession).getUserManager();
                        AccessControlManager accessControlManager = adminSession.getAccessControlMa-
nager();

                        Authorizable denyAccess = userMgr.getAuthorizable("deny-access");

                        AccessControlPolicyIterator policyIterator =

accessControlManager.getApplicablePolicies(CONTENT_GEOMETRIXX_FR);
                        AccessControlList acl;
                        try{
                                acl=(JackrabbitAccessControlList)
policyIterator.nextAccessControlPolicy();
                        }catch(NoSuchElementException nse){
                                acl=(JackrabbitAccessControlList)
 accessControlManager.getPolicies(CONTENT_GEOMETRIXX_FR)[0];
```

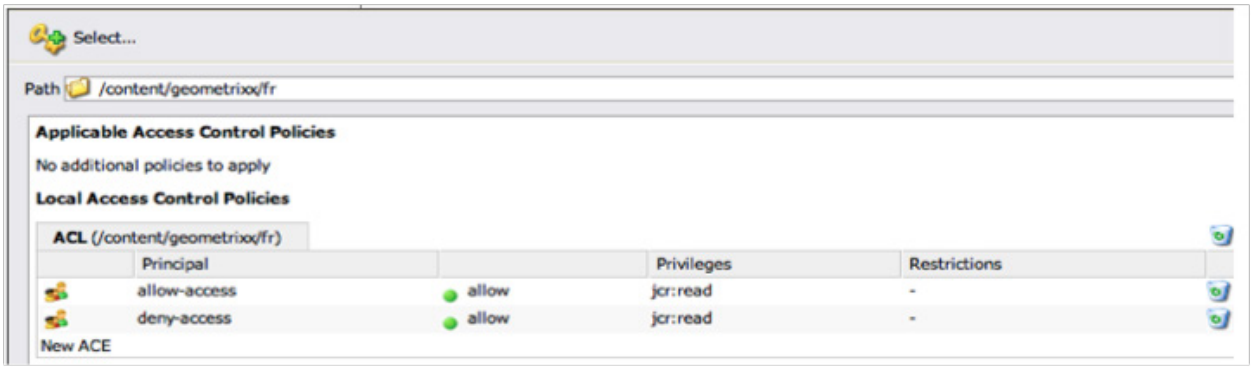
```
                        }

                        Privilege[] privileges =
{accessControlManager.privilegeFromName(Privilege.JCR_READ)};
                        acl.addAccessControlEntry(denyAccess.getPrincipal(), privileges);
                        accessControlManager.setPolicy(CONTENT_GEOMETRIXX_FR, acl);
                        adminSession.save();
                }catch (RepositoryException e){
                        LOGGER.error("**************************Repo Exception", e);
                }finally{
                        if (adminSession != null)
                                adminSession.logout();
                }
        }

}
```

2. Deploy the bundle by executing:

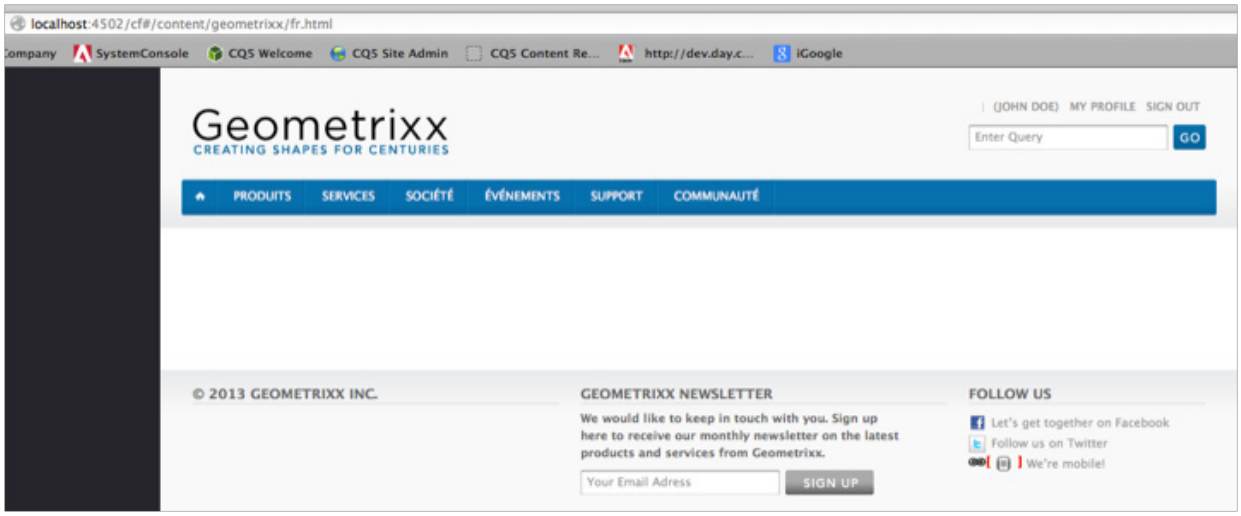
```
mvn clean install –P bundle
```

3. Reopen the ACL Editor and check the permissions for the page:

```
/content/geometrixx/fr
```



4. And of course, when you sign in with user John Doe again, you are able to open the French page.



10

Testing (Sling and Maven)

In this chapter we are going to describe how unit tests can be conducted inside AEM.

First we will see how to use the JUnit framework, and then we will use Easymock and Powermock to perform various tests. Finally we will proceed to do some Slingbased tests using JUnit.

Junit

JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.

How do you write testing code?

The simplest way is as a test expression in a debugger. You can change debug expressions without recompiling, and you can wait to decide which expressions to define until you have seen the running objects. You can also write test expressions as statements that print to the standard output stream. Both styles of tests are limited because they require human judgment to analyze their results. Also, they don't compose nicely- you can only execute one debug expression at a time and a program with too many print statements causes the dreaded "Scroll Blindness".

JUnit tests do not require human judgment to interpret, and it is easy to run many of them at the same time. When you need to test something, here is what you do:

1. Annotate a method with @org.junit.Test

2. When you want to check a value, import org.junit.Assert.* statically, call assertTrue() and pass a boolean that is true if the test succeeds (eventually you can use all the other methods provided by the Assert class like assertEquals, assertArrayEquals, assertFalse, assertNonNull, assertSame, etc..)

3. Then because we are using Maven we just need to test it by using the following script "mvn test" or "mvn clean test" to force Maven to recompile your project. (Eventually if you want to run only some tests amongst all the ones in your code you can do so by providing on the command line "mvn –Dtest=MyTestsCase, MyOtherTestCase test" to run only the mentioned tests.

EasyMock

EasyMock provides MockObjects for interfaces (and objects through the class extension) by generating them on the fly using Java's Dynamic Proxy mechanism. EasyMock is a library that provides an easy way to use Mock Objects for given interfaces or classes.

Mock Objects simulate parts of the behavior of domain code, and are able to check whether they are used as defined. Domain classes can be tested in isolation by simulating their collaborators with Mock Objects.

Writing and maintaining Mock Objects often is a tedious task that may introduce errors. EasyMock generates Mock Objects dynamically - no need to write them, and no generated code!

EasyMock Benefits

- Hand-writing classes for Mock Objects is not needed.
- Supports refactoring-safe Mock Objects: test code will not break at runtime when renaming methods or reordering method parameters
- Supports return values and exceptions.
- Supports checking the order of method calls, for one or more Mock Objects

Usually when writing a test using EasyMock the following steps are taken in order to Mock our Objects:

- Create a Mock Object for the interface we would like to simulate,
- Record the expected behavior, and
- Switch the Mock Object to replay state. Up to this state in our tests, all missed expectations are shown, as well as all fulfilled expectations for the unexpected call. If the method call is executed too often, the Mock Object complains, too
- Verify that the specified behavior has been used (so if none of the methods or expectations before are called we need also to be notified about it). To do this, we have to call verify(mock):

Here is a small example: most parts of a software system do not work in isolation, but collaborate with other parts to get their job done. In a lot of cases, we do not care about using collaborators in unit testing, as we trust these collaborators. Mock Objects help us to test the unit under test in isolation. Mock Objects replace collaborators of the unit under test.

The following simple example use the interface Collaborator and a Class that we want to test:

```
package org.easymock.samples;
        public interface Collaborator {
                void documentAdded(String title);
                void documentChanged(String title);
                void documentRemoved(String title);
                byte voteForRemoval(String title);
                byte[] voteForRemovals(String[] title); }
```

We want to test its behavior in the following class:

```
public class ClassUnderTest {
// ...
        public void addListener(Collaborator listener)
        { // ... }
        public void addDocument(String title, byte[] document)
        { // ... }
        public boolean removeDocument(String title)
        { // ... }
        public boolean removeDocuments(String[] titles)
        { // ... }
}
```

In order to Mock and verify it's behaviour we could write the following Example Test class:

```
import static org.easymock.EasyMock.*;
import org.junit.*;

public class ExampleTest {
private ClassUnderTest classUnderTest;
private Collaborator mock;
        }

//Initialization of the variables
@Before
public void setUp() {
//creation of the Mock interface we want to simulate
mock = createMock(Collaborator.class);

classUnderTest = new ClassUnderTest();
classUnderTest.addListener(mock);
        }

//Testing our class and its behavior
@Test
public void testAddDocument() {
//recording of the expected behavior
mock.documentAdded("New Document");
```

```
//switch the mock Object to a replay state
replay(mock);

classUnderTest.addDocument("New Document", new byte[0]);
//verify that the behavior of the Mock was as expected
verify(mock);
```

This is a very simple example of how a test could be achieved using EasyMock, of course, there are many other methods and different annotations. If you never used EasyMock, and you are interested on it, you can find all the necessary information on their webpage: http://www.easymock.org/

PowerMock

PoweMock extends **EasyMock** with static mocking and setting expectations on constructors. PowerMock uses a custom classloader and bytecode manipulation to enable mocking of static methods, constructors, final classes and methods, private methods, removal of static initializers and more. By using a custom classloader no changes need to be done to the IDE or continuous integration servers which simplifies adoption. Developers familiar with the supported mock frameworks will find PowerMock easy to use, since the entire expectation API is the same, both for static methods and constructors. PowerMock aims to extend the existing APIs with a small number of methods and annotations to enable the extra features.

The scope of this exercises is not to show how to do Unit tests using any of the frameworks described before, but how to use them inside AEM using maven and Sling, If you are interested on Unit testing you can take a look to the different homepages of the frameworks mentioned:

http://www.junit.org/
http://www.easymock.org/
http://code.google.com/p/powermock/

EXERCISE 10.1 - Unit Tests using Junit and Maven

Steps

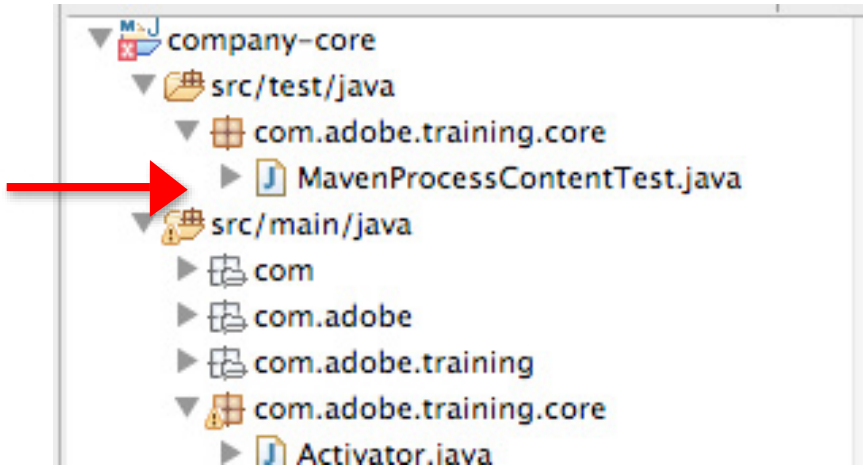
1. Create a class, ProcessContent, in *com.adobe.training.core*. It should contain 2 methods:
    - getContentPath – depends on JCR environment specifics
    - stripNonLettersOrNumbers – does not depend on JCR

```
package com.adobe.training.core;

import javax.jcr.Node;
import javax.jcr.RepositoryException;
import javax.jcr.Session;

public class ProcessContent {

	protected static final String CONTENT_NODENAME = "content";

	public String stripNonLettersOrNumbers(String in) {
		if(in != null) {
			return in.replaceAll( "[^\\p{L}\\p{N}]", "");
		} else {
			return null;
		}

	}

	public String getContentPath(Session session)  throws
RepositoryException {
		Node rootNode = session.getRootNode();
		if (rootNode.hasNode(CONTENT_NODENAME))  {
			Node contentNode = rootNode.getNode(CONTENT_NODENAME);
			return contentNode.getPath();

		} else {
			return rootNode.getPath();
		}
	}

}
```

2. In the company-core POM you can find the following corresponding references related to mock artifacts that we are going to use. The parent POM you received also contains those artifacts:

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.easymock</groupId>
    <artifactId>easymock</artifactId>
<scope>test</scope>
</dependency>
<dependency>
    <groupId>org.easymock</groupId>
    <artifactId>easymockclassextension</artifactId>
<scope>test</scope>
</dependency>
<dependency>
    <groupId>org.powermock</groupId>
    <artifactId>powermock-module-junit4</artifactId>
<scope>test</scope>
</dependency>
```

3. Create a class in the *src/test* tree of the company-core project:



4. Add a method to test ProcessContent.stripNonLettersOrNumbers. In this first example we will be using pure Junit. Notice the @Test annotation that tells Junit the procedure to test, as well as the assertEquals method, which will tell us if our class behaved as expected.

```
package com.adobe.training.core;

import static org.easymock.EasyMock.expect;
import static org.junit.Assert.assertEquals;
import static org.powermock.api.easymock.PowerMock.createMock;
import static org.powermock.api.easymock.PowerMock.replayAll;
import static org.powermock.api.easymock.PowerMock.verifyAll;
```

```
import javax.jcr.Node;
import javax.jcr.RepositoryException;
import javax.jcr.Session;

import org.junit.Test;

public class MavenProcessContentTest {
  @Test
  public void testStripNonLettersOrNumbers() {
    ProcessContent pc = new ProcessContent();
    assertEquals("abc1", pc.stripNonLettersOrNumbers("a_b!c.1"));
  }

}
```

5. Run "***mvn clean test***" to execute the test. You should see the following result:

```
-------------------------------------------------------
TESTS
-------------------------------------------------------
Running com.adobe.training.core.MavenProcessContentTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.049 sec
Results : Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
```

6. Let's change the Testing part in order to force an error in the test.

```
@Test
public void testStripNonLettersOrNumbers() {
        ProcessContent pc = new ProcessContent();
        assertEquals("abbc1", pc.stripNonLettersOrNumbers("a_b!c.1"));
}
```

7. Test again, now you should see an error

```
-------------------------------------------------------
TESTS
-------------------------------------------------------
Running com.adobe.training.core.MavenProcessContentTest
Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.058 sec <<< FAILURE!

Results :

Failed tests: testStripNonLettersOrNumbers(com.adobe.training.core. MavenTestSamplesTest):
expected:<ab[b]c1> but was:<ab[]c1>
Tests run: 1, Failures: 1, Errors: 0, Skipped: 0
```

And the project will refuse to build:

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.017s [INFO] Finished at: Fri May 18 16:26:36 CST 2012 [INFO] Final Memory: 13M/81M [INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.apache.maven. plugins:maven-surefire-plugin:2.12:test (default-test) on project company-core: There are test failures.
[ERROR]
[ERROR] Please refer to /Users/sarrivillaga/Adobe/CQ56AdvDev/ sampleproject/ company-core/target/surefire-reports for the individual test results.
[ERROR] -> [Help 1
```

EXERCISE 10.2 - Unit tests with Junit, EasyMock, PowerMock and Maven

We will now test the method ProcessContent.getContentPath. This method uses an instance of a Node and a Session Object.

We will make use of PowerMock to mock those two objects (note that we are not mocking any static object and that we are not setting any expectation on the constructor of those classes, so we could have used EasyMock instead to Mock the objects) We will use EasyMock to tell the different calls that we are expecting and the values that should be returned from those calls.

We will switch the Mock Object to replay state by using the PowerMock method *replayAll()* which replays all classes and mock objects known by PowerMock. This includes all classes that are prepared for test using the @PrepareForTest or @PrepareOnlyThisForTest annotations as well all mock instances created by PowerMock.

We will proceed to create a new ProcessContent, we still use Junit to test that our class method getContentPath(Session) retrieves the correct path ("/content"):

```
assertEquals("/content", pc.getContentPath(SESSION_MOCK));
```

We will verify those expectations with verifyAll().

Steps

1. Add a new ProcessContent test method to MavenProcessContentTest. Let's create our method testGetContentPath() to test the ProcessContent.getContentPath() method.

```
@Test
public void testGetContentPath()
throws RepositoryException {
        //create a mock repository session and prepare the expected method
calls
  final Session SESSION_MOCK = createMock(Session.class);
  final Node ROOT_NODE_MOCK = createMock(Node.class);
  expect(SESSION_MOCK.getRootNode()).andReturn(ROOT_NODE_MOCK);
                    expect(ROOT_NODE_MOCK.hasNode(ProcessContent.CONTENT_NODENAME)).
andReturn(true);

  final Node CONTENT_NODE_MOCK = createMock(Node.class);
                    expect(ROOT_NODE_MOCK.getNode(ProcessContent.CONTENT_NODENAME)).
andReturn(CONTENT_NODE_MOCK);
  expect(CONTENT_NODE_MOCK.getPath()).andReturn("/content");
  replayAll();

  ProcessContent pc = new ProcessContent();
  assertEquals("/content", pc.getContentPath(SESSION_MOCK));

  //verify that all expected methods calls have been executed
  verifyAll();
}
```

2. Run "mvn clean test" again. You should see this time your two tests being executed successfully, and the project being built.

```
-------------------------------------------------------
TESTS
-------------------------------------------------------
Running com.adobe.training.core.MavenProcessContentTest
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.148 sec
Results : Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO] ----------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ----------------------------------------------------
[INFO] Total time: 3.406s
[INFO] Finished at: Tue May 22 15:11:52 CST 2012
[INFO] Final Memory: 13M/81M
[INFO]                    -------------------------------------------------------------------
-------------------------------------------
```

EXERCISE 10.3 - Running tests on the server (Sling-based tests)

Next, we will run tests within the server (Sling-based tests). For that you will need to deploy the Sling testing bundles to your OSGi runtime (in the Adobe AEM Web console). This are the bundles that will allow you to perform those tests directly into the server.

This exercise has been tested with version 1.0.6 of the bundles.

1. Your trainer will give the bundles to you. Otherwise you can download the latest version at http://sling.apache.org/site/downloads.cgi

JCR WebDAV	2.1.0	jar (asc, md5)	tar.gz (asc, md5) zip (asc, md5)
JCR DavEx	2.1.0	jar (asc, md5)	tar.gz (asc, md5) zip (asc, md5)
JCR Web Console Plugin	1.0.0	jar (asc, md5)	tar.gz (asc, md5) zip (asc, md5)
JUnit Core	1.0.6	jar (asc, md5)	zip (asc, md5)
JUnit Remote Tests Runners	1.0.6	jar (asc, md5)	zip (asc, md5)
JUnit Scriptable Tests Provider	1.0.6	jar (asc, md5)	zip (asc, md5)
Mime Type Service	2.1.4	jar (asc, md5)	zip (asc, md5)
Launchpad API	1.1.0	jar (asc, md5)	zip (asc, md5)
Launchpad Base	2.4.0	jar (asc, md5)	zip (asc, md5)
Launchpad Base - Application Launcher	2.4.0	jar (asc, md5)	zip (asc, md5)

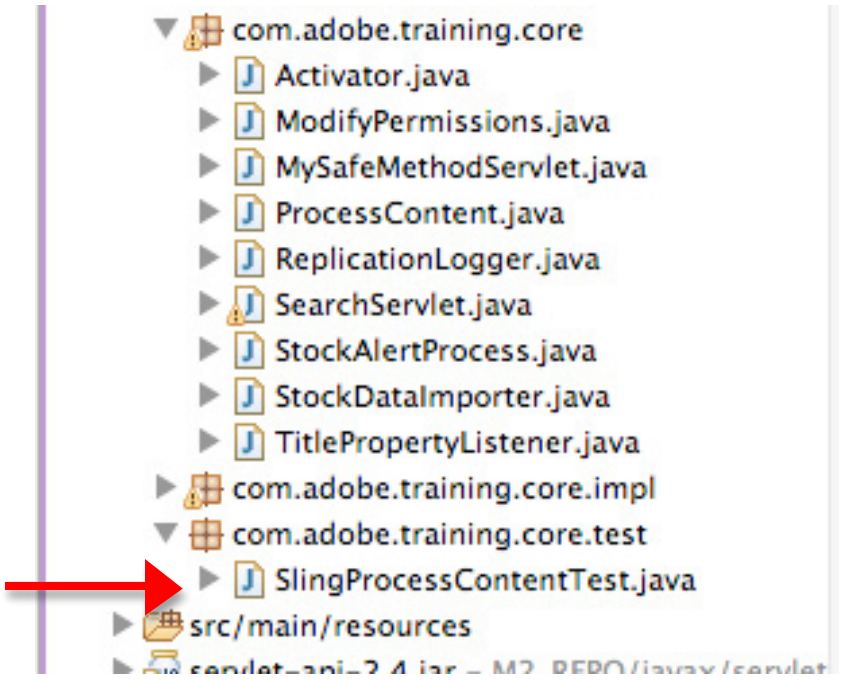
2. You should see your bundle's reference's in your parent's POM as well as in your core bundle's POM:

```
<dependency>
    <groupId>org.apache.sling</groupId>
    <artifactId>org.apache.sling.junit.core</artifactId>
    <version>1.0.6</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.sling</groupId>
    <artifactId>org.apache.sling.junit.remote</artifactId>
    <version>1.0.6</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.sling</groupId>
    <artifactId>org.apache.sling.junit.scriptable</artifactId>
    <version>1.0.6</version>
    <scope>provided</scope>
</dependency>
```

You can see also that we have included in your company-core POM a regular expression that defines which class names are to be executed as tests:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.felix</groupId>
            <artifactId>maven-scr-plugin</artifactId>
        </plugin>
        <plugin>
            <groupId>org.apache.felix</groupId>
            <artifactId>maven-bundle-plugin</artifactId>
            <extensions>true</extensions>
            <configuration>
                <instructions>
                    <!-- Export packages that should be visible to other bundles and JSPs -->
                    <Export-Package>
                        com.adobe.training.core.*, com.adobe.training.utils.*
                    </Export-Package>
                    <Import-Package>*;resolution:=optional</Import-Package>
                    <Embed-Dependency>*;scope=compile|runtime</Embed-Dependency>
                    <Sling-Test-Regexp>.*adobe.*Test</Sling-Test-Regexp>
                </instructions>
            </configuration>
        </plugin>
    </plugins>
```

3. Create a test class to test the ProcessContent class in the src/main part of the project. Put it in the com.adobe.training.core.test package:



```
package com.adobe.training.core.test;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import javax.jcr.Repository;
import javax.jcr.RepositoryException;
import javax.jcr.Session;

import org.apache.sling.jcr.api.SlingRepository;
import org.apache.sling.junit.annotations.SlingAnnotationsTestRunner;
import org.apache.sling.junit.annotations.TestReference;
import org.junit.Test;
import org.junit.runner.RunWith;

import com.adobe.training.core.ProcessContent;

@RunWith(SlingAnnotationsTestRunner.class)
public class SlingProcessContentTest {

}
```

4. Add a method testGetcontentPath to test ProcessContent.getContentPath() (but running within the repository).

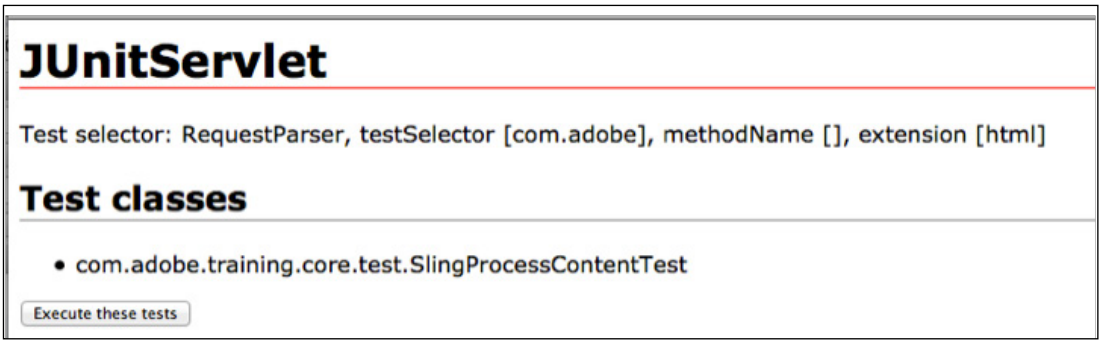
For that we first use the annotation @RunWith. When a class is annotated with @RunWith or extends a class annotated @RunWith, JUnit will invoke the class it references to run the tests in that class instead of the runner built into JUnit. So, in our case, SlingAnnotationsTestRunner will be used to run our tests. Because of this we can use @TestReference, to access OSGi services.

In order to have access to the SlingRepository (and be able to run our classes within it) we use @ TestReference. @TestReference will inject the service of our SlingRepository in our code; it will be made available to our classes by the Service Component Runtime.

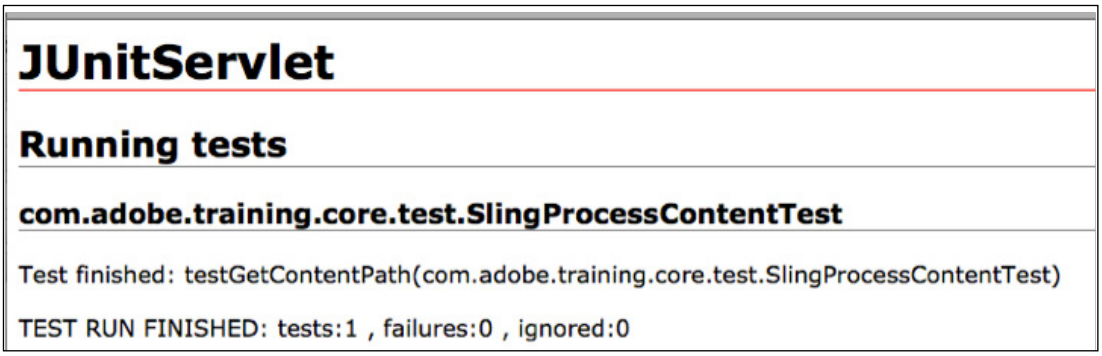
The SlingRepository extends the standard JCR repository interface with two methods: getDefaultWorkspace() and loginAdministrative(String). This method eases the use of a JCR in a Sling application. The default (or standard) workspace used by the application may be configured and application bundles may use a simple method to get an administrative session instead of being required to provide their own configuration of administrative session details.

```
package com.adobe.training.core.test;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import javax.jcr.Repository;
import javax.jcr.RepositoryException;
import javax.jcr.Session;

import org.apache.sling.jcr.api.SlingRepository;
import org.apache.sling.junit.annotations.SlingAnnotationsTestRunner;
import org.apache.sling.junit.annotations.TestReference;
import org.junit.Test;
import org.junit.runner.RunWith;

import com.adobe.training.core.ProcessContent;

@RunWith(SlingAnnotationsTestRunner.class)
public class SlingProcessContentTest {

	@TestReference
	private SlingRepository repository;

	@Test
	public void testRepoName() {
		assertTrue(repository.getDescriptor(Repository.REP_NAME_DESC).equals("Apache
Jackrabbit Oak"));
	}

	@Test
	public void testGetContentPath() {
		Session adminSession= null;
		try {
			adminSession = repository.loginAdministrative(null);
			ProcessContent pc = new ProcessContent();
			assertEquals("/content",pc.getContentPath(adminSession));
		} catch (RepositoryException e) {


		}finally {
			if(adminSession != null) {
				if (adminSession.isLive())
					adminSession.logout();
			}
		}

	}
}
```

5. Deploy the bundle to the repository (mvn clean install -P bundle). Open http://localhost:4502/system/sling/junit/com.adobe.training.html and execute the tests.



You should see the following result:



Server-side tests depending on an injected environment object

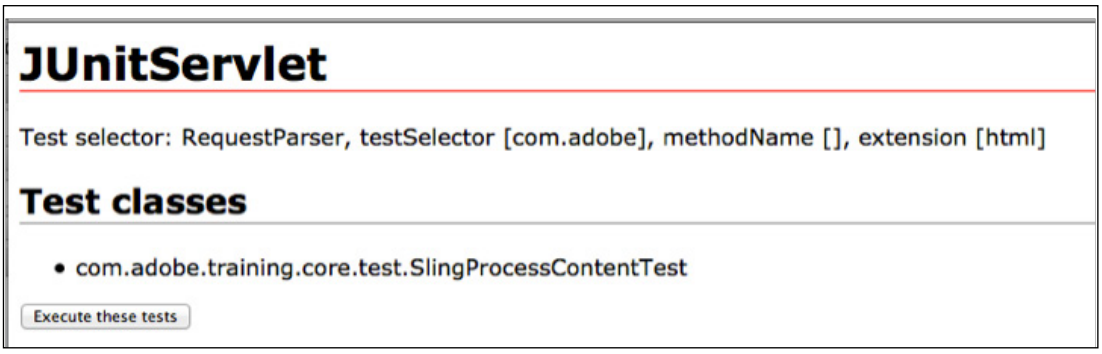
Let's add a test that depends on injected environment objects. As stated before, in order to have access to the SlingRepository (and be able to run our classes within it) we use @TestReference. @TestReference will inject the service of our SlingRepository in our code; it will be made available to our classes by the Service Component Runtime.

1. Create the...

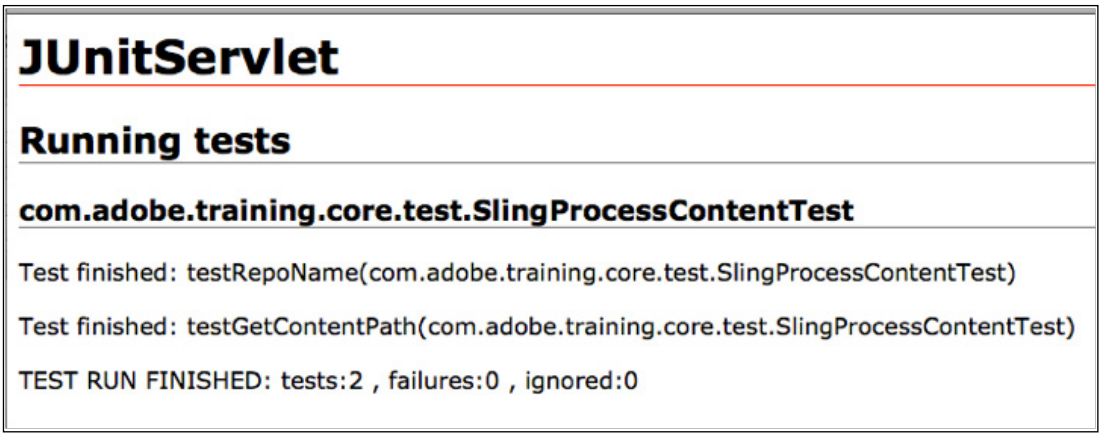
```
@RunWith(SlingAnnotationsTestRunner.class)
public class SlingProcessContentTest {

    @TestReference
    private SlingRepository repository;

    @Test
    public void testRepoName() {
        assertTrue(repository.getDescriptor(Repository.REP_NAME_DESC).equals("Apache Jackrabbit Oak"));
    }

    @Test
    public void testGetContentPath() {
}
```

We then use JUnit in order to check that the repository attribute REP_NAME_DESC correspond to the String "CRX".

2. Deploy the bundle to the repository (mvn clean install -P bundle), open http://localhost:4502/system/sling/junit/com.adobe.training.html and execute the tests.



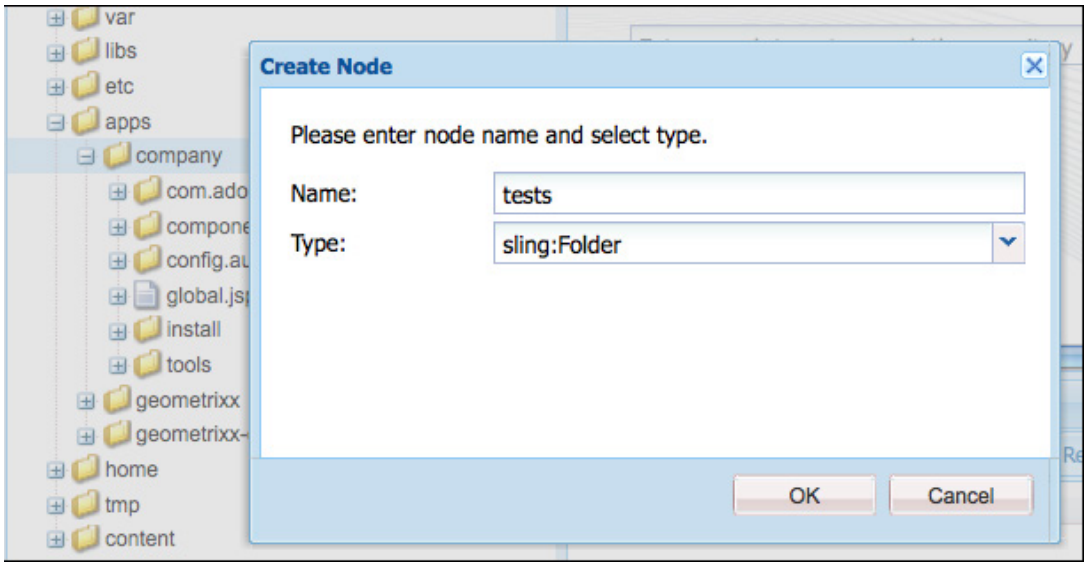
You should see the following result:

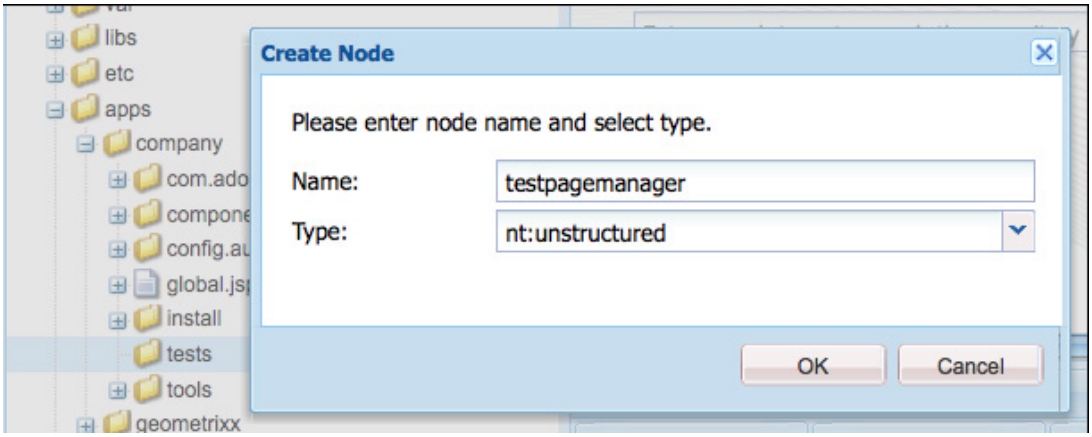


Running scriptable server side tests

Scriptable server side tests are usually made by creating a node with the mixin type sling:Test. This node has to have a property sling:resourceType pointing to the script used for testing purposes. Then to execute the test, just ask for this resource in AEM.

1. Open CRXDE Lite and add the node "tests" of node type sling:Folder to /apps/company





2. In the tests folder that you just created, create the node testpagemanager ...

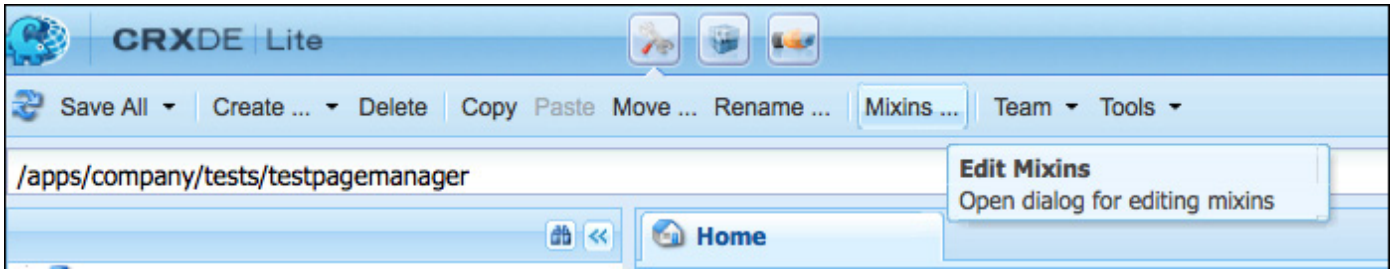
3. Add the following property to the testpagemanager node:

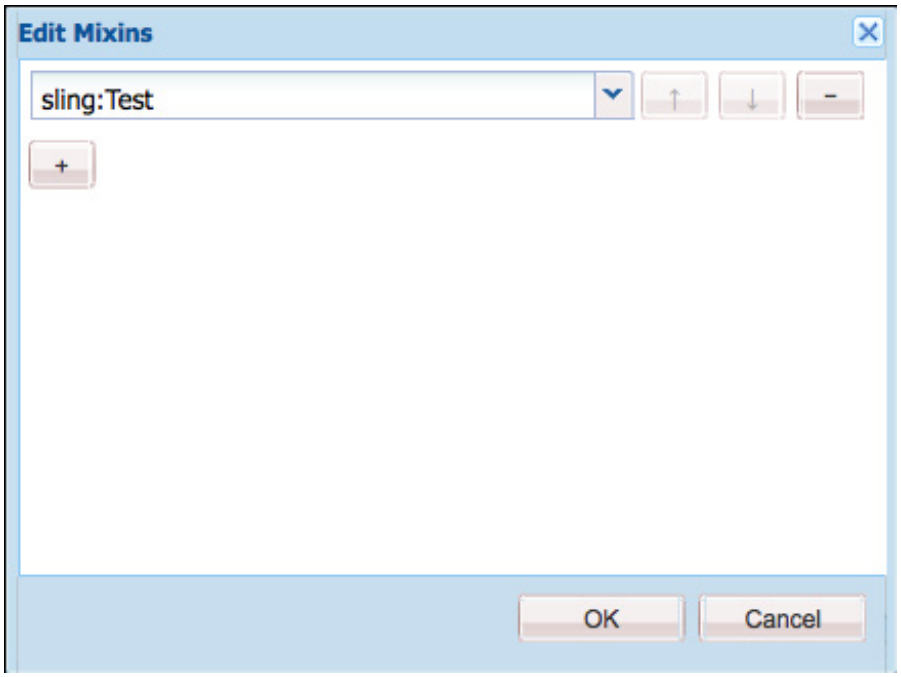
   Name: sling:resourceType

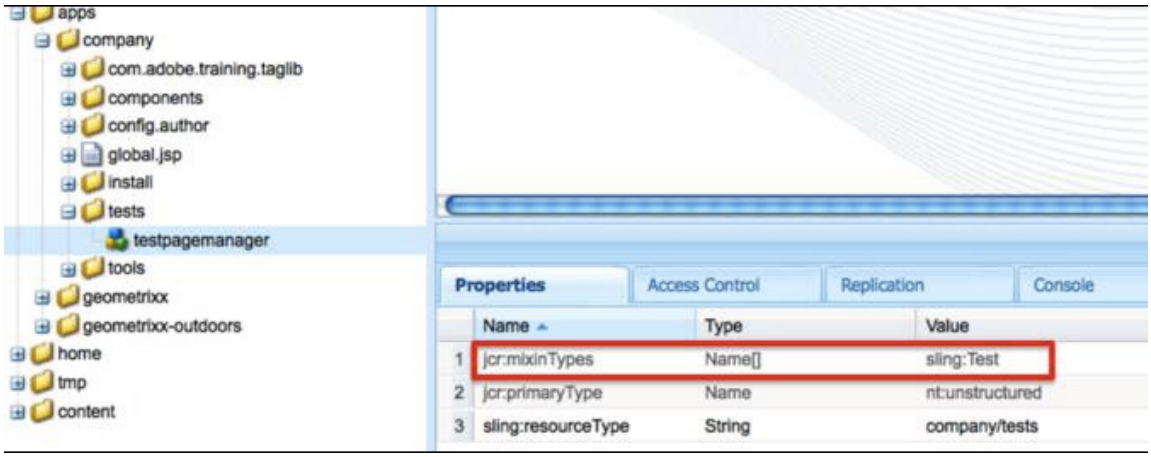
   Type: String

   Value: company/tests

4. Add the mixin type sling:Test to the testpagemanager node. Note: You may have to refresh the browser screen before the sling:Test mixin will apprear in the mixin dropdown list.

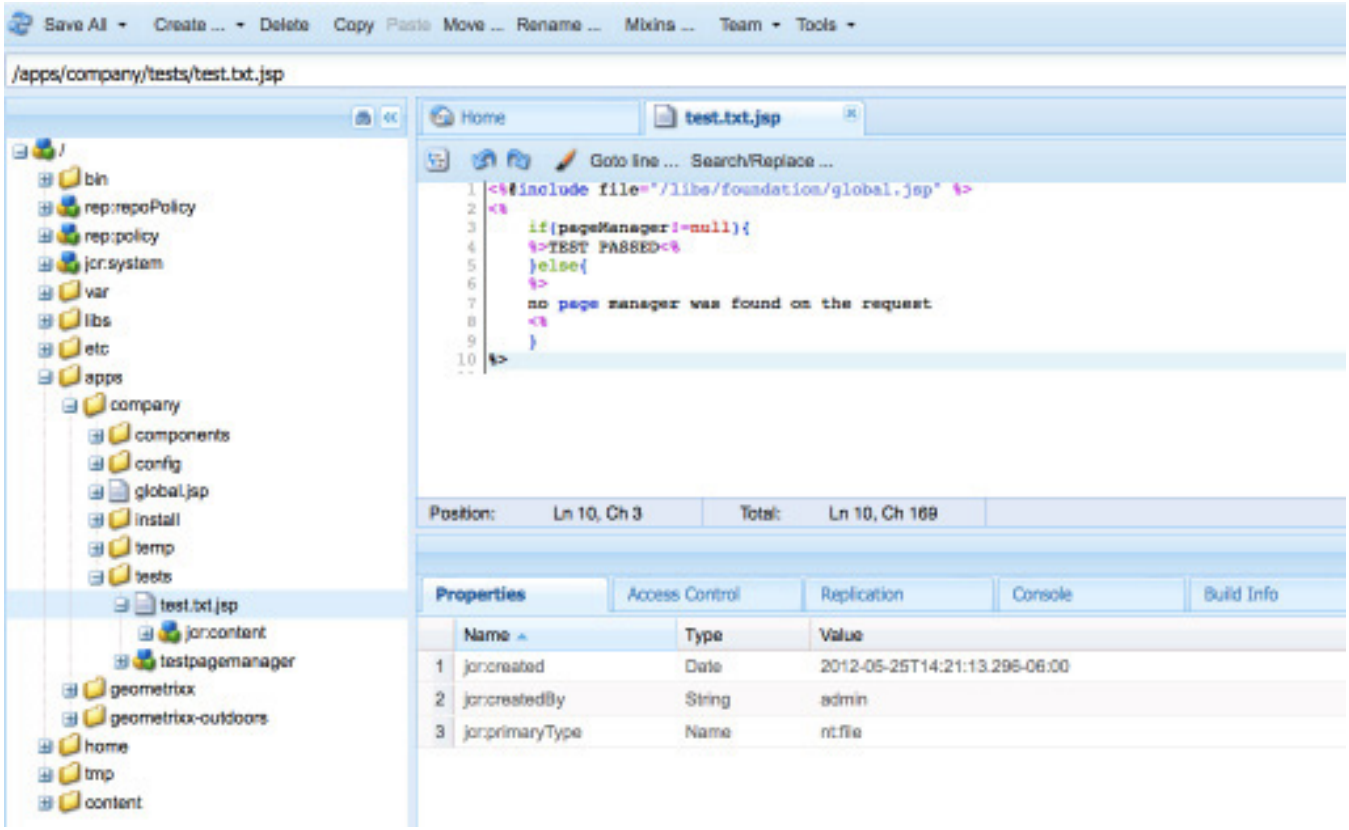






5. The test script shall test if /libs/foundation/global.jsp actually puts the PageManager object on the request. Create the a jsp called test.txt.jsp (notice the selector and the extension used) and add the following code:

```
<%@include file="/libs/foundation/global.jsp" %>
<%
        if(pageManager!= null) {
                %> TEST PASSED <%
        } else {
           %> no page manager was found on the request <%
        }
%>
```



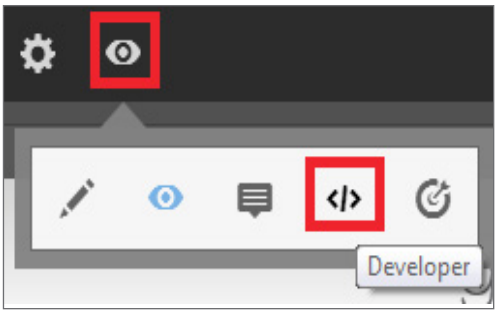
6. Execute the test by browsing to http://localhost:4502/apps/company/tests/testpagemanager.test.txt:

Testing framework

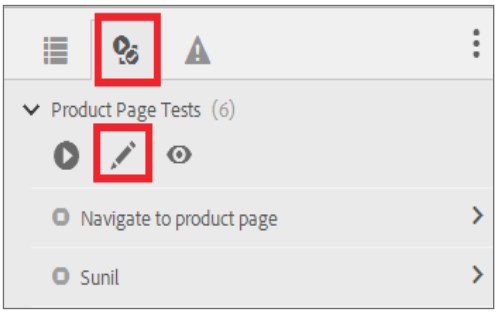
AEM 6.0 provides you with a UI testing framework that you can access in the Developer mode. From the Touch-Optimized GUI, you can run the test cases. You can also update your test scripts by directly navigating to the test scripts folder using CRXDE Lite.

EXERCISE 10.4 - Create a test suite in AEM

1. Using Siteadmin, navigate to one of the Geometrixx Outdoors' pages.
2. Click the **Select Another Mode** button and select the Developer mode.



3. Click the Toggle Side panel button to display the components.
4. Go to the Test tab.



5. Click the Edit button to navigate to the test scripts using CRXDE Lite.
6. In the test folder, create a new file named, `TrainingPageTests.js`.
7. Create a new test suite by adding the following code at the top of the `TrainingPageTests.js` file.
   A test suite has many test cases. In other words, a test suite is a group of related test cases.

   new hobs.TestSuite("Training Page tests", {path:"/apps/geometrixx-outdoors/tests/ProductPageTests.js", register: true})
8. Now add a test case under the test suite as follows:

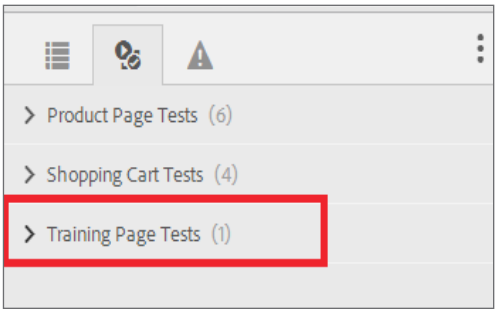
.addTestCase(new hobs.TestCase("First training test case")

.navigateTo("/content/geometrixx-outdoors/en/men/shorts/jola-summer.html")
.asserts.location("/content/geometrixx-outdoors/en/men/shorts/jola-summer.html", true)

.asserts.visible("article.product[data-sku='mnapjs.1-S']", true)
.asserts.visible("article.product[data-sku='mnapjs.2-S']", false)
)

9. Analyze the test case. You are doing the following:
   - Instructs the framework to navigate to `jola-summer.html` page.
   - Checks if the location changes to `jola-summer.html`. Test case is marked as fail if it doesn't.
   - Checks if the SKU attribute, S, is present. If yes, mark as Pass.
   - Checks if the SKU attribute, S, is present. If Yes, mark as Fail. (You just want see how a test case can fail.)

10. Save the file in CRXDE Lite.
11. Open `js.txt` and add the following entry: `TrainingPageTest.js`
12. Return to the product page and refresh it.
13. Observe that the Test Suite that you added is displayed there.



14. Expand the test suite and click Run.
15. Note that the page is refreshing with `jola-summer.html`.
16. Observe the test results.

17. Note that the last test case fails, because the size S was visible. You have instructed to make the test case pass if this element is not present.

11

Deployment and Packaging

Packaging

Once you have created application packages, you need a process for deploying these to other environments. On a simple level, this can just be a case of moving the package zip file to a new environment, uploading it to the repository, and importing its contents. Alternatively, the package can be activated using the tools section of the AEM Admin Console. However some consideration should be given to managing this process, to ensure that your applications can be deployed efficiently and without any problems.

Considerations

- Make sure your application code is separate from the configuration, as you may need to use different configurations with the same application for different environments. Creating separate packages makes it easy to deploy the appropriate configuration for each environment.

- Make sure your application code does not depend on specific content, as this may not be present in all environments.

- Code packages will be created in the development environment, and will then travel from development, to test, then to production. Content will generally travel from production, to test and then to development to be used as test content, so separate packages and processes may be needed.

There are two approaches that can be taken when creating packages for deployment, so that the configuration information can be configured separately from the application code. Either two separate packages may be created for application and for configuration information, or one package can be used with separate configuration runmodes.

Packaging - Style 1

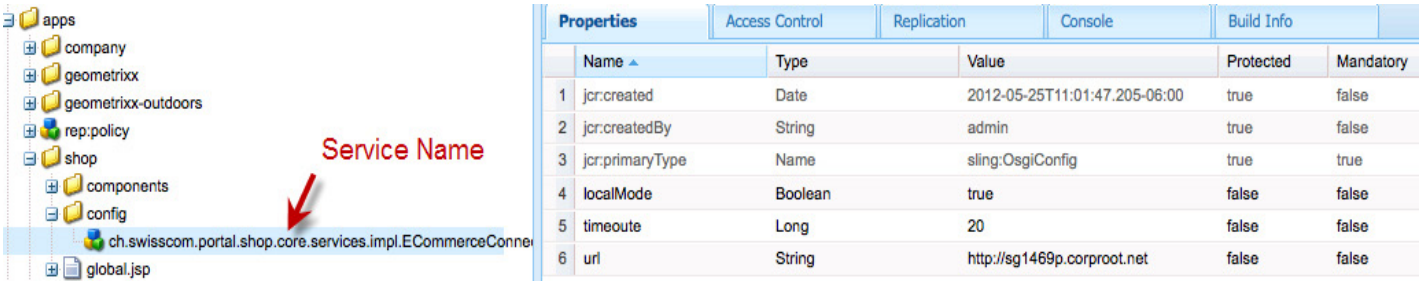
Deploy 2 packages:

- One with the application code.
- One with the environment specific configuration for development or test or production.

By arranging the packages like this, your tested application will not have to be changed when it is deployed in production. You can simply adjust the environment configuration to suit the environment.

How To Create The config Package

- Use the Apache Felix mechanism.
- Nodes deployed into folders called "config" with a node type of `Sling:OsgiConfig` will get automatically installed into the OSGi configuration.
- Use this technique for application-specific settings and for system settings.



Packaging - Style 2

- Package the configuration for all relevant environments into one package together with the application code.
- Use different runmodes to select the correct configuration.

Runmodes

- AEM has built-in author or publish runmodes (Do NOT remove these!).
- You can add multiple additional custom runmodes if required.

For example, you can configure runmodes for...

- ***Environment***: local, dev, test, prod
- ***Location***: berlin, basel, timbuktu
- ***Company***: acme, partner, customer
- ***Special system type***: importer

Setting Runmodes

It is possible to define specific run-mode(s) a specific instance should run on. By default an author instance runs on run-mode author and a publish instance runs on run-mode publish. It is possible to define several run-modes for one instance, for example `author, foo` and `dev`.

These run-modes have to be set as VM options. For example, from the command line:

```
java -Dsling.run.modes=author,foo,dev -Xmx1024m -jar cqquickstart-5.6.0.jar
```

or in the start script:

```
# default JVM options
CQ_JVM_OPTS='-Dsling.run.modes=author,foo,dev'
```

or by adding entries to the crx-quickstart/conf/sling.properties file for example:

```
sling.run.modes=author,dev,berlin
```

Configurations per run mode

To create separate configuration settings per runmode, folder names in the form `config.<runmode>` are used, e.g. "config.publish":

```
/apps/myapp/config.publish
```

For systems with run modes `publish` and `berlin`:

```
/apps/myapp/config.publish.berlin
```

Configurations For Different Runmodes

Some examples of configuration settings that may be needed for different runmodes:

Different mailserver configurations per location:

- `config.basel/com.day.cq.mailer.DefaultMailService`
- `config.berlin/com.day.cq.mailer.DefaultMailService`

En-/Disabling debugging per environment:

- `config.prod/com.day.cq.wcm.core.impl.WCMDebugFilter`
- `config.dev/com.day.cq.wcm.core.impl.WCMDebugFilter`

Configurations Per Run Mode

When using different configurations for separate runmodes, the following apply:

- Partial configurations are not supported.
- The configuration with the most matching runmodes wins.

To avoid unexpected results:

- Always set all properties to avoid confusion.
- Use a type indicator (e.g. {Boolean}, {String}, etc.) in every property.

Deployment

In addition to manual deployment of packages, you can automate deployment with the Package Manager API. For more details, see:

> http://dev.day.com/docs/en/crx/current /how_to/package_manager.html#Package%20Manager%20HTTP%20Service%20API

You can also activate packages in the Tools section of the WCM user interface to install them on all publish servers.

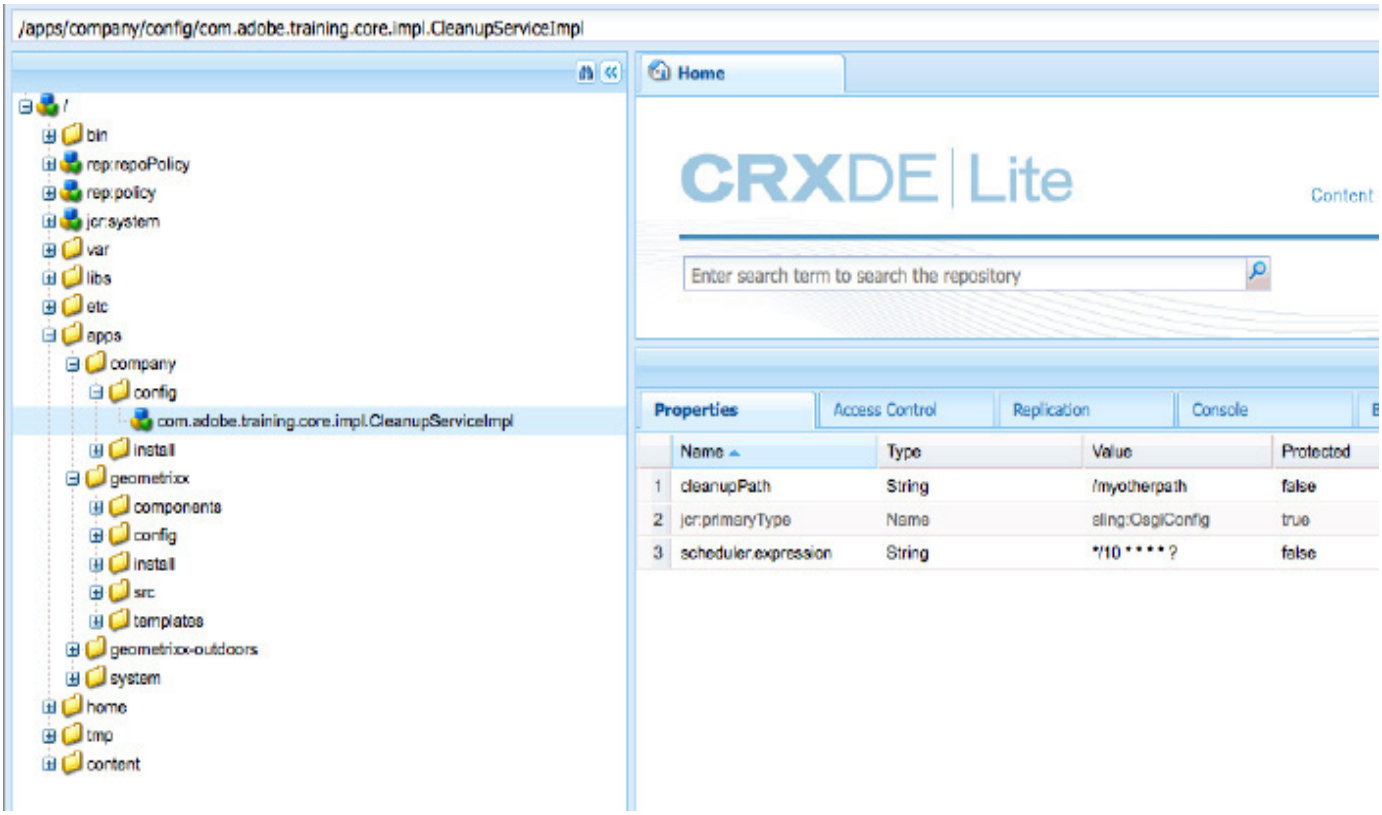
EXERCISE 11.1 - Configuration Package

Goal

The aim of this exercise is to modify the custom configuration created earlier, and provide different settings on author and on publish instances. We will create a package, activate it, and check the results. If time allows you can also checkout the configuration into a new config project.

Steps

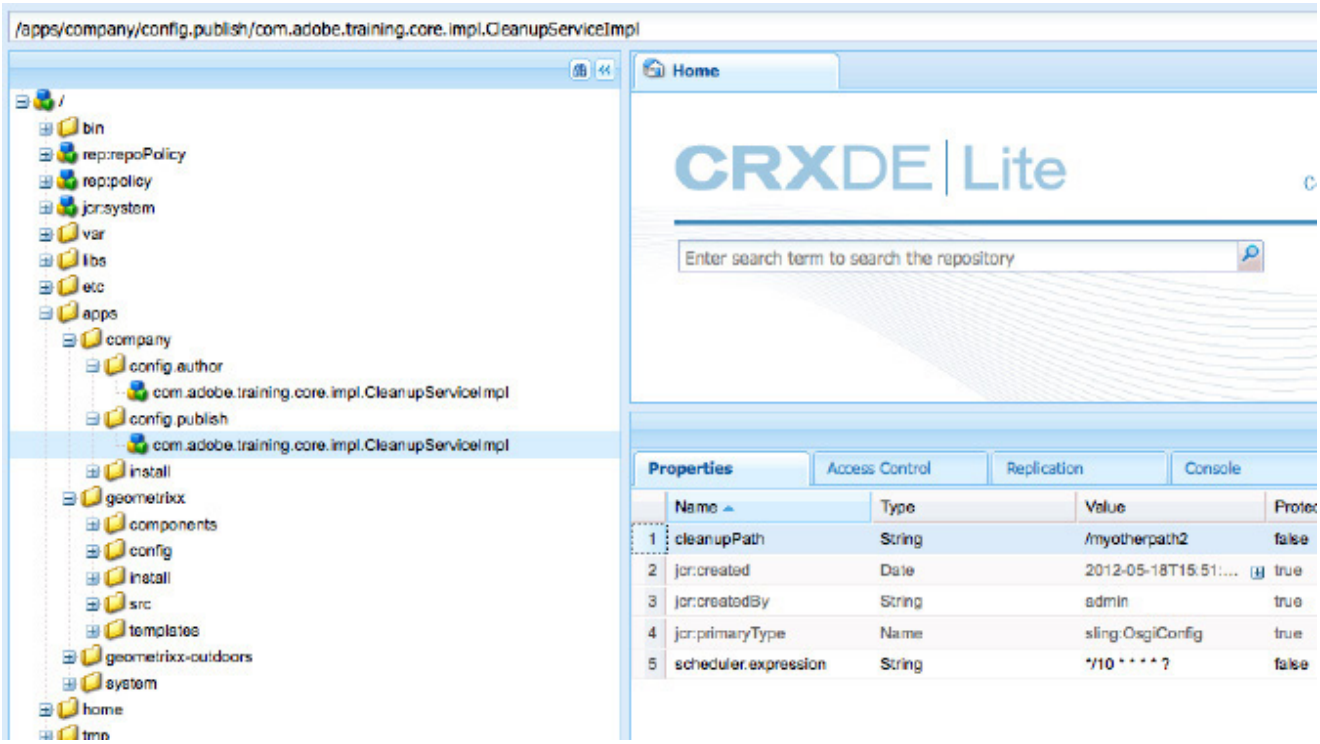
1. Start by reexamining the setting we created earlier under the node config.author:



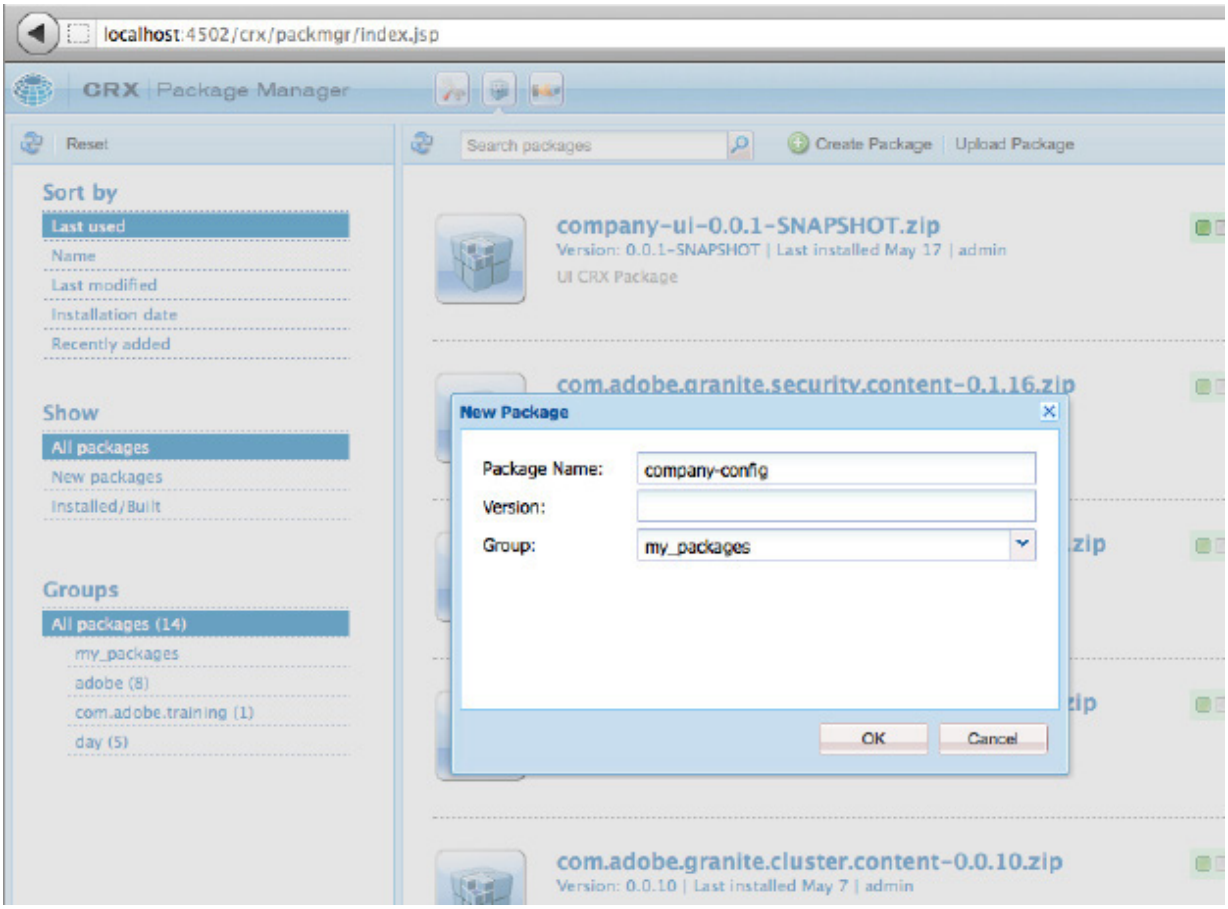
2. We have been using mvn clean install –P bundle, which rebuilds and deploys the jar file, but does not rebuild the company-ui content package. We will need this updated bundle to be in the package when we install the package on the publish instance. Build and deploy the core bundle to update the company-ui package that has the bundle in it.

   mvn clean install –P full

3. Copy the `config.author` folder, and paste a new copy into the same location, then rename the new folder name to `config.publish`. Now change the setting values for `config.publish` so that you can check which one has been loaded. Notice that the value for the cleanup Path property has changed:



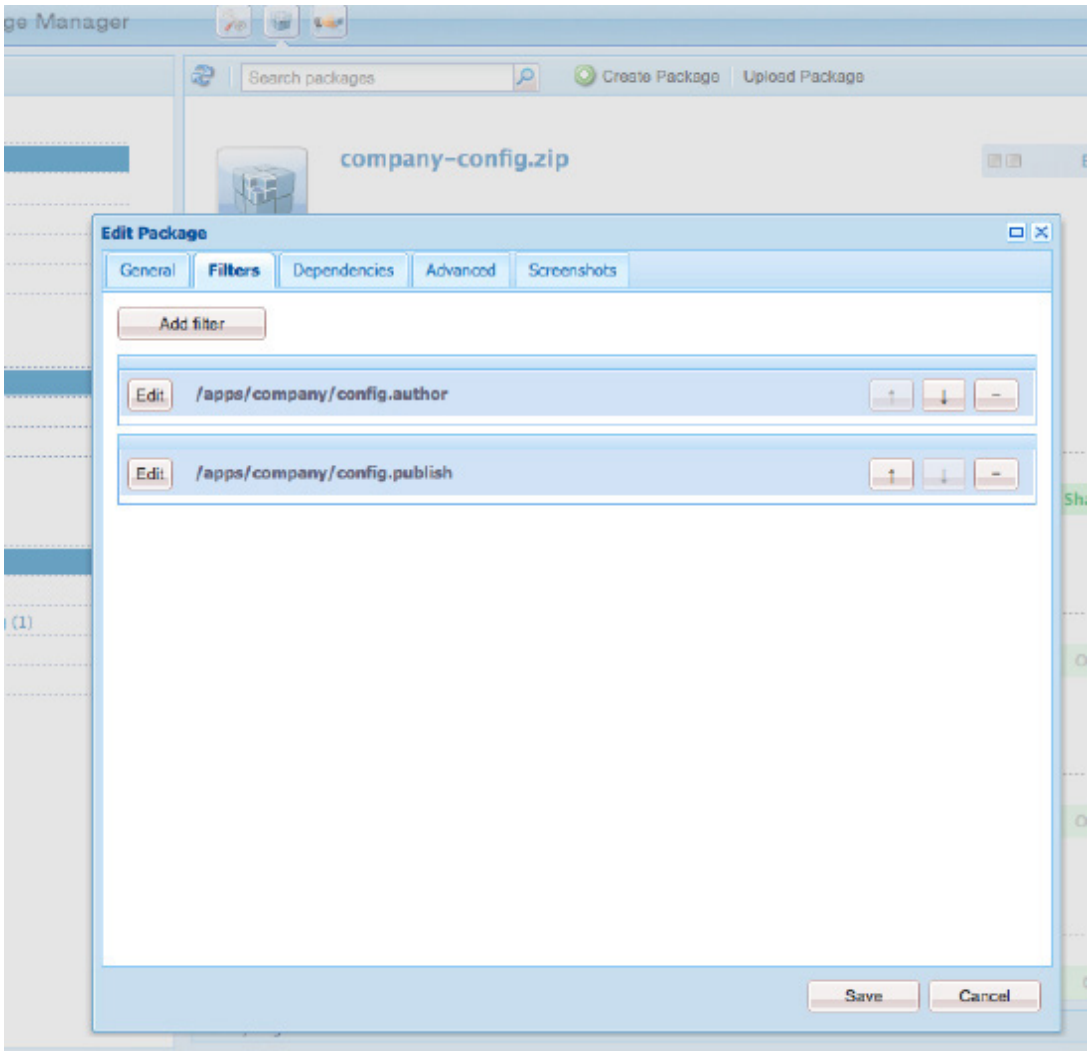
4. Create a package containing the `config.author` and `config.publish` folders. Go to http://localhost:4502/crx/packmgr/index.jsp and click on `create package`:



5. Add the package name company-config, and group name `my _ packages`, then click on OK.

6. Open the package, click on the Edit button, and add the two filters `/apps/company/config.author` and `apps/company/config.publish`:

7. Build the package.

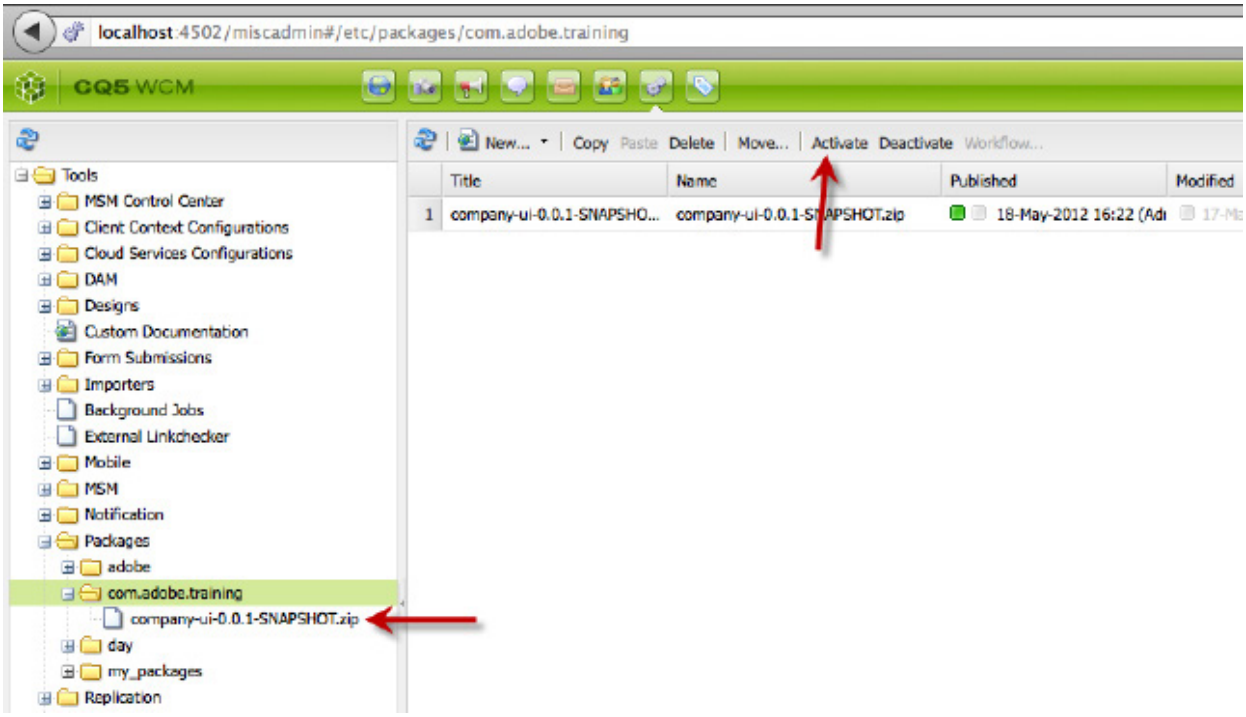


8. Go to sampleproject/company-ui/src/main/content/jcr_root and execute the command: vlt up to copy the new configuration information to the file system.

9. Start the publish server, and in the author Tools console, activate the application package `company-ui-0.0.1-SNAPSHOT.zip` and then afterwards the config package `company-config.zip`.

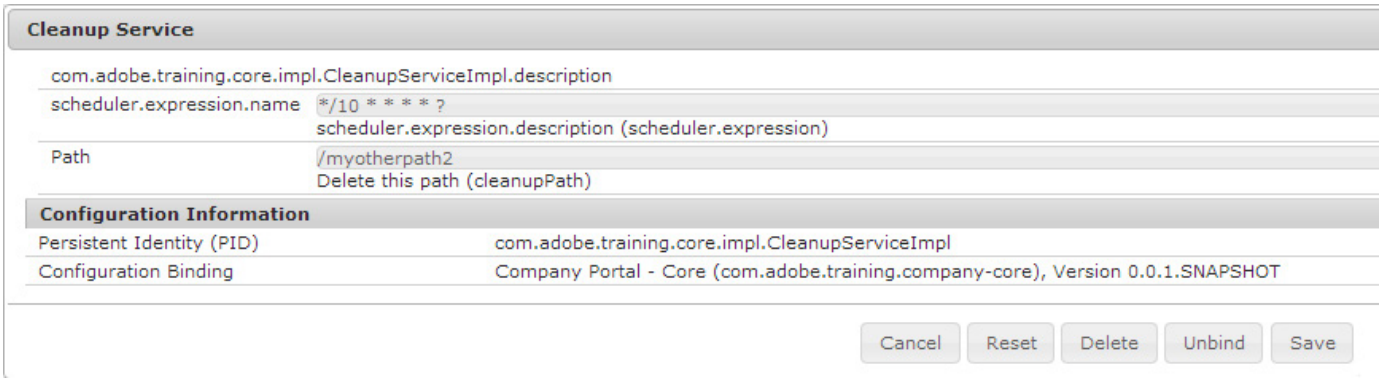
NOTE: You must activate the application package first and then the configuration package.

Congratulations! You have successfully deployed your application, along with a custom configuration for the publish environment. If time allows, try to checkout the configuration into a new config project with vlt.



Check the functionality of the package (you can test that the servlet you created at the beginning of the course is now running on the publish server, by going to http://localhost:4503/bin/company/repo/). Also check the configuration on the publish server's Adobe AEM Web console and ensure that you are now using the config.publish configuration.

Congratulations! You have successfully deployed your application, along with a custom configuration for the publish environment. If time allows, try to checkout the configuration into a new config project with vlt.



12

Dispatcher, Reverse Replication

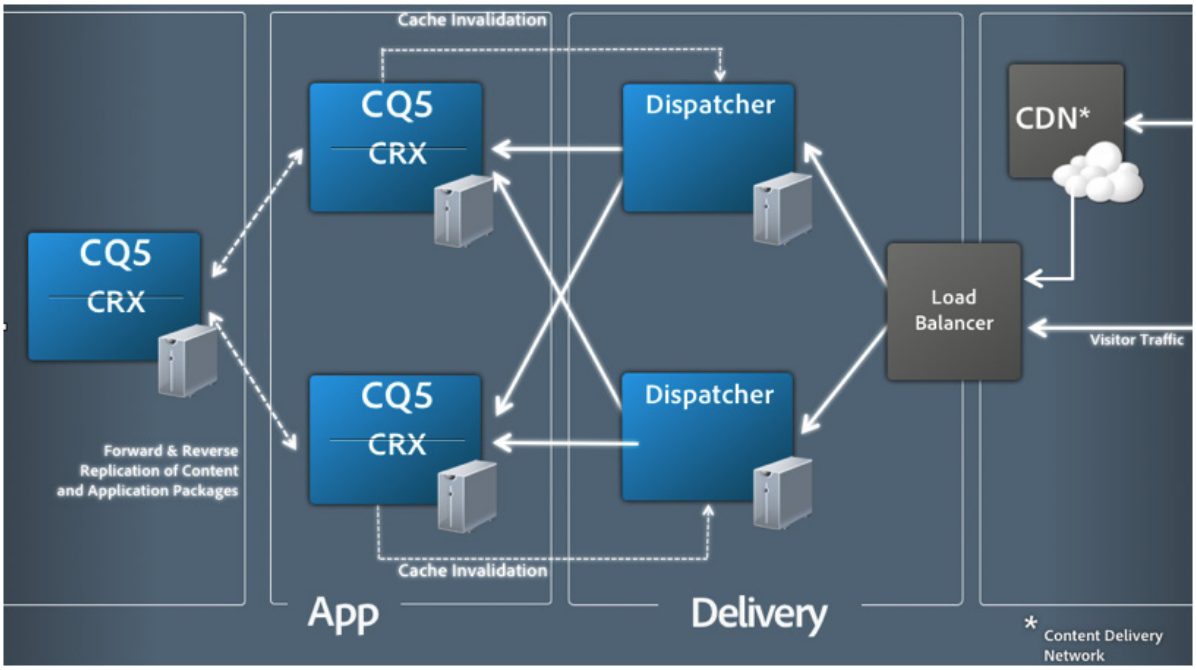
Dispatcher

The Dispatcher is Adobe's caching and/or load balancing tool. Using the Dispatcher also helps protect your application server from attack. Therefore, you can increase protection of your AEM instance by using the Dispatcher in conjunction with an industry-strength web server.

The Dispatcher helps realize an environment that is both fast and dynamic. It works as part of a static HTML server, such as Apache, with the aim of:

- Storing (or "caching") as much of the site content as is possible, in the form of a static website
- Accessing the layout engine as little as possible

The Dispatcher contains mechanisms to generate, and update, static HTML based on the content of the dynamic site. You can specify in detail which documents are stored as static files and which are always generated dynamically.



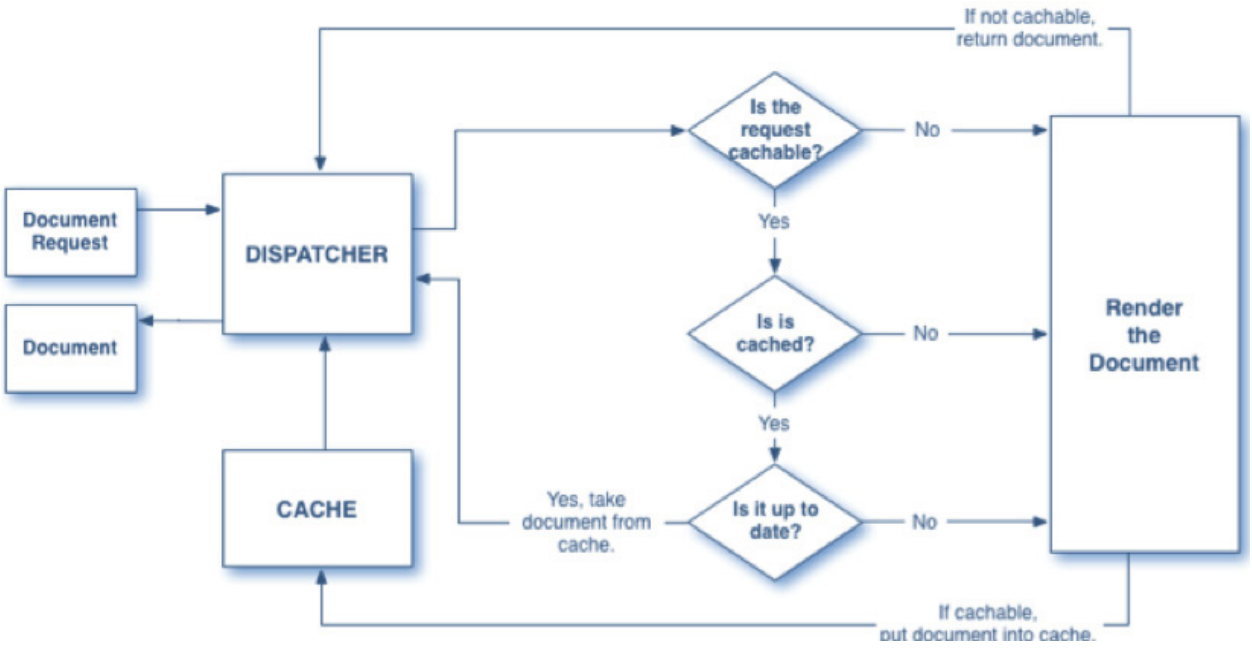
The Basics Revisited

The Dispatcher is a web server plugin. The process for deploying the Dispatcher is independent of the web server and OS platform chosen. The process for installing and configuring the Dispatcher is as follows:

- Install the supported web server of your choice according to the documentation for that web server
- Install the Dispatcher module appropriate to the chosen web server and configure the web server accordingly
- Configure the Dispatcher
- Integrate with AEM to update the cache when the content in AEM changes (Configure the Dispatcher Flush Agent)

Caching in dispatcher is equivalent to files in a filesystem. The Dispatcher stores the cached files on the web server as if they were part of a static website.

How the Dispatcher Returns Documents



Cache Invalidation (Expiration)

Cache invalidation needs to be triggered by author or publish (usually as a result of an activation). The Dispatcher is told by the Dispatcher Flush Agent to invalidate the cache. The Dispatcher then touches the .stat file (does not remove content file), creating a timestamp against which new document requests will be checked. Cache invalidation is hierarchical! Typically a level in the cache is chosen and all documents below that level are invalidated.

The Dispatcher's Role in AEM Projects

The Dispatcher contains mechanisms to generate, and update, static HTML based on the content of the dynamic site. You can specify in detail which documents are stored as static files and which are always generated dynamically.

A good practice is to think about the Dispatcher and caching right from the start of your project. Make it an integral part of your application and content architecture. The content hierarchy can/should be adapted to cache expiration considerations. The default: expire at the level of the language root. For example,

`/content/mysite/en` (i.e. one language tree)

Configuring the Cache - dispatcher.any

By default the Dispatcher configuration is stored in ***dispatcher.any,*** though you can change the name and location of this file during installation.

What to Cache - the Rules Section

It is in the */rules* section of the ***dispatcher.any*** file that you specify which documents are cached. If you do not have dynamic pages (beyond those already excluded by the Dispatcher's own rules), you can let the Dispatcher cache everything.

By default the following requests are not cached by the Dispatcher:

- Requests that do not return http code 200
- requests with suffixes
- requests with request parameters (i.e. "?")
- programmatically: send http header

  `response.setHeader`("Dispatcher", "no-cache");

The ***/invalidate*** section defines a list of all documents that are automatically rendered invalid after any content update.

```
/cache
                {
                # Cache configuration
/rules
            {
                        /0000
             {
              /glob "*"
              /type "allow"
              }
            }
               /invalidate
            {
            /0000
            {
            /glob "*"
            /type "deny"
             }
            /0001
             {
             /glob "*.html"
            /type "allow"
            }
}
```

Denying Access - The Filter Section

Usually, dispatcher is also used to restrict external access to resources you need to be aware of this when coding your application. Using filters, you can specify which requests are accepted by the

Dispatcher module. All other requests are sent back to the server, where they are offered to the other modules that run on the web server.

```
# the glob pattern is matched against the first request line
/filter
  {
  # deny everything and allow specific entries
  /0001 { /type "deny" /glob "*" }

  # open consoles
            # /0011 { /type "allow" /glob "* /admin/*" } # allow servlet engine
admin
            # /0012 { /type "allow" /glob "* /crx/*" } # allow content repository
            # /0013 { /type "allow" /glob "* /system/*" } # allow OSGi console
            # allow non-public content directories
            # /0021 { /type "allow" /glob "* /apps/*" } # allow apps access
            # /0022 { /type "allow" /glob "* /bin/*" }
            /0023 { /type "allow" /glob "* /content*" } # disable this rule to
allow mapped content only
            # /0024 { /type "allow" /glob "* /libs/*" }
            # /0025 { /type "allow" /glob "* /home/*" }
            # /0026 { /type "allow" /glob "* /tmp/*" }
            # /0027 { /type "allow" /glob "* /var/*" }
            # enable specific mime types in non-public content directories
            /0041 { /type "allow" /glob "* *.css *" } # enable css
            /0042 { /type "allow" /glob "* *.gif *" } # enable gifs
            /0043 { /type "allow" /glob "* *.ico *" } # enable icos
            /0044 { /type "allow" /glob "* *.js *" } # enable javascript
            /0045 { /type "allow" /glob "* *.png *" } # enable png
            /0046 { /type "allow" /glob "* *.swf *" } # enable flash
            # enable features
            /0061 { /type "allow" /glob "POST /content/[.]*.form.html"
} # allow POSTs to form selectors under content
            /0062 { /type "allow" /glob "* /libs/cq/personalization/*"
} # enable personalization
            # deny content grabbing
            /0081 { /type "deny" /glob "GET *.infinity.json*" }
            /0082 { /type "deny" /glob "GET *.-1.json*" }
            /0083 { /type "deny" /glob "GET *.tidy.json*" }
            /0084 { /type "deny" /glob "GET *.sysview.xml*" }
            /0085 { /type "deny" /glob "GET *.docview.json*" }
            /0086 { /type "deny" /glob "GET *.docview.xml*" }
}
```

Additional Information on configuring the Dispatcher

For additional information, see:

http://dev.day.com/docs/en/cq/current/deploying/dispatcher.html

Caching - Getting Better Performance

Cache Expiration - Subtrees that Expire Separately

Content subtrees expire separately when the cache is set to expire at any level below ***/content,*** for example you have configured /***statfileslevel*** = 2. This example will expire the cache at the language level. The question now becomes how to handle links between the subtrees - in this case, language switching?

Given **statsfileslevel** = 2 and a structure where the following subtrees exist:

/content/geometrixx/en
/content/geometrixx/de

An activated page in the "en" subtree would expire the "en" tree, but not the "de" tree.

If you have a language switch, e.g. a link on the English page that refers to theGerman translation directly, you may encounter the following problems:

1. A new "en" page is published. It gets requested and lies in the dispatcher cache afterwards. There is no link to the German translation page, because that page does not yet exist

2. The German page is published. It links to the English page correctly (because when it gets rendered for the first time the English page already exist on the publish instance). However, the English page is not invalidated in the dispatcher cache, hence the link to the German page never appears on the English page.

In addition, should the English page get unpublished, the link on the German page to the English page is not removed, as the German page will not be invalidated.

There are 2 solutions:

- use server-side includes on the apache web server that renders the language-switch links.
    - Drawback: this solution produces more load on web server
- Do not render the language on the server in the page directly, but let the client load them via ajax.
    - Drawback: more load on the client and the links are not indexed by google.

This above is a common example of the more generic problem of links between subtrees that expire separately.

Caching Queries

As mentioned above, requests with query parameters not cached. We recommend that you use selectors for passing query parameters and still use caching. For example:

/content/mypage.param1.param2.html

This method of using selectors also applies to group-wise personalization.

Caching Fragments

Consider an HTML fragment that appears on each page but is expensive to compute. You may improve performance by caching the HTML fragment and integrating it into the HTML page via SSI or client-side includes.

Caching and Periodic Importers

You have two choices:

- import on author and activate content
- import on author and publish, do not activate

NOTE: Make sure to not expire the cache too often. Depending on requirements you can:
• access content uncached in dispatcher
• activate only when content has actually changed

Advanced Dispatcher

Permission sensitive caching: uses HEAD request to check ACLs. This feature requires an additional package, that will be deployed inside AEM (you need to code a servlet that respond to the url and return the correct HTTP code). After deployment, check whether a user is allowed to access some page by requesting:

/bin/permissioncheck.html?uri=<handle>.

See:

http://dev.day.com/content/kb/home/cq5/CQ5SystemAdministration/PSCachingDelivery.html

for more information about this feature of the Dispatcher and how to configure the dispatcher.any file with the /authchecker configuration.
Sticky sessions are possible, but try to stay stateless on the publish instance.

Additional Dispatcher Performance Tips

In general, only 10% of total document request time is spent on server, the rest is transfer and client-side time. To improve performance:

- reduce number of total requests
- enable ***gzipping*** on web server

- use the HTML Library Manager to zip, concatenate and minify JS and css
    - we will cover that topic a bit later

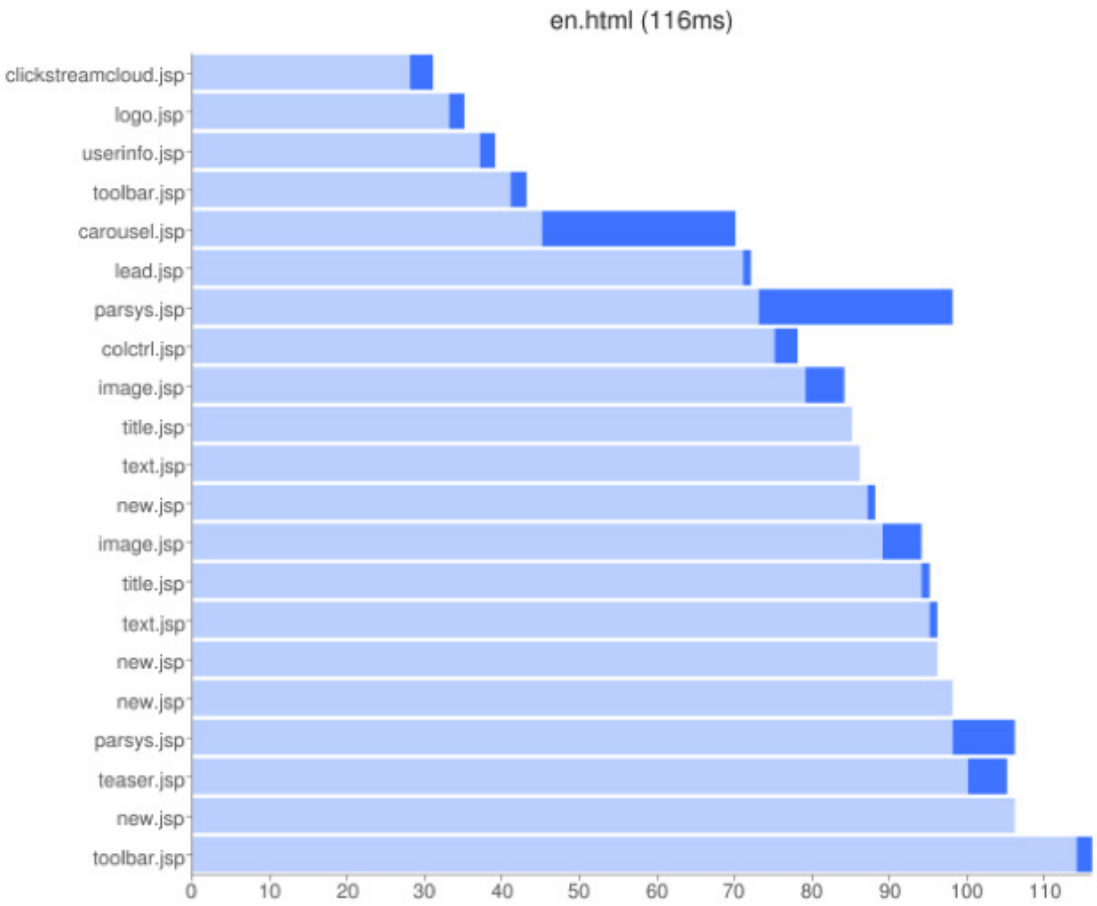
Finding Server-side Execution Problems

```
<!--
More detailed timing info is available by uncommenting some code in the timing.jsp component
Timing chart URL:
http://chart.apis.google.com/chart?chtt=en.html+%28116ms%29&cht=bhs&chxt=x&chco=c6d9fd,4d89f9&chbh=a&ch
-->
```

In order to find server-side execution bottlenecks in components, remember the Foundation timing component from the AEM Basic Developer training. The `<cq:include>` of the timing component:

```
<cq:include path="timing" resourceType="foundation/components/timing"/>
```

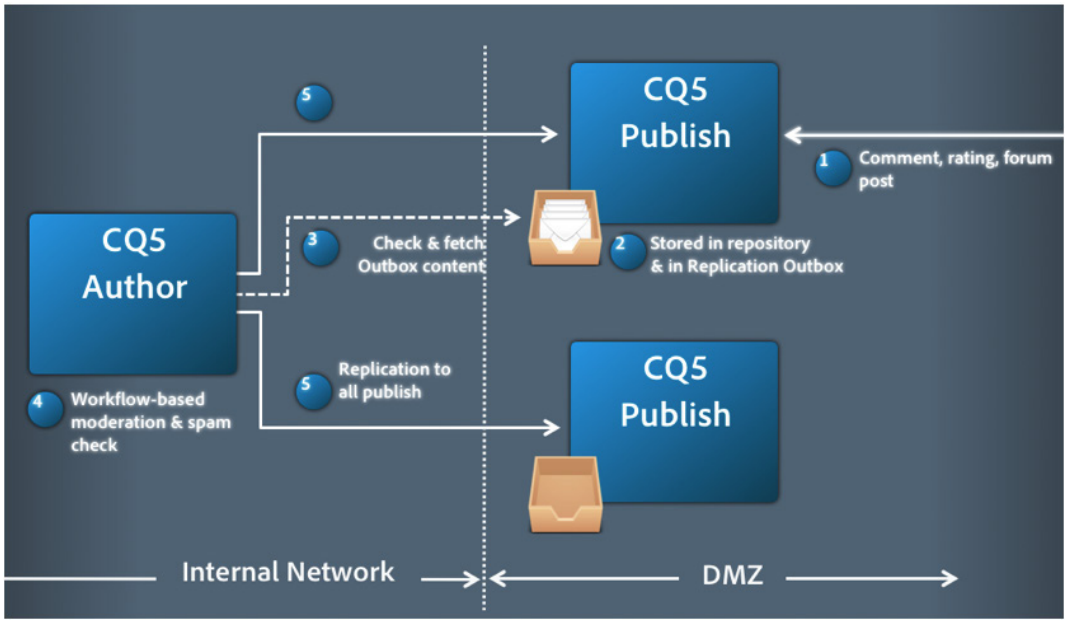
produces a comment in the html source that can be fed to Google Charts to produce output like the following:



Reverse Replication

Features such as comments and forms, allow users to enter information on a publish instance. For this a type of replication is needed to return this information to the author environment, from where it is redistributed to other publish environments. However, due to security considerations, any traffic from the publish to the author environment must be strictly controlled.

This is known as reverse replication and functions using an agent in the publish environment which references the author environment. This agent places the input into an outbox. This outbox is matched with replication listeners in the author environment. The listeners poll the outboxes to collect any input made and then distribute it as necessary. This ensures that the author environment controls all traffic.



The Basics

Reverse replication uses a polling approach, i.e. the author instance periodically asks the publisher instances for new content. Since the author instance is in control, the solution is firewall/DMZ-friendly.

The default mechanisms only cover transfer from 1 publisher instance back to the author instance. If you have multiple publish instances, then you must activate the moderated content to all publish instances, either through an explicit activation or through a workflow.

For automatic activation set up a workflow launcher or Sling Event Handler to activate when specified content is created/modified on the author instance.

Reverse replication and clustering

Your reverse replication need should be taken into consideration while selecting the appropriate clustering mechanism:

- If you don't have user generated content, use TarMK clustering.
- If you have user generated content and you DON'T need immediate content consistency, use TarMK clustering.
- If you have user generated content and you need immediate content consistency, use MongoMK cluster.

Programmatically Triggering Reverse Replication

Content created on publish is not automatically reverse-replicated. Out of the box, reverse replication only works for AEM's Blogs, Comments, Forums. Your application must write in one transaction (i.e. in one "save"):

- `cq:distribute`
- `cq:lastModified`
- `cq:lastModifiedBy`

Additional Information

For more information on reverse replication, see:

http://dev.day.com/docs/en/cq/current/deploying/configuring_cq/replication.html

13

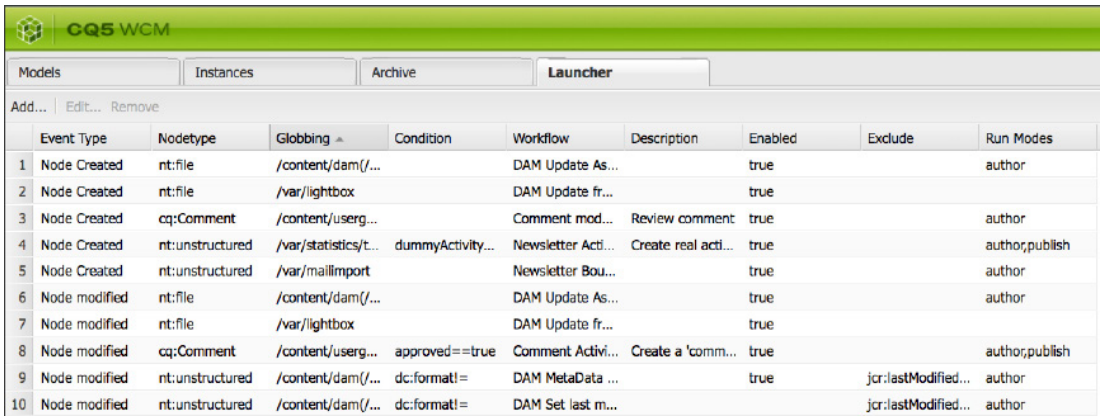
Content Automation, Periodic Importers

Event-based Workflow Launchers, what you learned already:

In the AEM Developer training course, you learned how to create a custom workflow step by implementing: `com.day.cq.workflow.exec.WorkflowProcess` and implementing the method:

`execute(WorkItem item, WorkflowSession session, MetaDataMap args)` to carry out the process step.

You can also make use of the Workflow Launcher to automate the launching of a workflow in response to any action which affects the repository. The Workflow Launcher configuration can be set to start a workflow when a specified node selection or nodes of a specified type are created, modified or deleted.



	Event Type	Nodetype	Globbing	Condition	Workflow	Description	Enabled	Exclude	Run Modes
1	Node Created	nt:file	/content/dam(/...		DAM Update As...		true		author
2	Node Created	nt:file	/var/lightbox		DAM Update fr...		true		
3	Node Created	cq:Comment	/content/userg...		Comment mod...	Review comment	true		author
4	Node Created	nt:unstructured	/var/statistics/t...	dummyActivity...	Newsletter Acti...	Create real acti...	true		author,publish
5	Node Created	nt:unstructured	/var/mailimport		Newsletter Bou...		true		author
6	Node modified	nt:file	/content/dam(/...		DAM Update As...		true		author
7	Node modified	nt:file	/var/lightbox		DAM Update fr...		true		
8	Node modified	cq:Comment	/content/userg...	approved==true	Comment Activi...	Create a 'comm...	true		author,publish
9	Node modified	nt:unstructured	/content/dam(/...	dc:format!=	DAM MetaData ...		true	jcr:lastModified...	author
10	Node modified	nt:unstructured	/content/dam(/...	dc:format!=	DAM Set last m...			jcr:lastModified...	author

So now you have several choices regarding the best way to automate processes in AEM. You can create a JCR Observation Listener, or use Sling Eventing, to listen for a suitable event such as NODE_ADDED, and trigger whatever action is required. Or you can use the Workflow Launcher.

Comparison:

Workflow Launcher	JCR Observation Listener and Sling Eventing
Has User Interface	No User Interface
Simple to start / stop	Requires use of Apache Felix console to stop listener component.
Workflow model can be changed on the fly	Requires programmatic or configuration changes.
More overhead – better where only moderate amounts of events are expected.	Less overhead, can handle more frequent events.
Cluster-aware (runs only on cluster master).	Not cluster aware. (Slingeventing)

Custom Periodic Importers

The feed importer is a framework to repeatedly import content from external sources into your repository. The idea of the feed importer is to poll a remote resource at a specified interval, to parse it, and to create nodes in the content repository that represent the content of the remote resource. In AEM Sites (out of the box), the feed importer is used for the following:

In the blog to support the auto-blogging feature, which automatically creates blog posts from an external RSS or Atom feed.

In the calendar for iCalendar subscriptions, which automatically creates calendar events from an external ICS file or subscribes to/from other calendars. The feed importer is found in the WCM Administrative interface under Tools >Importers > Feed Importer. Double-clicking on the Feed Importer node will open a page which allows configuration of standard feed importers for RSS, Atom, Calendar, IMAP, and POP3 but in this exercise we will create our own bespoke importer. The importer can be configured to poll at regular intervals specified in the configuration details. To implement your own feed importer, you use the Interface: com.day.cq.polling.importer.Importer

Example code

```
@Component
@Service
@Property(nameRef="Importer.SCHEME_PROPERTY"values.0="rss"
values.1="atom")
public class FeedImporter implements Importer {
        public void importData(String scheme, String dataSource,
Resource target) throws ImportException {
```

- `Importer.SCHEME_PROPERTY` defines the type of import (arbitrary, re-used in UI)
- `importData(String scheme, String dataSource, Resource target)` contains no inherent logic; it is up to the application developer.
- In order to add a new schema to the Tools UI, overlay `cq/ui/widgets/source/widgets/wcm/FeedImporter.js`
- If that is not needed you can just add the config as a node to the project.

Alternative Approaches

Alternative ways of achieving the desired results could include:

- A Web-DAV enabled "dropbox" and a workflow action (requires file-based interaction).
- Sling Cron Jobs (but a disadvantage is that there is no UI available).

EXERCISE 13.1 - AEM Importer And Custom Workflow Step

Goal

The aim of this exercise is to create a polling importer to import configurable stock data, e.g. for Adobe, available from:

http://finance.yahoo.com/d/quotes.csv?s=ADBE&f=snd1l1yr, then save the latest stock data in the repository. The repository modification will trigger a workflow, and this will check for and report if the stock level reaches a certain value.

Overview

- Create a custom polling importer to import stock prices.
- Create a custom workflow step and include it in a workflow.
- Trigger the workflow after the importer has run with a workflow launcher configuration.
- When the imported stock data exceeds a certain threshold, a log entry shall be written.
- Add the importer setting and workflow model and launcher to your config package.

Steps

1. The pom.xml files will require the group Id `com.day.cq` and the artifactId, `com.day.cq.cq-polling-importer`, however for simplicity this dependency has already been added to the files for you.

2. Create the StockDataImporter class (skip typing the import statements - they can be generated by eclipse as or after you type the body code):

```
package com.adobe.training.core;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Arrays;
import java.util.Calendar;
import java.util.regex.Pattern;

import javax.jcr.Node;
import javax.jcr.RepositoryException;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Property;
import org.apache.felix.scr.annotations.Service;
import org.apache.sling.api.resource.Resource;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.day.cq.commons.jcr.JcrUtil;
import com.day.cq.polling.importer.ImportException;
import com.day.cq.polling.importer.Importer;

@Service(value = Importer.class)
@Component
@Property(name = Importer.SCHEME_PROPERTY, value = "stock",
propertyPrivate = true)
public class StockDataImporter implements Importer {
```

The importer.scheme property is used to identify different kinds of importers (RSS, etc).

3. Implement the download logic

The Polling Importer will periodically call the method importData(). You need to override it.

```
private final String SOURCE_URL =
        "http://finance.yahoo.com/d/quotes.csv?f=snd1l1yr&s=";

private final Logger LOGGER = LoggerFactory.getLogger
        (StockDataImporter.class);

@Override
public void importData(final String scheme, final String
        dataSource, final Resource resource) throws
ImportException {
        try {
        // dataSource will be interpreted as stock symbol
        URL sourceUrl = new URL(SOURCE_URL + dataSource);
        BufferedReader in = new BufferedReader
                (new InputStreamReader(sourceUrl.openStream()));
        String readLine = in.readLine();
        // expecting only one line
        String lastTrade = Arrays.asList(Pattern.compile
        (",").split(readLine)).get(3);
        LOGGER.info("Last trade for stock {} was {}",
                dataSource, lastTrade);
        in.close();

        //persist
        writeToRepository(dataSource, lastTrade, resource);
        }
        catch (MalformedURLException e) {
        LOGGER.error("MalformedURLException", e);
        }
        catch (IOException e) {
        LOGGER.error("IOException", e);
        }
        catch (RepositoryException e) {
        LOGGER.error("RepositoryException", e);
        }
}
```

4. The helper function to persist the last trade data:

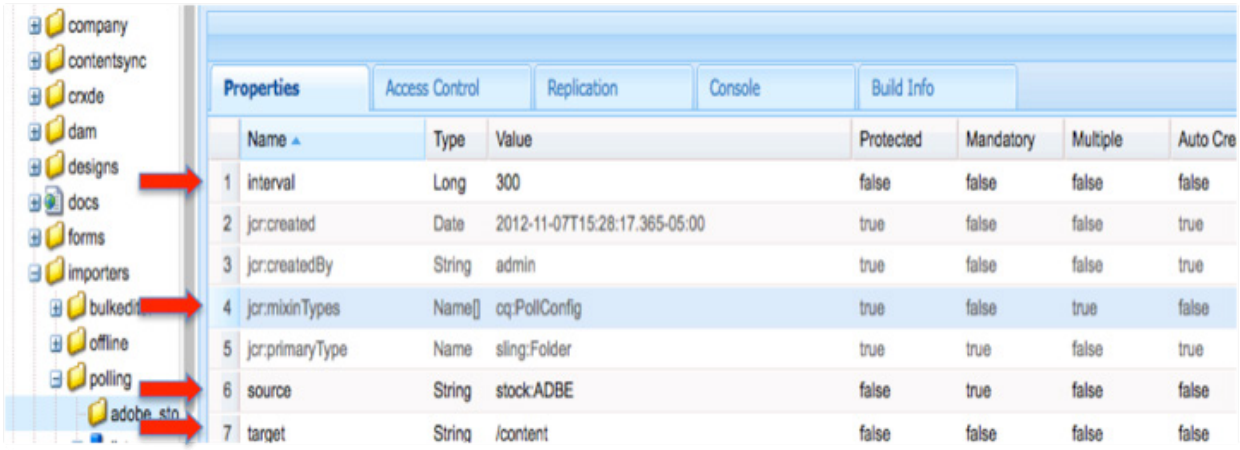
```
private void writeToRepository(final String stockSymbol,
            final String lastTrade, final Resource resource)
            throws RepositoryException {
            Node parent = resource.adaptTo(Node.class);
            Node stockPageNode = JcrUtil.createPath(parent.getPath()
            + "/" + stockSymbol, "cq:Page", parent.getSession());
            Node lastTradeNode = JcrUtil.createPath (stockPageNode.getPath()
            + "/lastTrade", "nt:unstructured", parent.getSession());
            lastTradeNode.setProperty("lastTrade", lastTrade);
            lastTradeNode.setProperty("lastUpdate", Calendar.getInstance());
            parent.getSession().save();
   }
}
```

NOTE: Because the created cq:Page node does not have a jcr:content child, you cannot open the page.

This will create a page node with the name of the stock symbol as node name, and store the last trade and last update time as properties below the page.

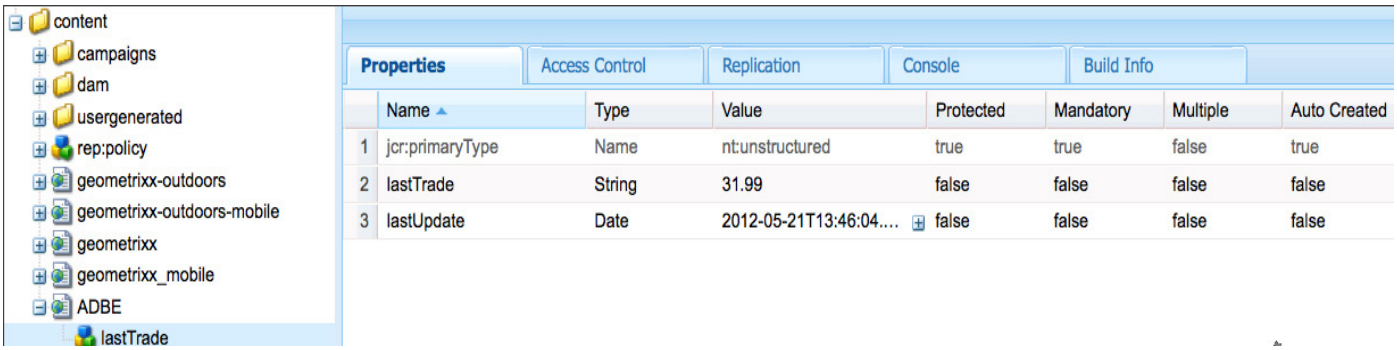
5. Deploy the core bundle and check that the component is installed in the Felix console.

6. Add an importer configuration in `/etc/importers/polling`. Create a new node of type `sling:Folder`,and name `adobe_stock`, and apply ***all*** the properties shown below (some are generated automatically: For example the jcr:mixin Type property is added when you add the cq:Poll Config mixin to the adobe.stock node:



7. Check the configuration in http://localhost:4502/etc/importers/polling.html (the page is linked in Tools > Importers > Polling).

8. After 5 minutes you should see a page created under the content node, and properties showing the latest trade (refresh the view to see this):



9. Next, create a workflow that alerts authors when a stock value exceeds a certain threshold. Again, note the required workflow artifacts which have already been added to the pom.xml and imported to eclipse:

```
<dependency>
    <groupId>com.day.cq.workflow</groupId>
    <artifactId>cq-workflow-api</artifactId>
    <version>5.5.0</version>
    <scope>provided</scope>
</dependency>
```

10. Create the automated workflow process, StockAlertProcess that checks a workflow item (assuming that is an imported stock data) against a list of alert thresholds and log if the threshold is exceeded:

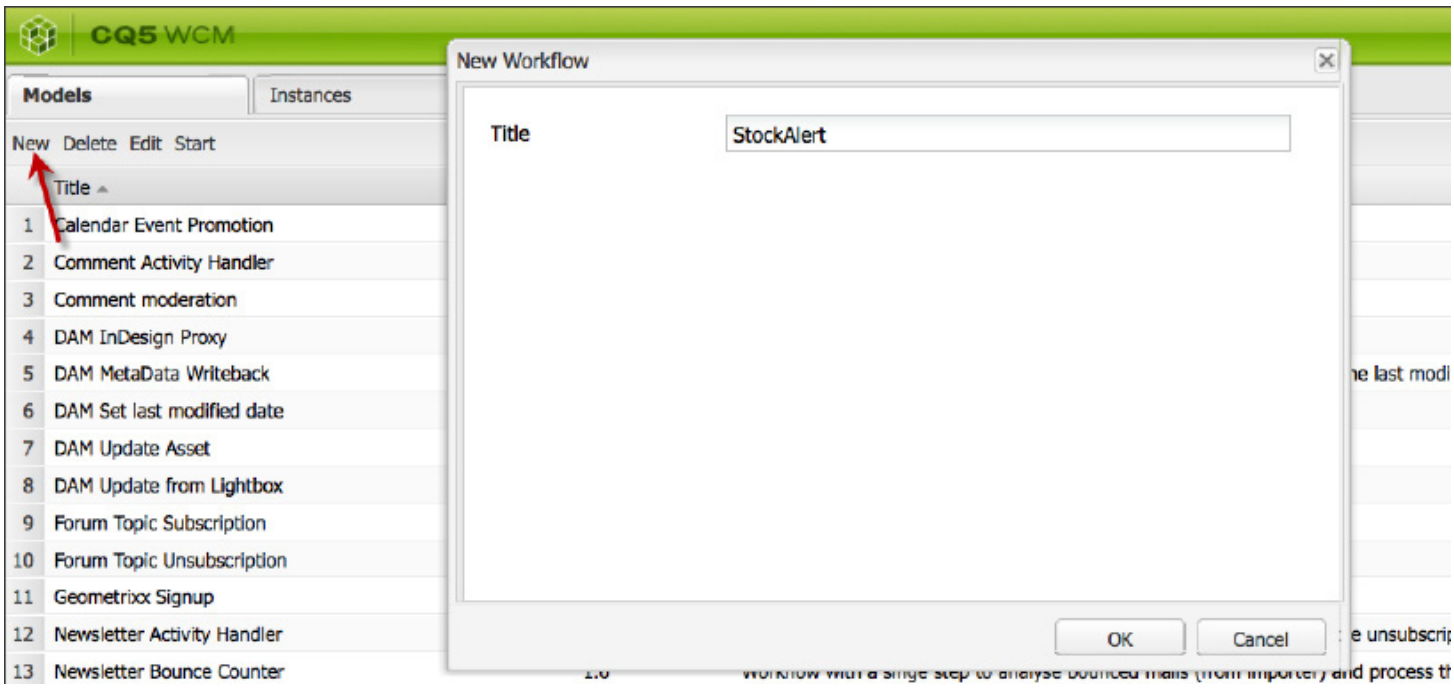
```
package com.adobe.training.core;

import java.util.Arrays;
import java.util.Iterator;
import java.util.List;
import java.util.regex.Pattern;

import javax.jcr.Node;
import javax.jcr.RepositoryException;
import javax.jcr.Session;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Property;
import org.apache.felix.scr.annotations.Service;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.day.cq.workflow.WorkflowException;
import com.day.cq.workflow.WorkflowSession;
import com.day.cq.workflow.exec.WorkItem;
import com.day.cq.workflow.exec.WorkflowData;
import com.day.cq.workflow.exec.WorkflowProcess;
```

```
import com.day.cq.workflow.metadata.MetaDataMap;

@Service
@Component(metatype = false)
@Property(name = "process.label", value = "Stock Threshold
Checker")
public class StockAlertProcess implements WorkflowProcess {

private static final String PROPERTY_LAST_TRADE =
"lastTrade";
private static final String TYPE_JCR_PATH = "JCR_PATH";
private static final String TYPE_JCR_UUID = "JCR_UUID";
private static final Logger LOGGER = LoggerFactory.getLogger
                (StockAlertProcess.class);

@Override
public void execute(WorkItem workItem, WorkflowSession
workflowSession,
        MetaDataMap args) throws WorkflowException {
        try {
        // get the node the workflow is acting on
        Session session = workflowSession.getSession();
        WorkflowData data = workItem.getWorkflowData();
        Node node = null;
        String type = data.getPayloadType();
        if(type.equals(TYPE_JCR_PATH) &&
                data.getPayload() != null) {
            String payloadData = (String) data.getPayload();
            if(session.itemExists(payloadData)) {
                node = session.getNode(payloadData);
            }
        }
        else if (data.getPayload() != null && type.equals
                (TYPE_JCR_UUID)) {
            node = session.getNodeByIdentifier((String)
                data.getPayload());
        }
        LOGGER.info("running with node {}", node.getPath());
        // parent's is expected to be stock symbol
        String symbol = node.getParent().getName();
        LOGGER.info("found symbol {}", symbol);
        if (node.hasProperty(PROPERTY_LAST_TRADE)) {
            Double lastTrade = node.getProperty
                (PROPERTY_LAST_TRADE).getDouble();
            LOGGER.info("last trade was {}", lastTrade);
            // reading the passed arguments
            Iterator<String> argumentsIterator =
            Arrays.asList(Pattern.compile("\n").split
            (args.get("PROCESS_ARGS", "").iterator();
            while (argumentsIterator.hasNext()) {
                List<String> currentArgumentLine =
                Arrays.asList(Pattern.compile("=")
```

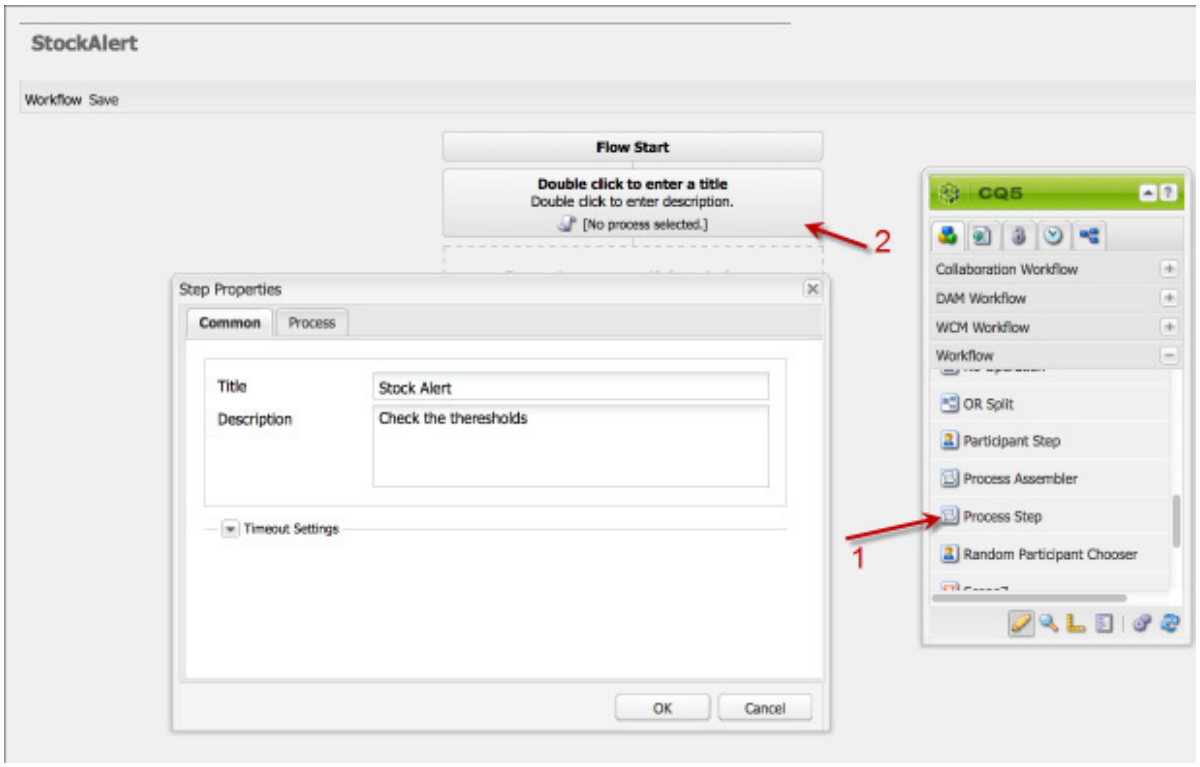
```
                    .split(argumentsIterator.next()));
                    String currentSymbol =
                            currentArgumentLine.get(0);
                    Double currentLimit = new Double
                            (currentArgumentLine.get(1));
                    if (currentSymbol.equalsIgnoreCase(symbol)
                                    && currentLimit < lastTrade) {
                LOGGER.warn("Stock Alert! {} isover {}", symbol, currentLimit);
                            }
                        }
                    }
            }
            catch (RepositoryException e) {
            LOGGER.error("RepositoryException", e);
            }
        }
}
```

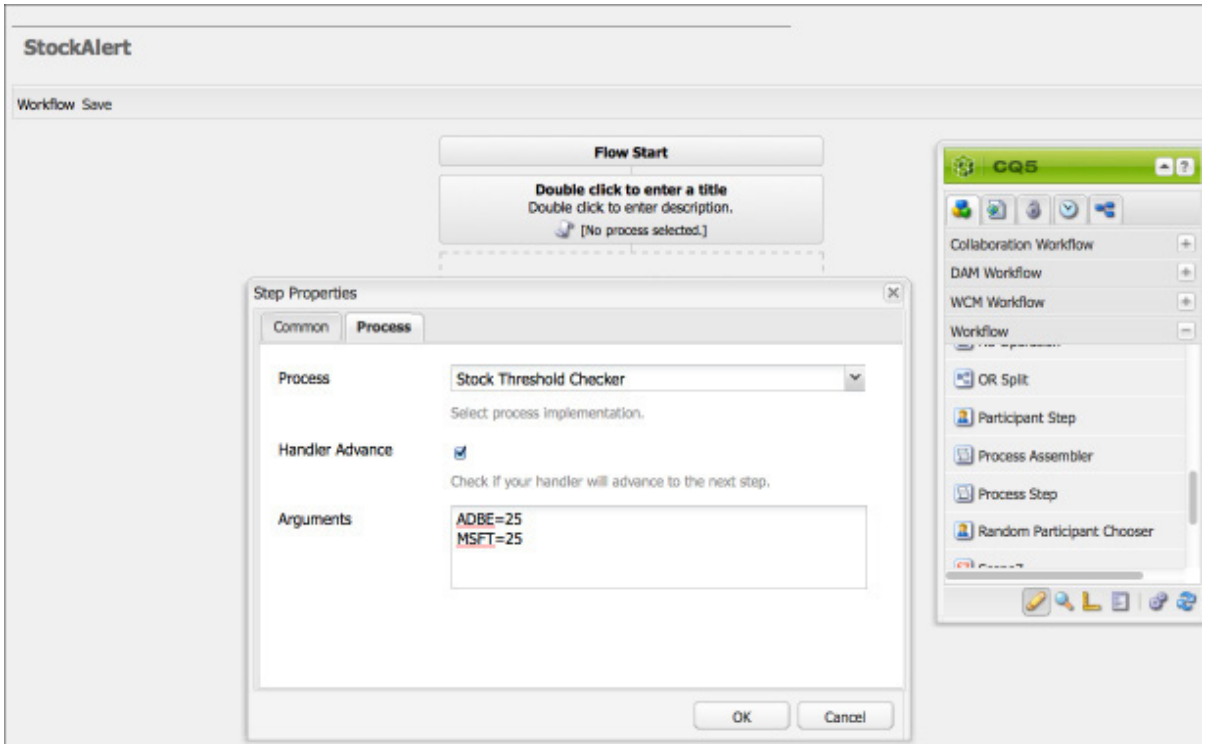
11. Deploy the workflow process via mvn.

12. In the workflow admin UI (linked from AEM's homepage) create a new workflow model that contains this step (NB: unless you first disable the TitlePropertyListener it will add a "!" to the title):



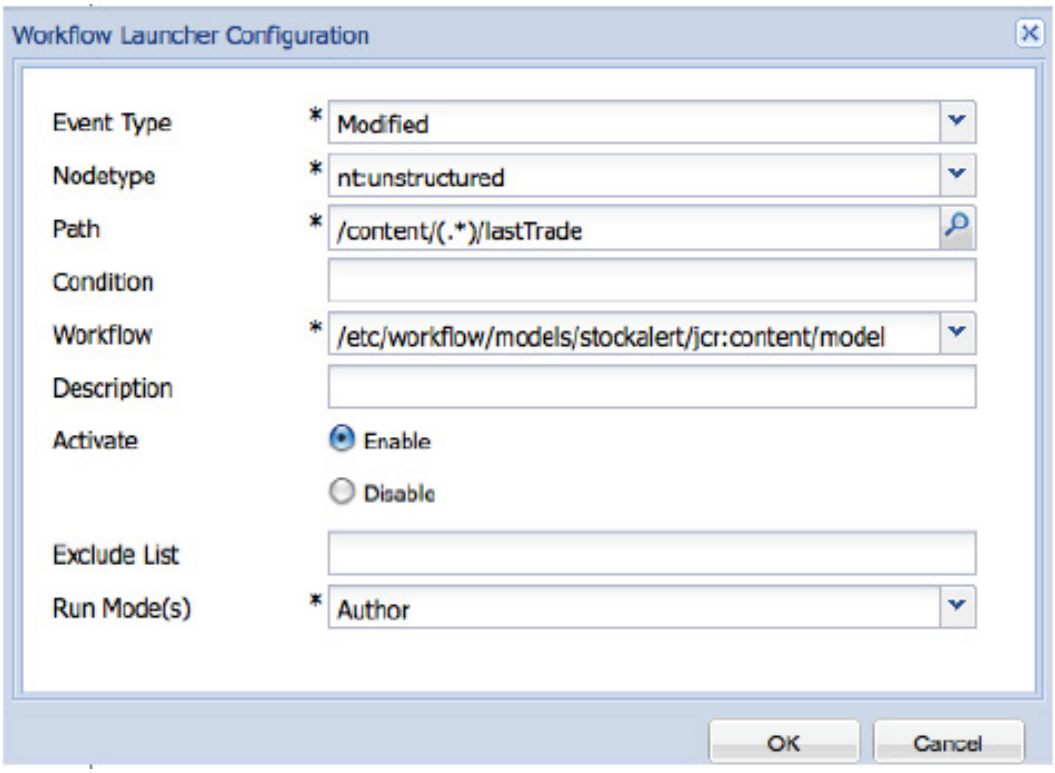
13. Open the new workflow for editing, and add a process step as shown below:





14. Do not forget to save the model!

15. Create a workflow launcher to start the workflow whenever a "lastTrade" node is modified:



16. Export the workflow model and launcher configuration, and the importerconfiguration via vlt to your file system. You will have to modify META-INF/vault/filter.xml in your company-ui project to add something like this:

```
<filter root="/etc/workflow/models/stockalert"/>
<filter root="/etc/workflow/launcher/config"/>
<filter root="/etc/importers/polling/adobe_stock"/>
```

Then use the vlt command:

```
vlt up
```

To update the local file contents with changes in the repository.

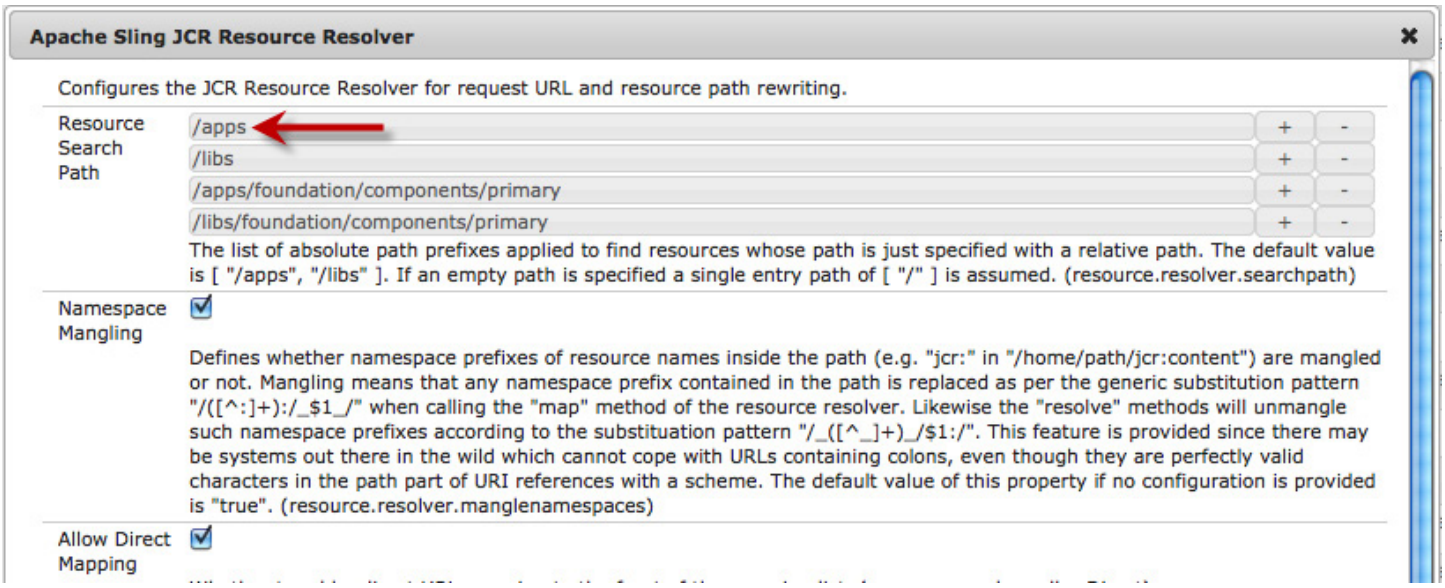
Congratulations! You have now created a custom AEM importer, and a custom workflow step as part of a workflow triggered by a launcher configuration.

14

Overlays, Extending Foundation Components, Content Reuse

Reuse: Overlays, Extending the Foundation Components

This is actually a Sling topic, but most useful in the context of an application like CQ5. Sling's JCR Resource Resolver will search for resources in a list of specific paths, as defined by the Apache Sling JCR Resource Resolver. The default search path is: first */apps*, then */libs*



As a result, you can change the out of the box functionality, as provided in */libs* by adding a resource/ file at the same path in */apps*. This ability to override default functionality is called an "**overlay**".

Use Cases

You can overlay single files in an out of the box component in /libs to change their behavior. You can also customize functionality, for example the 404 error handler located in */libs/sling/servlet/ errorhandler.*

Beware

Once you copy a system file from /libs to /apps to overlay it with custom code, your custom code will not pick up any modifications, to the system component/file/ script/dialog box, that result from the application of a hotfix, featurepack, or upgrade.

A careful examination of the release notes of any upgrade, featurepack, or hotfix should be a step in your upgrade plan. This way you will be able to evaluate any changes and make a plan for incorporating them into your application.

Extending the Foundation Components

In addition to the "overlay" capability described above you can also extend the Foundation Components through the use the of Resource Hierarchy and super types. Using the super type allows you to inherit some things, for example thumbnails and dialog boxes, from the Foundation Component, while overriding other things, for example rendering scripts, through the use of local copies.

In addition to the resource types (primarily defined by the ***sling:resourceType*** property) there is also the resource super type. This is generally indicated by the ***sling:resourceSuperType*** property. These super types are also considered when trying to find a script. The advantage of resource super types is that they may form a hierarchy of resources where the default resource type **sling/servlet/default** (used by the default servlets) is effectively the root.

The resource super type of a resource may be defined in two ways:

- **sling:resourceSuperType** property of the resource
- ***sling:resourceSuperype*** property of the node to which the ***sling:resourceType*** points

15

Content Migration/Import

It is a common requirement in many AEM projects to migrate content from an existing CMS into AEM. There are three basic possibilities to import content into AEM:

- vlt
- Sling POST servlet
- JCR API

Vlt-based migration

1. Export the content into a vlt serialization. This will create .content.xml files containing an xml representation of the content at each node.
2. Use vlt to import the content into AEM.
    - A variation on this method involves creating a content package (i.e. zip the exported content) which can then be handled by the AEM Package Manager.
    - Vlt-based migration is efficient for large quantities of content, but it may not be easy to debug if you get the format wrong.

Sling POST Servlet

1. Export the content into a file system.

2. Use a shell script to transform the content into curl commands to the Sling POST Servlet.

   - This is good for ad hoc testing, but inefficient for large quantities of content.
   - The Sling POST Servlet is not useful for creation of AEM:Page nodes.

JCR API

1. Export the content into a file system in an arbitrary structure.
2. Run a (JSP) script that accesses the file system and creates corresponding content in the repository.

   - This method is efficient, and allows the use of AEM/jcr methods.
   - It is also useful for scripted fixes to the content structure.

EXERCISE 15.1 - Explore Approaches To Import Content From Legacy CMSs

Goal

The aim of this exercise is to use vlt, the Sling POST servlet and a JSP script to import content which might have come from a legacy system.

Using vlt

First, we will look at the use of vlt to migrate content. We will simulate the import of new content by adding a new campaign using vlt.

All existing campaigns are firstly exported into the file system, so that we have the required structure of nodes and properties represented in the file system, and can easily replicate this. Then by copying an existing campaign structure and modifying the copy, we simulate the import of new content in the required structure. Finally we use vlt to update the repository with the new content.

1. In `company-ui/src/main/content/META-INF/vault/filter.xml` add a filter for `/content/campaigns` (so that vlt also serializes these nodes to the file system):

```
<?xml version="1.0" encoding="UTF-8"?>
<workspaceFilter version="1.0">

    <filter root="/apps/company">
        <exclude pattern="/apps/apps/config.*"/>
    </filter>
    <filter root="/etc/designs/company"/>

    <filter root="/etc/workflow/models/stockalert"/>
    <filter root="/etc/workflow/launcher/config"/>
    <filter root="/etc/importers/polling/adobe_stock"/>

    <filter root="/content/campaigns"></filter>
</workspaceFilter>
```

2. On the command line change to directory `company-ui\src\main\content\jcr _ root` and execute vlt up. You should see the `campaigns` nodes and `.content.xml` files being copied into the file system:

```
Connecting via JCR remoting to http://localhost:4502/crx/server
U .content.xml
U etc/.content.xml
U etc/designs/.content.xml
A etc/importers
A etc/importers/.content.xml (text/xml)
A etc/importers/polling
A etc/importers/polling/.content.xml (text/xml)
A etc/workflow
A etc/workflow/.content.xml (text/xml)
A etc/workflow/launcher
A etc/workflow/launcher/.content.xml (text/xml)
A etc/workflow/models
A etc/workflow/models/.content.xml (text/xml)
U apps/.content.xml
U apps/company/install/company-core-0.0.1-SNAPSHOT.jar
U apps/company/components/.content.xml
U apps/company/components/my_node/.content.xml
apps/company/components/my_file.jsp Remote Binary type differs from actual data. Content Type: appli
apps/company/components/my_file.jsp can't merge. binary content
C apps/company/components/my_file.jsp
D apps/company/components/my_file.jsp.dir/.content.xml
D apps/company/components/my_file.jsp.dir
A content
A content/.content.xml (text/xml)
A content/campaigns
A content/campaigns/.content.xml (text/xml)
A content/campaigns/geometrixx-outdoors
A content/campaigns/geometrixx-outdoors/.content.xml (text/xml)
A content/campaigns/geometrixx-outdoors/banner
A content/campaigns/geometrixx-outdoors/banner/.content.xml (text/xml)
A content/campaigns/geometrixx-outdoors/banner/summer-female-under30
A content/campaigns/geometrixx-outdoors/banner/summer-female-under30/.content.xml (text/xml)
A content/campaigns/geometrixx-outdoors/banner/summer-female-over30
A content/campaigns/geometrixx-outdoors/banner/summer-female-over30/.content.xml (text/xml)
A content/campaigns/geometrixx-outdoors/banner/winter-female-under30
A content/campaigns/geometrixx-outdoors/banner/winter-female-under30/.content.xml (text/xml)
A content/campaigns/geometrixx-outdoors/banner/winter-female-over30
A content/campaigns/geometrixx-outdoors/banner/winter-female-over30/.content.xml (text/xml)
A content/campaigns/geometrixx-outdoors/banner/summer-male-under30
A content/campaigns/geometrixx-outdoors/banner/summer-male-under30/.content.xml (text/xml)
A content/campaigns/geometrixx-outdoors/banner/summer-male-over30
A content/campaigns/geometrixx-outdoors/banner/summer-male-over30/.content.xml (text/xml)
A content/campaigns/geometrixx-outdoors/banner/winter-male-under30
A content/campaigns/geometrixx-outdoors/banner/winter-male-under30/.content.xml (text/xml)
A content/campaigns/geometrixx-outdoors/banner/winter-male-over30
A content/campaigns/geometrixx-outdoors/banner/winter-male-over30/.content.xml (text/xml)
A content/campaigns/geometrixx-outdoors/banner/summer-female
A content/campaigns/geometrixx-outdoors/banner/summer-female/.content.xml (text/xml)
A content/campaigns/geometrixx-outdoors/banner/winter-female
A content/campaigns/geometrixx-outdoors/banner/winter-female/.content.xml (text/xml)
A content/campaigns/geometrixx-outdoors/banner/summer-male
A content/campaigns/geometrixx-outdoors/banner/summer-male/.content.xml (text/xml)
```

3. Copy the `scott-recommends` campaign to a new campaign called `shantanu-recommends`, e.g. by executing

   On Mac:

   ```
   cp -R content/campaigns/geometrixx/scott-recommends
   content/campaigns/geometrixx/shantanu-recommends
   ```

   On Windows:

   ```
   xcopy content\campaigns\geometrixx\scott-recommends content\
   campaigns\geometrixx\shantanu-recommends /S /E /I /H
   ```

4. Make sure you remove the `.vlt` file below `shantanu-recommends`:

   Mac:

   ```
   rm content/campaigns/geometrixx/shantanu-recommends/.vlt
   ```

   Windows:

   ```
   del content\campaigns\geometrixx\shantanu-recommends\.
   vlt
   ```

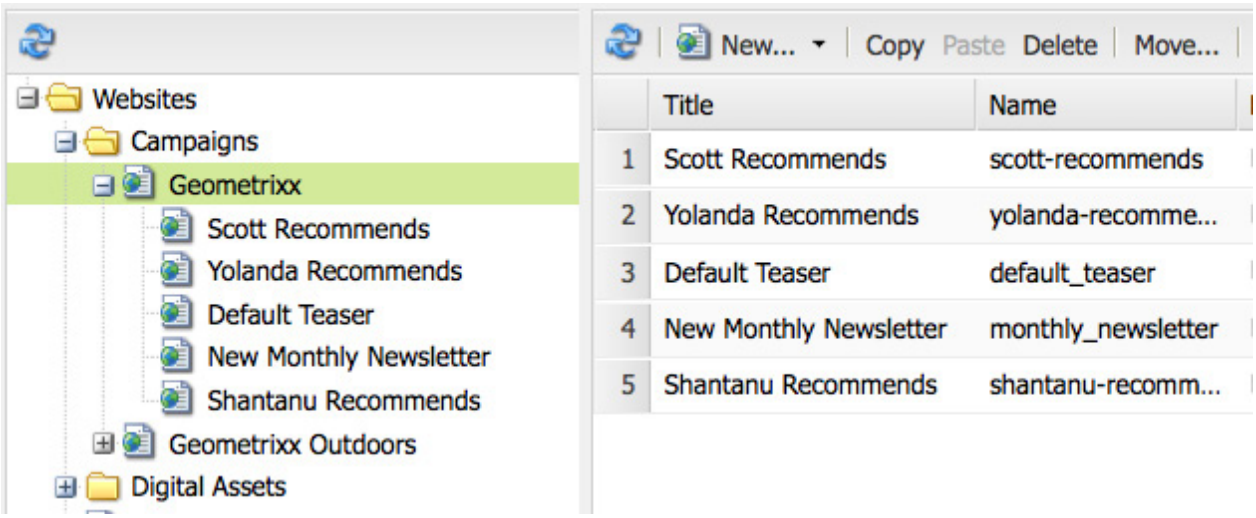
5. Edit `shantanu-recommends/.content.xml` and replace "`Scott`" with "`Shantanu`", e.g.:

```
<jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0" xmlns:cq="
    jcr:primaryType="cq:Page">
    <jcr:content
        cq:lastModified="{Date}2011-02-02T16:46:52.932+01:00"
        cq:lastModifiedBy="admin"
        cq:segments="/etc/segmentation/geometrixx/male"
        cq:template="/libs/cq/personalization/templates/teaser"
        jcr:primaryType="cq:PageContent"
        jcr:title="Shantanu Recommends"
```

6. Put (add) the new content under version control and check-in (ci) the new content into the repository with vlt:

   ```
   vlt add content/campaigns/geometrixx/shantanu-recommends
   vlt ci content/campaigns/geometrixx/shantanu-recommends
   ```

   Inspect the new campaign in the site admin:



Using The Sling POST servlet

Next, we will use the Sling POST servlet, and curl, to add some new content. Curl is used to issue an http request, and by using a POST request we can make use of curl to call the Sling POST servlet and post new content into the repository.

If you have not previously installed curl then you will need to do this now.

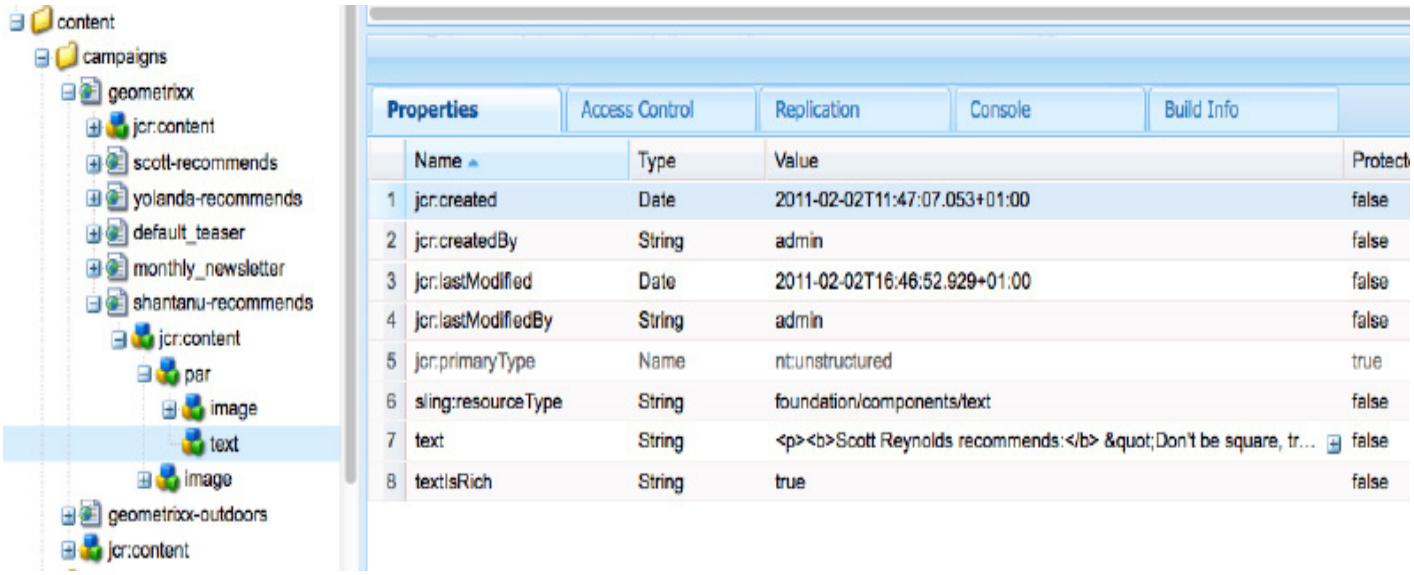
Installing Curl

Curl is provided on the USB memory stick. You can also download curl from http://curl.haxx.se/download.html. Please ensure that you choose the correct version for your operating system.

To use curl, simply extract the file to a suitable location (e.g. `/usr/bin` on the Mac, or `C:\curl` in Windows) then ensure that the PATH environment variable is configured to include the path to curl. (On windows, type `SET PATH=%PATH%;C:\curl`. The Mac PATH should already include `/usr/bin`). Type `curl --version` to test that the installation is working.

Using Curl

1. Inspect the campaigns in CRXDE Lite. We would like to add a text node to Shantanu's campaign.

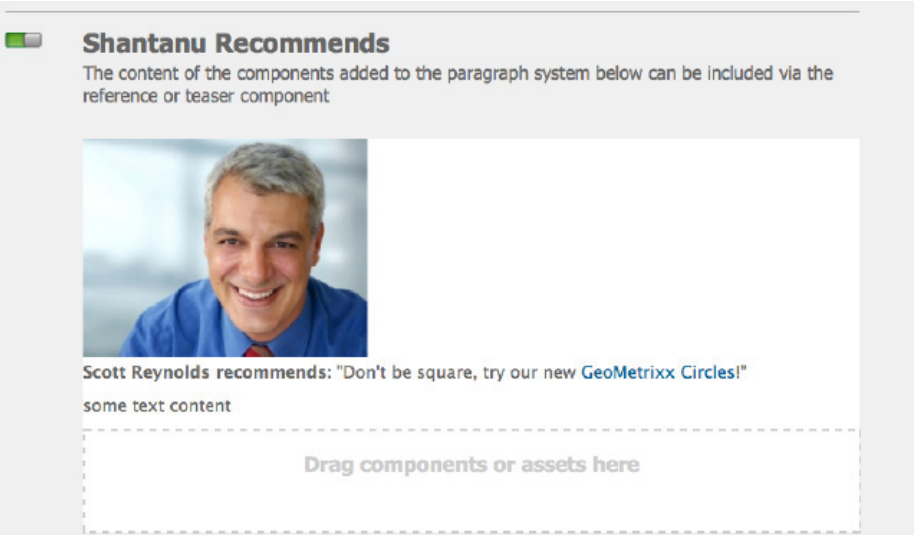


2. On the command line execute (in one line, below is split for better readability, and with no spaces in the URI):

```
curl -u admin:admin -X POST
-d "sling:resourceType=foundation/components/
text&text=<p>some
text content</p>&textIsRich=true"
http://localhost:4502/content/campaigns/geometrixx/shantanu-
recommends/jcr:content/par/*
```

This is POSTing a rich text string to the par node under the shantanu-recommends campaign. Have a look at http://curl.haxx.se/docs/ for full details on all the curl options and settings.

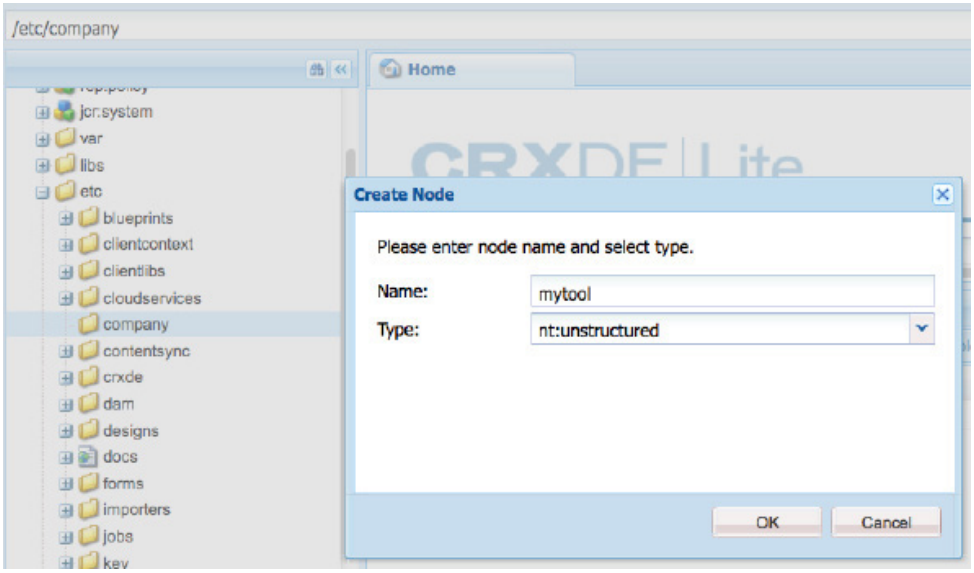
3. You should see when you open the campaign that there is a new text component:

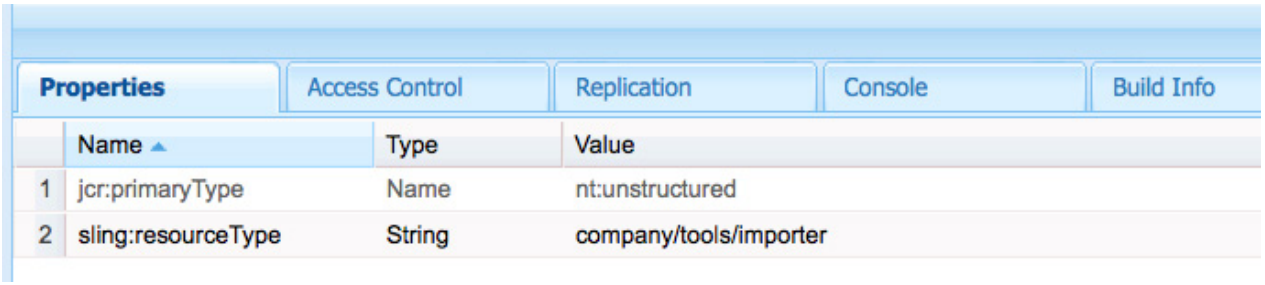


Using The JCR API

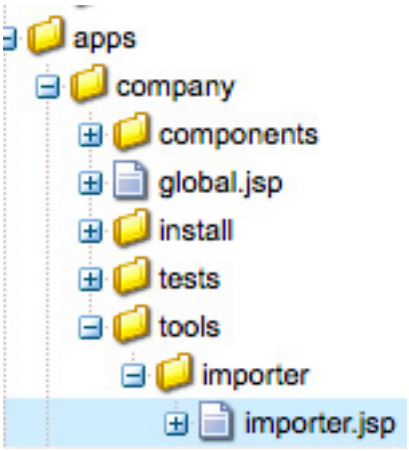
Finally, we will use a JSP script which makes use of the JCR API to search for specific content and add new content wherever it is found. This is a simple example, but the technique could be used to update content from a legacy system and add required nodes or properties to make the content useable in AEM.

1. Using CRXDE Lite, in the `/etc` node, add a node called "`company`" of type "`sling:Folder`", and inside this node an `nt:unstructured` node called "`mytool`". Add a `sling:resourceType` property to "mytool" with type `String` and value "`company/tools/importer`". This will be the resource that we address with the browser, in order to execute the JSP script. Save the changes.





2. In /apps/company add the folders tools/importer and a jsp "importer.jsp" Save the changes.

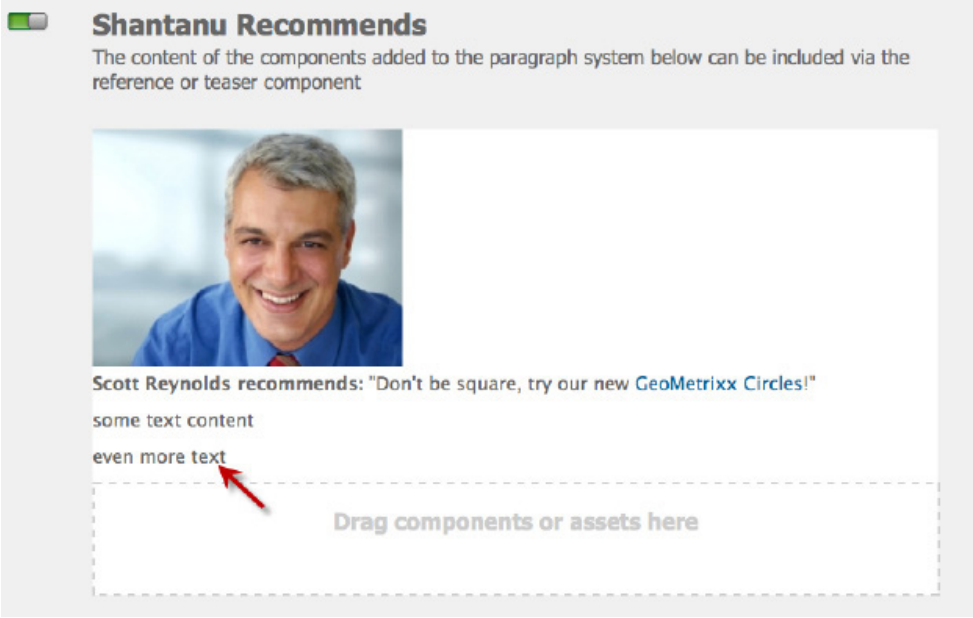


3. Now write the following jsp script, which will search for paragraph nodes in the Geometrixx campaigns, (by checking for sling:resourceType='foundation/components/parsys') and add child nodes with text to all the nodes found:

```
<%@page import="javax.jcr.*,javax.jcr.query.*,java.util.*,
        com.day.AEM.commons.jcr.JcrUtil"%>
<%@page contentType="text/html; charset=utf-8"%>
<%@include file="/libs/foundation/global.jsp"%>
<html>
<head>
<title>Campaigns Update</title>
</head>
<body>
<%
        String q = "/jcr:root/content/campaigns/geometrixx//*" +
          "[@sling:resourceTy pe='foundation/components/parsys']";
           Query query = currentNode.getSession().getWorkspace().
getQueryManager()
            .createQuery(q, "xpath");
          NodeIterator result = query.execute().getNodes();
          int counter = 0;
          while (result.hasNext()) {
          Node n = result.nextNode();
                Node newTextNode = JcrUtil.createUniqueNode(n,"newtext",
```

```
            "nt:unstructured", currentNode.getSession());
            newTextNode.setProperty("sling:resourceType",
            "foundation/components/text");
            newTextNode.setProperty("text", "<p>even more text</p>");
            newTextNode.setProperty("textIsRich", "true");
            counter++;
        }
        currentNode.getSession().save();
        out.println("Added nodes: " + counter);
%>
</body>
</html>
```

4. Point your browser to http://localhost:4502/etc/company/mytool.html to execute the code. Afterwards the campaigns should have an additional text node:



Congratulations! You have used vlt, the Sling POST servlet, and the JCR API to import and modify content in the AEM repository. These techniques can be adapted to solve many legacy migration issues that you may face.

Upgrading to AEM 6.0

Upgrading the existing version to AEM 6.0 doesn't require any additional effort.
Follow these procedure:

1. Stop the AEM instance.
2. Replace the Quickstart jar with that of AEM 6.0.
3. Restart the new AEM instance.
4. Wait for AEM Admin console to appear.

When you upgrade to AEM 6.0 as explained above, the following activities happen in the background:

- The Sling Launchpad code detects that it is using the state of an older instance, and activates the UPDATE startup mode.
- Some content packages, when installed, back up content under `/var/upgrade`, so that upgrade services can pick it up later for merging with new content.
- The `com.day.cq.cq-compat-codeupgrade` bundle registers a number of services that implement the `CodeUpgradeTask` interface.
- Later, at a higher OSGi start level, the *`com.day.cq.cq-upgrades-executor`* bundle runs all available CodeUpgradeTask services, if the startup mode is UPDATE. Those services are executed sequentially in order of their OSGi `service.ranking` service property, and many of them set a property under `/var/upgrade/status` to make sure they run only once.

While the upgrade services are running, any HTTP request to AEM returns a 503 status with a descriptive message.

DataStore Migration

With AEM 6, when running the upgrade script some decisions about the DataStore migration must be made. There are two major options available:

- Migrate the whole repository including the binaries and migrate them into the new TarMK and Tar based DataStore.
- Only the repository data is migrated to TarMK, binaries remain in the FileStore as in the previous versions of AEM.

Possible reasons for selecting option 2:

- The file based data store is too large to be migrated; migration may take a long time.
- When testing migration, not migrating the binaries speeds up the migration significantly.

16

Higher Level APIs

EXERCISE 16.1 - Explore some higher level AEM APIs

When you are developing with AEM you will very probably be using low-level APIs in order to develop all those features that do not come out of the box with AEM. But you have to keep in mind that maybe you don't need to use any Sling or JCR API in order to achieve your goals. Instead of reinventing the wheel it would be easier for you to check out our high level APIs and see if you can use those instead.

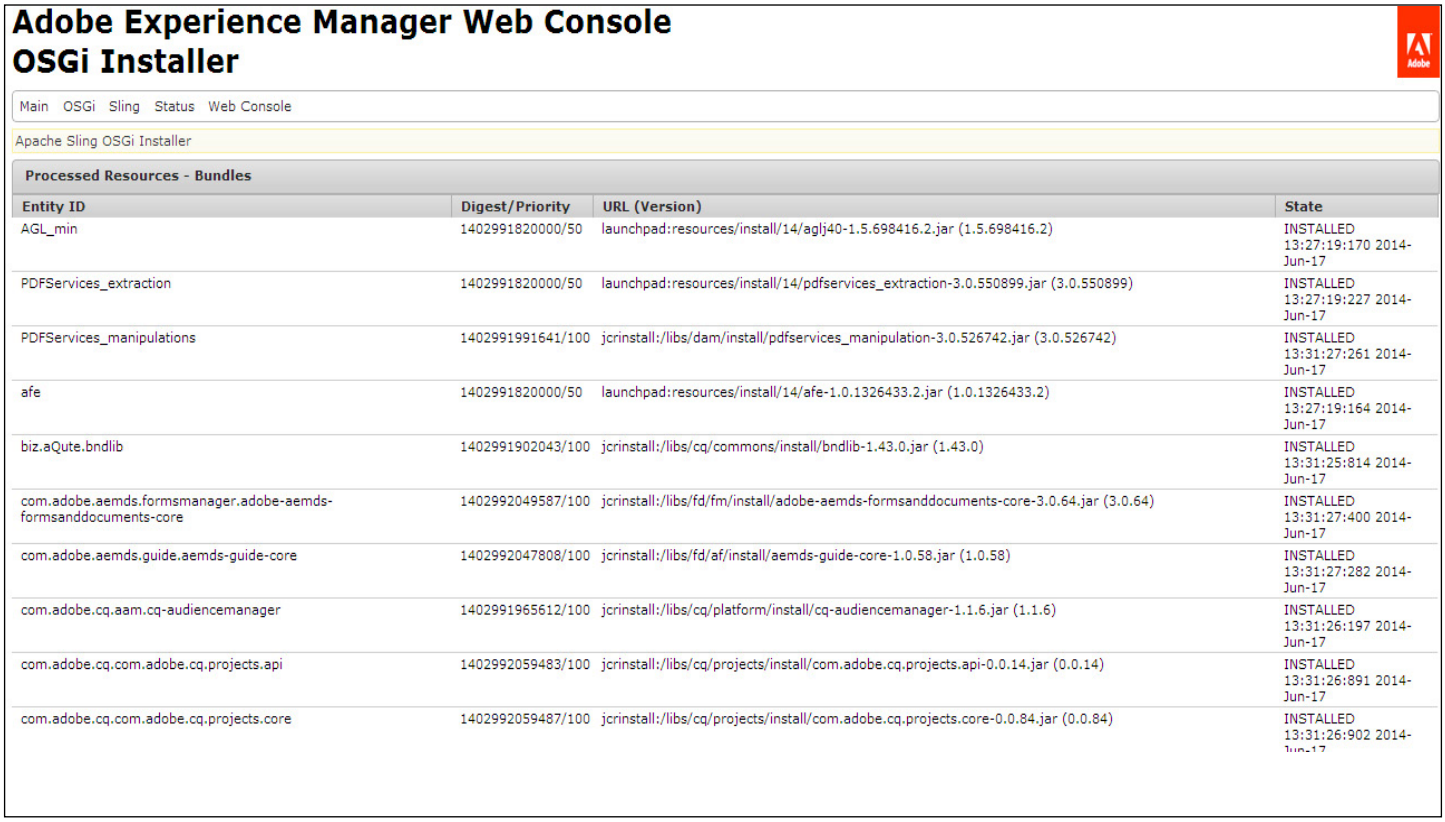
As stated during the developer training you should get familiar with all the classes and methods included in global.jsp. Those are:

- **ComponentContext** : The current component context object of the request (com.day.cq.wcm.api.components.ComponentContext interface).
- **Component** : The current AEM component object of the current resource (com.day.cq.wcm.api.components.Component interface).
- **CurrentDesign**: The current design object of the current page (com.day.cq.wcm.api.designer.Design interface).
- **CurrentPage**: The current AEM Sites page object (com.day.cq.wcm.api.Pageinterface).
- **CurrentNode**: The current JCR node object (javax.jcr.Node interface).
- **CurrentStyle**: The current style object of the current cell (com.day.cq.wcm.api.designer.Style interface).

- **Designer**: The designer object used to access design information (com.day.cq.wcm.api.designer.Designer interface).
- **EditContext**: The edit context object of the AEM component (com.day.cq.wcm.api.components.EditContext interface).
- **PageManager**: The page manager object for page level operations (com.day.cq.wcm.api.PageManager interface).
- **PageProperties**: The page properties object of the current page (org.apache.sling.api.resource.ValueMap).
- **Properties**: The properties object of the current resource (org.apache.sling.api.resource.ValueMap).
- **Resource**: The current Sling resource object (org.apache.sling.api.resource.Resource interface).
- **ResourceDesign**: The design object of the resource page (com.day.cq.wcm.api.designer.Design interface).
- **ResourcePage**: The resource page object (com.day.cq.wcm.api.Pageinterface).

The WCM API is not reduced to the few Classes in global.jsp, there are many more Classes which will help you to create new powerful features, procedures and applications; all that in just a few lines of code.

But this is not just about the WCM API, some other high-level APIs are included with AEM, the full list of them can be seen in the AEM Web Console, in the bundle section or in the OSGi installer section:



In there you will find all the different packages corresponding to the high level APIs that are available to you. You will find the packages corresponding to collaboration, analytics, audit, authentication, i18n, mailing, packaging, personalization, replication, search, security, statistics, spellchecker, tagging, DAM, mobile support, workflow etc.... etc....

Lets take a look to some of those classes, and use them in two very simple exercises.

EXERCISE 16.2 - Creating, tagging and activating a page using high-level APIs

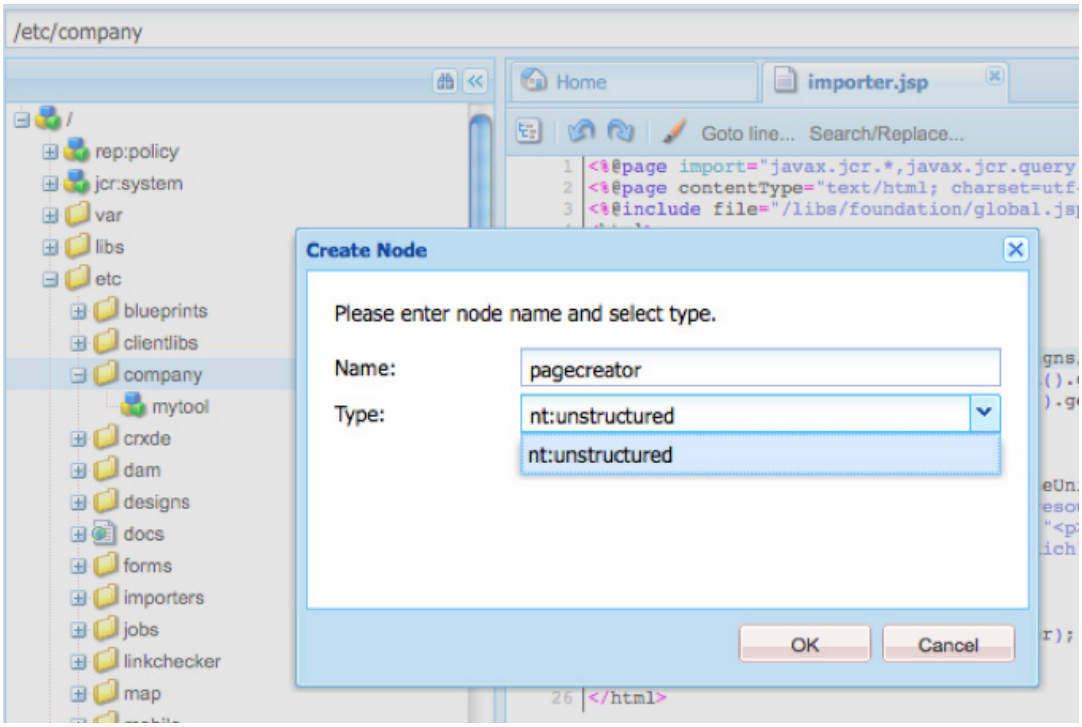
In our first exercise we will use the PageManager, TagManager and Replicator classes to create, tag and activate a page. We will do this by using a simple jsp page and with the help of our high level API. As you will see this can be made in a few lines of code.

You will note that based on the very same example you could create a procedure or a scheduler job that would allow you to create several pages with content that could be imported from external sources (something similar to the scaffolding tool), this is a very recurrent question during the developer training.

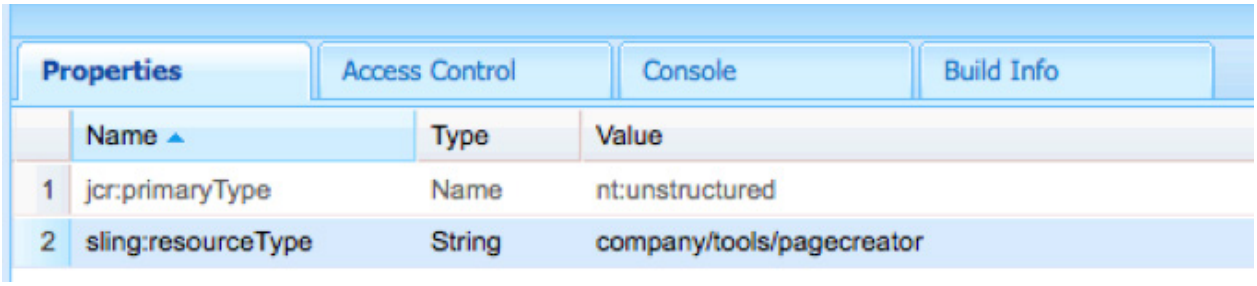
Steps

1. Open CRXDE and add an unstructured node "pagecreator" under the node "/etc/company" node

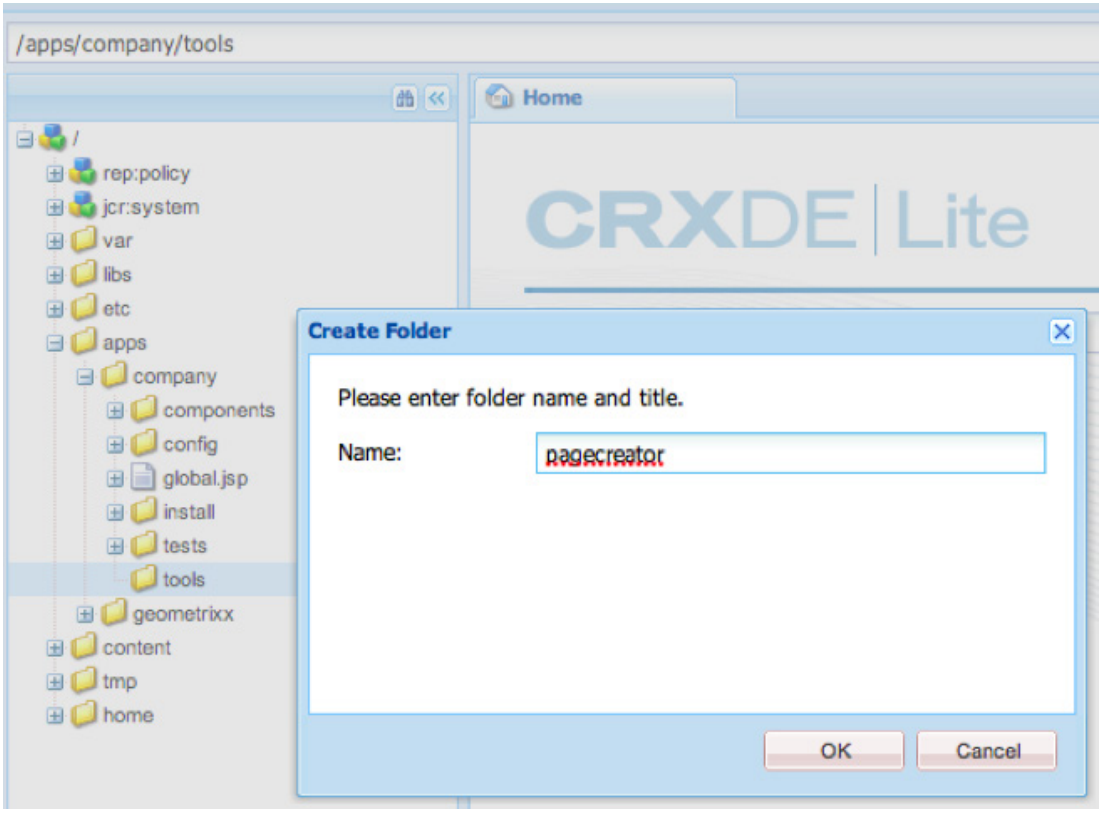
created in the previous exercise.



The sling:resourceType of "pagecreator" shall be "company/tools/pagecreator"



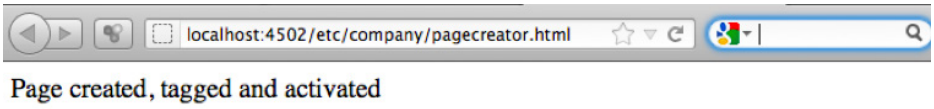
2. Create a new folder "pagecreator" under the folder /apps/company/tools created in the previous exercise.



3. Create a the file pagecreator.jsp under the newly created folder and enter the following code:

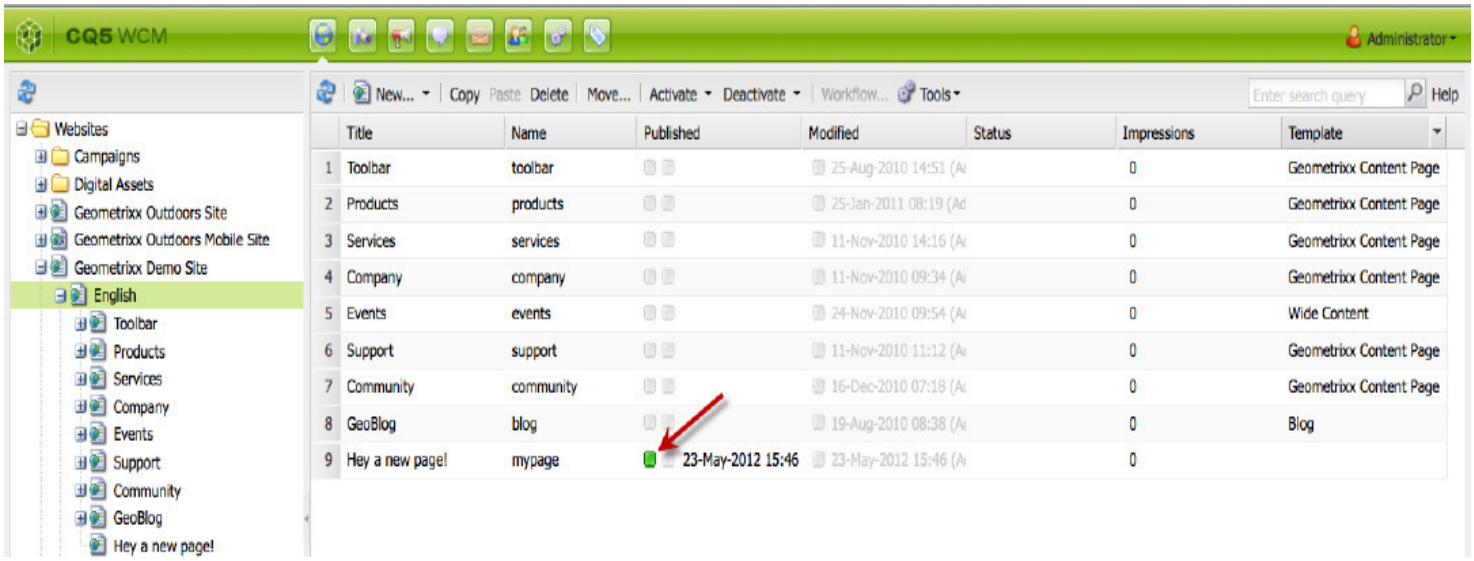
```
<%@page import="com.day.cq.tagging.*,
                com.day.cq.wcm.api.*,
                com.day.cq.replication.*"%>
<%@page contentType="text/html; charset=utf-8"%>
<%@include file="/libs/foundation/global.jsp"%>
<html>
<head>
<title>Page creator </title>
</head>
<body>
<%
    // pageManager is defined in /libs/foundation/global.jsp
    Page p=pageManager.create("/content/geometrixx/en",
                              "mypage",
                              "/apps/geometrixx/template/contentpage",
                              "Hey a new page!");

    // TagManager can be retrieved via adaptTo
    TagManager tm=resource.getResourceResolver().adaptTo(TagManager.class);
    tm.setTags( p.getContentResource(),
                new Tag[]{tm.resolve("/etc/tags/marketing/interest/")},
                true);

  // Replicator is exposed as a service
  Replicator r= sling.getService(Replicator.class);
  r.replicate( currentNode.getSession(),
               ReplicationActionType.ACTIVATE,
               p.getPath());
%>

    <p> Page created, tagged and activated </p>
</body>
</html>
```

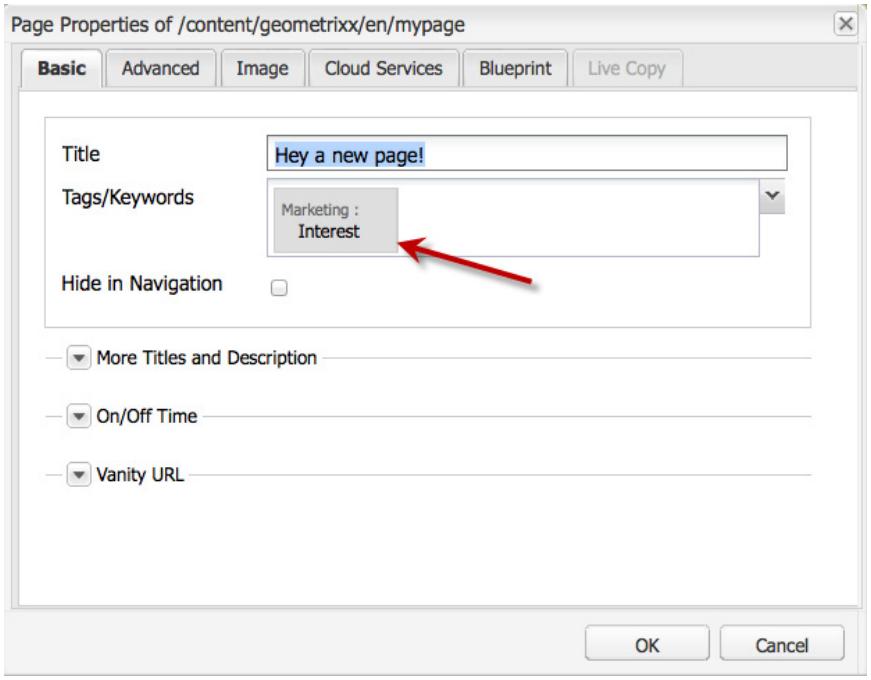
4. Execute the script at http://localhost:4502/etc/company/pagecreator.html



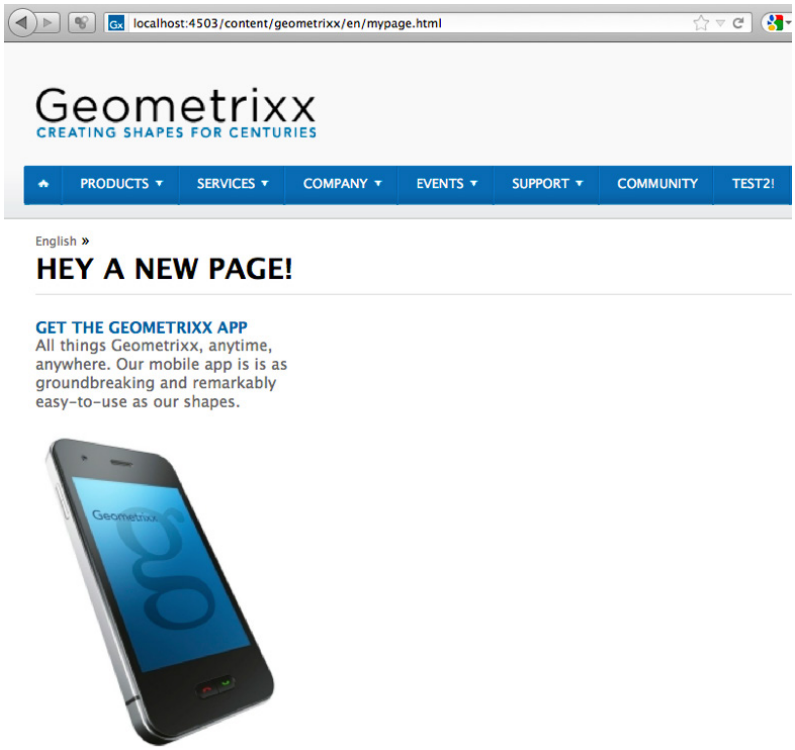
Page created, tagged and activated

5. Inspect the newly created page in the site admin.





6. Open the new page in the publish server.



EXERCISE 16.3 - Creating, tagging and activating an Asset using high- level APIs

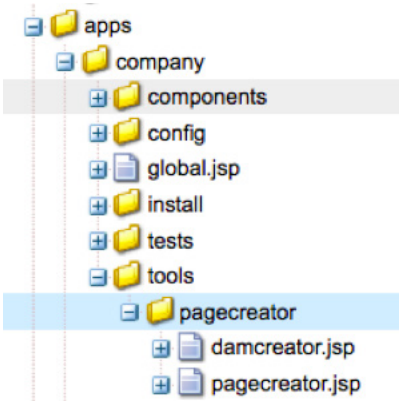
In this second exercise we will use the AssetManager, Asset, TagManager and Replicator classes to create, tag and activate an Asset. Again we will use our high level API to achieve this is a few lines of code.

The asset that we are going to create is an image hoted in a website, of course the same approach could be use to create any sort of asset. Again this could be use as part of a procedure or a scheduler job that would allow you to create several images directly in your DAM (for migration purposes for example).

Steps

1. In order to save us a few steps we will reuse the same pagecreator node (/etc/company pagecreator) created in the last exercise as the pagecreator node under /apps/company/tools/pagecreator.

Taking advantage of the structure already created before we will create in CRXDE a new jsp called damcreator.jsp under /apps/company/tools/pagecreator.

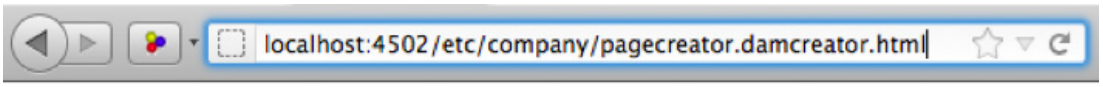


2. Enter the following code:

```
<%@page import="com.day.cq.tagging.*,
                com.day.cq.dam.api.*,
                com.day.cq.replication.*,
                java.net.URL"%>
<%@page contentType="text/html; charset=utf-8"%>
<%@include file="/libs/foundation/global.jsp"%>
<html>
<head>
<title>Asset creator </title>
</head>
<body>
<%
   // we fetch the URL of the image that we want to import
   URL fileURL= new URL("http://dev.day.com/content/dam/portal/banner-deepdive.png");

   // AssetManager can be retrieved via adaptTo()
   AssetManager am = resource.getResourceResolver().adaptTo(AssetManager.class);

   // creating the asset
   Asset myAsset=am.createAsset("/content/dam/geometrixx/myImage.png",
                                fileURL.openStream(),
                                 "image/png",
                                 true);

   // TagManager can be retrieved via adaptTo
   TagManager tm=resource.getResourceResolver().adaptTo(TagManager.class);

   // the tags of the Assets are stored under jcr:content/metadata
   String pathMetadata=myAsset.getPath()+"/jcr:content/metadata";
   Resource resourceImage=resource.getResourceResolver().resolve(pathMetadata);
   tm.setTags( resourceImage,
               new Tag[]{tm.resolve("/etc/tags/stockphotography/technology/")},
               true);

  // Replicator is exposed as a service
  Replicator r= sling.getService(Replicator.class);
  r.replicate( currentNode.getSession(),
               ReplicationActionType.ACTIVATE,
               myAsset.getPath());
%>

   <p> Asset created, tagged and activated </p>
</body>
</html>
```

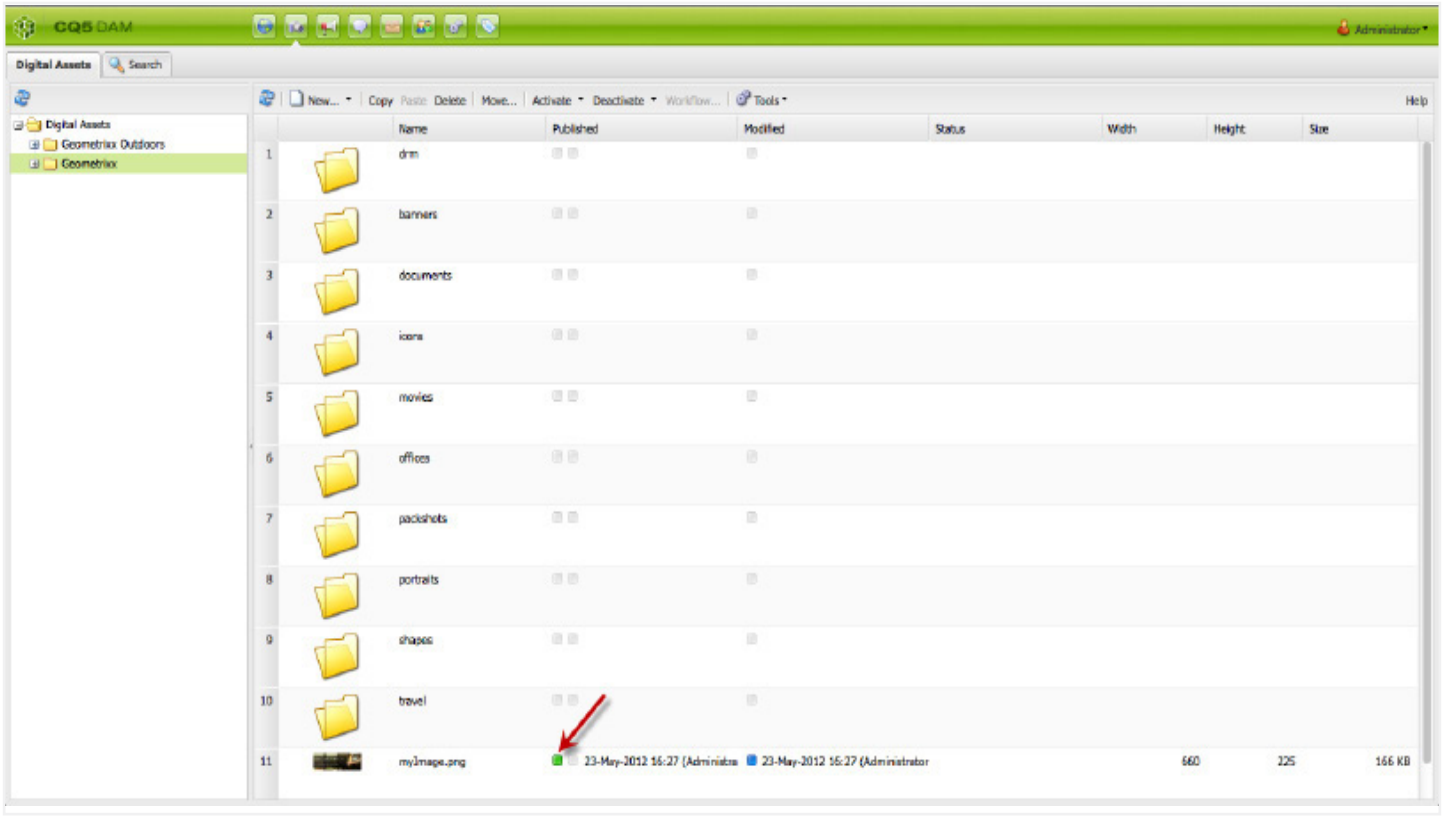
3. In order to execute the script we use the name of our jsp as a selector, let's execute the script by opening the following url:

   http://localhost:4502/etc/company/pagecreator.damcreator.html



Asset created, tagged and activated

4. Inspect the newly created Asset in the site admin under /content/dam.



17

Client Libraries (Extra Credit)

In today's modern Web development comprehensive JavaScript libraries, in conjunction with HTML and CSS, are responsible for some very exciting Web experiences.

Managing these client-side assets can become quite cumbersome, especially since AEM allows authors to select components and templates at their own convenience. Developers can not really plan when and where client-side assets will be used.

Another challenge is that many components and templates require client-assets.

Client- or HTML Libraries

AEM has introduced a very interesting concept: Client Libraries, aka "clientlibs".

Client Libraries are "folders" (nodes of node-type cq:ClientLibraryFolder) that contain the client-side assets, CSS and JS files and required resources, e.g. images or fonts.

There can be unlimited client library folders. The folder content can be loaded individually and at any given time.

Client Library Conventions

Create client libraries either under /etc/clientlibs or within the component folder.

A client-library "folder" is created as a node with node type cq:ClientLibraryFolder, with the following properties:

- jcr:primaryType: cq:ClientLibraryFolder
- categories: an array of names to identify the client-library
- dependencies: an array of categories (dependent client libraries)
- embed: an array of client libraries that will be included

Create sub folders for the CSS and JS files, e.g. /scripts or /styles.

Create a css.txt and/or a js.txt file and add the .css and .js files that will be minified and loaded by templates or components. If the assets are in a sub-folder, the .txt file must start with "#base=sub-folder".

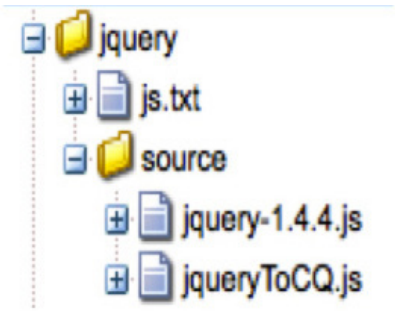
Resources, like images or fonts, that are used by JS or CSS files, are stored in the client library folder.

Client libraries are loaded with the <cq:includeClientLib> tag. The <cq:includeClientLib/> tag Includes a client library, which can be a JS, a CSS or a Theme library. For multiple inclusions of different types, for example JS and CSS, this tag needs to be used multiple times in the jsp.

Client libraries can be loaded programmatically with the com.day.cq.widget. HtmlLibraryManager service interface.

Examples of Client Libraries

An example to define jQuery:



Properties of jQuery folder:

jcr:primaryType = cq:ClientLibraryFolder
categories = cq.jQuery (dot-notated names are allowed)

The js.txt file contains:

```
#base=source
jquery-1.4.4.js
jqueryToCQ.js
```

The file starts with defining the folder where the JS files can be found, then the JS files are listed that will be loaded.

This client library can now be used as a "dependent" one or it can be "embedded" in another client library.

Dependencies: The libraries of categories listed in "dependencies" have to be already included, else the current library will not be included.

Embed: The libraries of categories listed in "embed" will be included to the HTML page as well. If the libraries have already been included, they are ignored.

Include Client Libraries

AEM provides a custom JSP tag, which makes it easy to include client libraries or parts of them: **<cq:includeClientLib>**.

The purpose of the <cq:includeClientLib> tag is to include JS and CSS assets to the HTML page. The parameters are:

Categories: A list of comma-separated client library categories. This will include all Javascript and CSS libraries for the given categories. The theme name is extracted from the request.

js: A list of comma-separated client library categories. This will include all Javascript libraries from the listed categories.

css: A list of comma-separated client library categories. This will include all css libraries from the listed categories.

theme: A list of comma-separated client library categories. This will include all theme related libraries (both CSS and JS) for the listed categories. The theme name is extracted from the request.

themed: A flag that indicates if only themed or non themed libraries should be included. If omitted, both sets are included. Only applies to pure JS or CSS includes (not for categories or theme includes).

Examples of <cq:includeClientLib> Tag

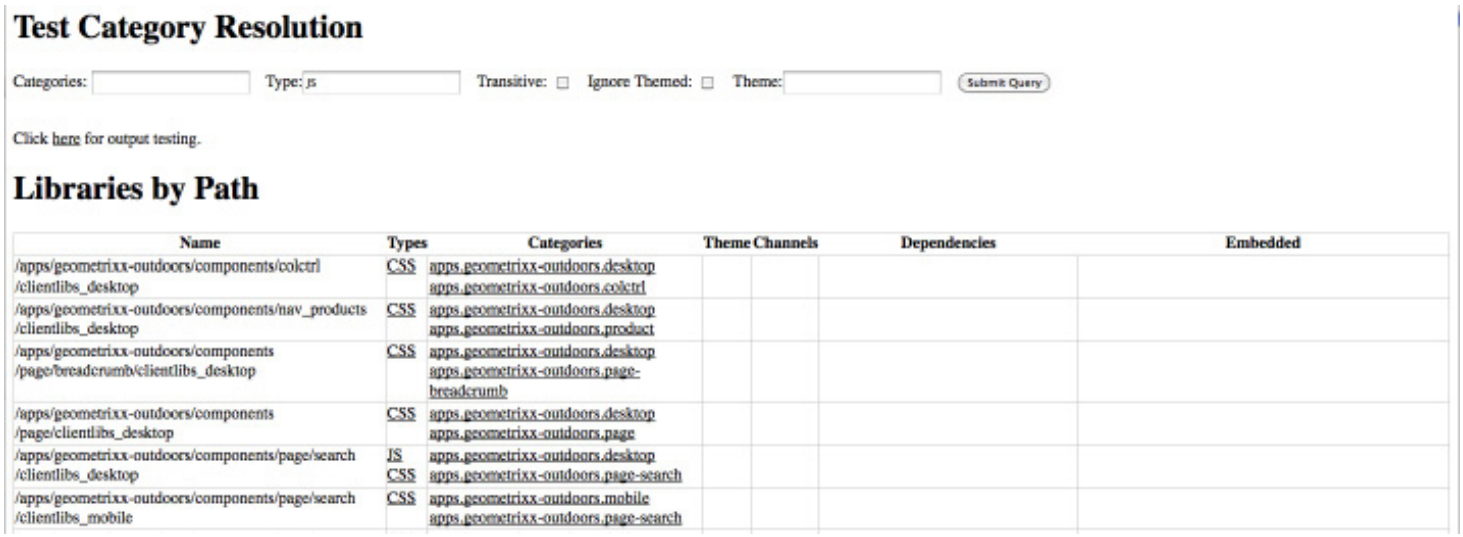
```
<%-- all: js, css and theme (theme-js + css) --%>
<cq:includeClientLib categories="cq.wcm.edit" />

<%-- only js libs --%>
<cq:includeClientLib js="cq.collab.calendar, cq.security" />

<%-- theme only (theme-js + css) --%>
<cq:includeClientLib theme="cq.collab.calendar, cq.security" />

<%-- css only --%>
<cq:includeClientLib css="cq.collab.calendar, cq.security" />
```

Manage Client Libraries

AEM has a tool that lists all the defined client libraries:

http://localhost:4502/libs/cq/ui/content/dumplibs.html.

Dumplibs lists all the defined client libraries and the properties.



Test Category Resolution

Categories: Type: js Transitive: ☐ Ignore Themed: ☐ Theme: Submit Query

Click here for output testing.

Libraries by Path

Name	Types	Categories	Theme	Channels	Dependencies	Embedded
/apps/geometrixx-outdoors/components/colctrl /clientlibs_desktop	CSS	apps.geometrixx-outdoors.desktop apps.geometrixx-outdoors.colctrl				
/apps/geometrixx-outdoors/components/nav_products /clientlibs_desktop	CSS	apps.geometrixx-outdoors.desktop apps.geometrixx-outdoors.product				
/apps/geometrixx-outdoors/components /page/breadcrumb/clientlibs_desktop	CSS	apps.geometrixx-outdoors.desktop apps.geometrixx-outdoors.page-breadcrumb				
/apps/geometrixx-outdoors/components /page/clientlibs_desktop	CSS	apps.geometrixx-outdoors.desktop apps.geometrixx-outdoors.page				
/apps/geometrixx-outdoors/components/page/search /clientlibs_desktop	JS CSS	apps.geometrixx-outdoors.desktop apps.geometrixx-outdoors.page-search				
/apps/geometrixx-outdoors/components/page/search /clientlibs_mobile	CSS	apps.geometrixx-outdoors.mobile apps.geometrixx-outdoors.page-search				

Planning Client Libraries

Since client-libraries can be created basically anywhere it is very important that they are planned carefully. It might make sense to add client libraries to components, since AEM avoids loading same files multiple times and minifies CSS and JS files. However, creating CSS style fragments among a lot of components can make managing CSS very challenging.

Therefore it's recommended to collect client libraries in /etc/clientlibs and place them within a component only if necessary.

Although "dependencies" can be very powerful, it's recommended to not go beyond one level. More levels (e.g. a client library that depends on another one, which depends on another one, etc.) makes it quite cumbersome to maintain.

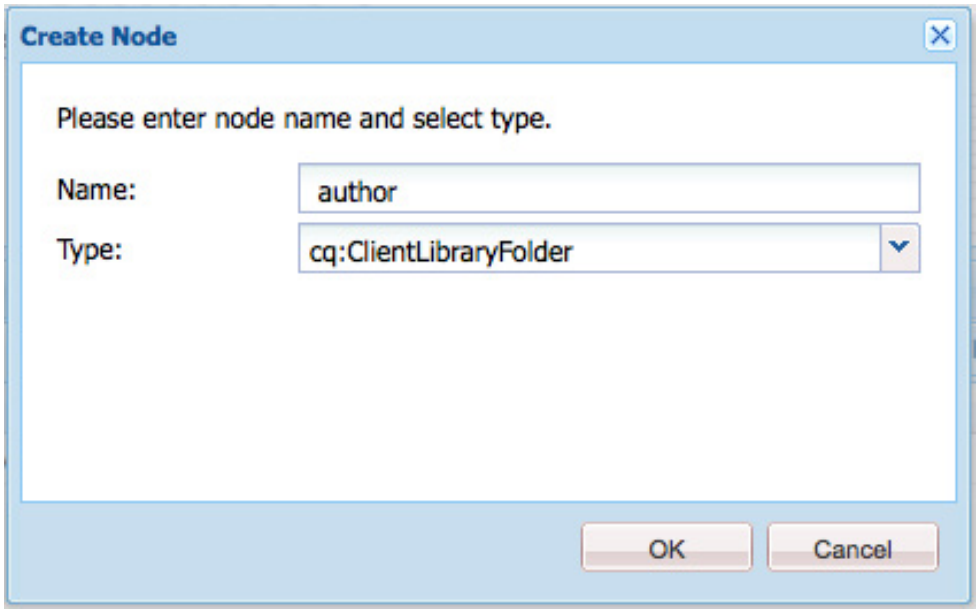
EXERCISE 17.1 - Client Libraries (global)

Goal

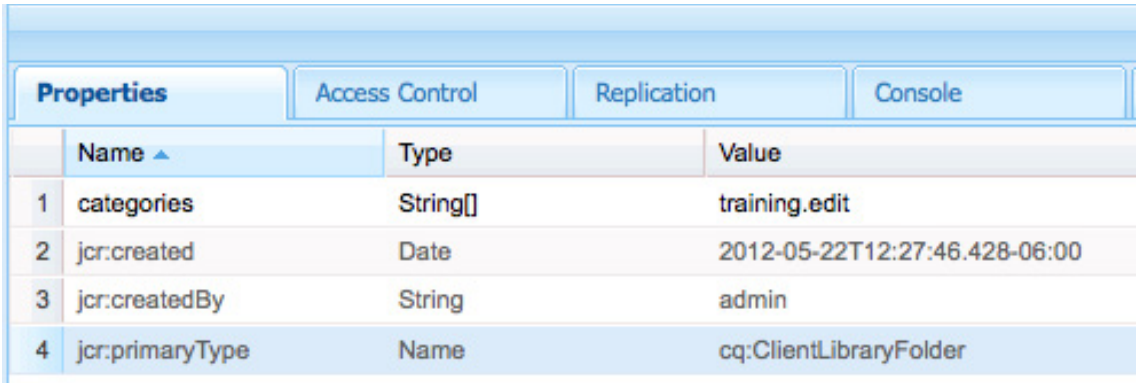
- Manage all client libraries from one place (/etc/clientlibs)
- Render categories individually for author and publish instances

Steps

1. In CRXDE create a new folder in /etc/clientlibs. Usually the folder has the same name as the project in /apps.
2. In the project folder (e.g. /etc/clientlibs/training) create a new node, name it "author" and select node-type cq:ClientLibraryFolder.



3. Add the property "categories" as String array. This defines a reference to the client library we will use later. As value enter "training.edit" and then click "Save".



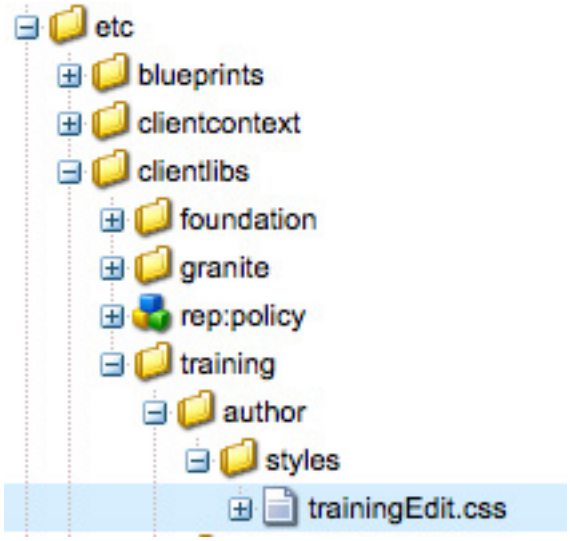
	Name	Type	Value
1	categories	String[]	training.edit
2	jcr:created	Date	2012-05-22T12:27:46.428-06:00
3	jcr:createdBy	String	admin
4	jcr:primaryType	Name	cq:ClientLibraryFolder

4. Repeat the steps 2 and 3 with a client library folder name "publish". Set the categories property for the publish folder to "training".

5. Next we add a CSS file to the author client library and a slightly different one to the publish client library.

6. In /etc/clientlibs/training/author create a new folder "styles"

7. In that folder create a new file trainingEdit.css



8. Enter a style, e.g.

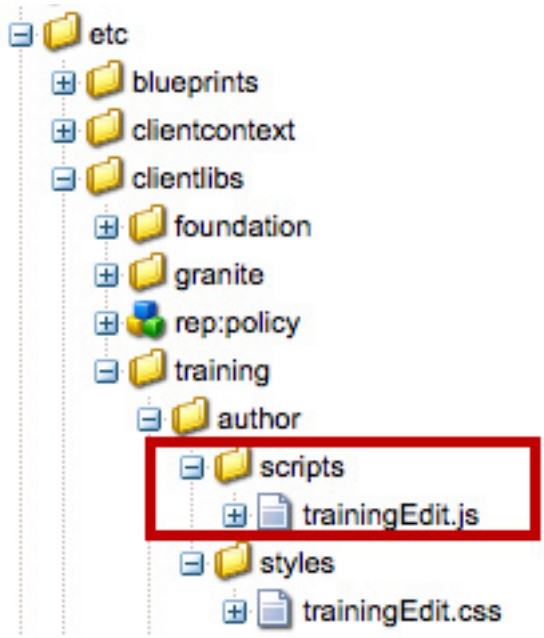
```
/* Training CSS File - Author */
#authorData { color: blue; }
```

9. Repeat for the publish client library. In /etc/clientlibs/training/publish create a new folder "styles", add a new CSS file training.css and add a new style, e.g.

```
/* Training CSS File */
#publishData { color: red; }
```

10. We repeat above steps to create two simple JavaScript files. In/etc/clientlibs/training/author create a new folder "scripts"

11. In that folder create a new file trainingEdit.js



12. Enter a JavaScript snippet, e.g.

```
/* Training JS File - Author */
alert("This is an alert in Author mode");
```

13. Repeat for the publish client library. In /etc/clientlibs/training/publish create a new folder "scripts", add a new JavaScript file training.js and add a new snippet, e.g.

```
/* Training JS File */
alert("This is an alert in Publish mode");
```

14. Since AEM will "minify" the CSS and JS files, we need to create a text file defining which .css and which .js files to load and minify.

15. In /etc/clientlibs/training/author create a file called css.txt. By convention the text file has to be named css.txt.

16. Enter where to find the CSS file(s) (folder name following #base) and then list the file names that should be loaded when this client library is requested.

```
#base=styles
trainingEdit.css
```

17. Repeat this step for /etc/clientlibs/training/publish

```
#base=styles
training.css
```

18. In /etc/clientlibs/training/author create a file called js.txt. By convention this file has to be named js.txt.

19. Enter where to find the JS file(s) (folder name following #base) and then list the file names that should be loaded when this client library is requested.

```
#base=scripts
trainingEdit.js
```

20. And repeat the same for the publish client library

```
#base=scripts
training.js
```

21. The client libraries are now ready to be included into an HTML page. The page will need to import WCMMode to detect which run mode we are using. One way to do this is to create a jsp script file and select it using a selector in the URL.
22. Create a new jsp file called testpage.jsp under the /apps/geometrixx/components/page/ node. Add the following html code:

```
<%@page import="com.day.cq.wcm.api.WCMMode" %>
<%@include file = "/libs/foundation/global.jsp" %>
<html>
  <head>
    <title>Training</title>
    <cq:include script="/libs/wcm/core/components/init/init.jsp"/>
    <cq:includeClientLib categories="training"/>
    <c:if test="<%= WCMMode.fromRequest(slingRequest) == WCMMode.EDIT%>">
      <cq:includeClientLib categories="training.edit"/>
      DEBUG - In Edit mode
    </c:if>
  </head>
  <body>
    <p id="publishData">Publish-mode Data</p>
    <p id="authorData">Author-mode Data</p>
  </body>
</html>
```

Note: In order to load and minify the new client library, we need to restart the HTML Library Manager service (in the OSGi console under the component tab).

23. Now select any page in the geometrixx website with testpage as a selector in the URL, e.g. localhost:4502/content/geometrixx/en.testpage.html

Pop-up messages should appear for both publish and author mode and the colored text should appear on the page.

The html code only explicitly checks for author mode, so the publish mode message and text color always appear. You could try to improve the code as an extra credit exercise. Also, if you have time, try promoting the clientlib to the publish instance, and activate and test the page there to check that the publish mode works as expected.

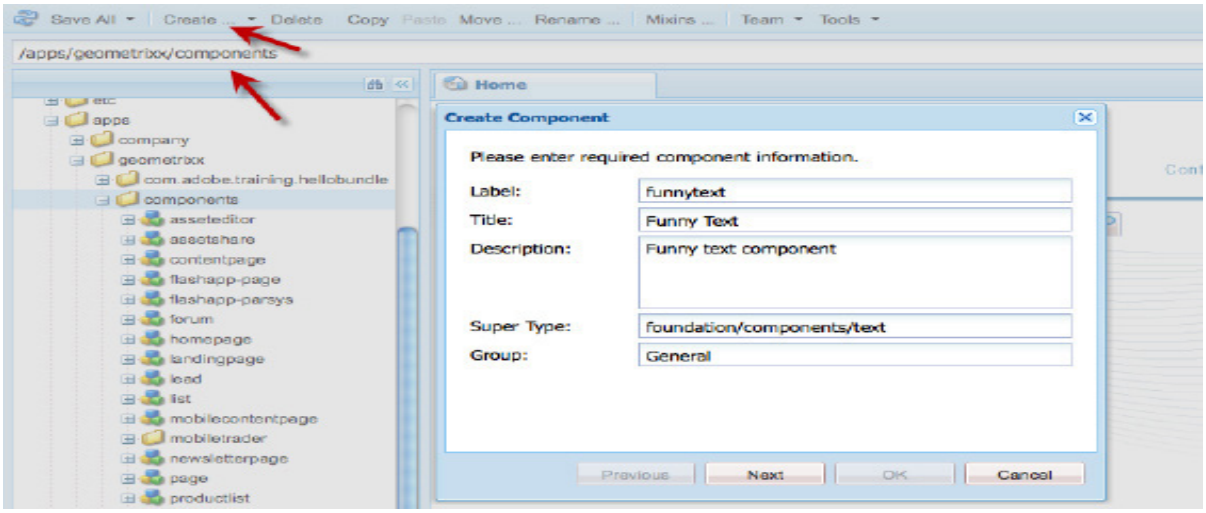
EXERCISE 17.2 - Client Libraries (below Components)

Goal

- Create a new Client Library in Geometrixx
- Make sure it gets embedded in rendered HTML pages

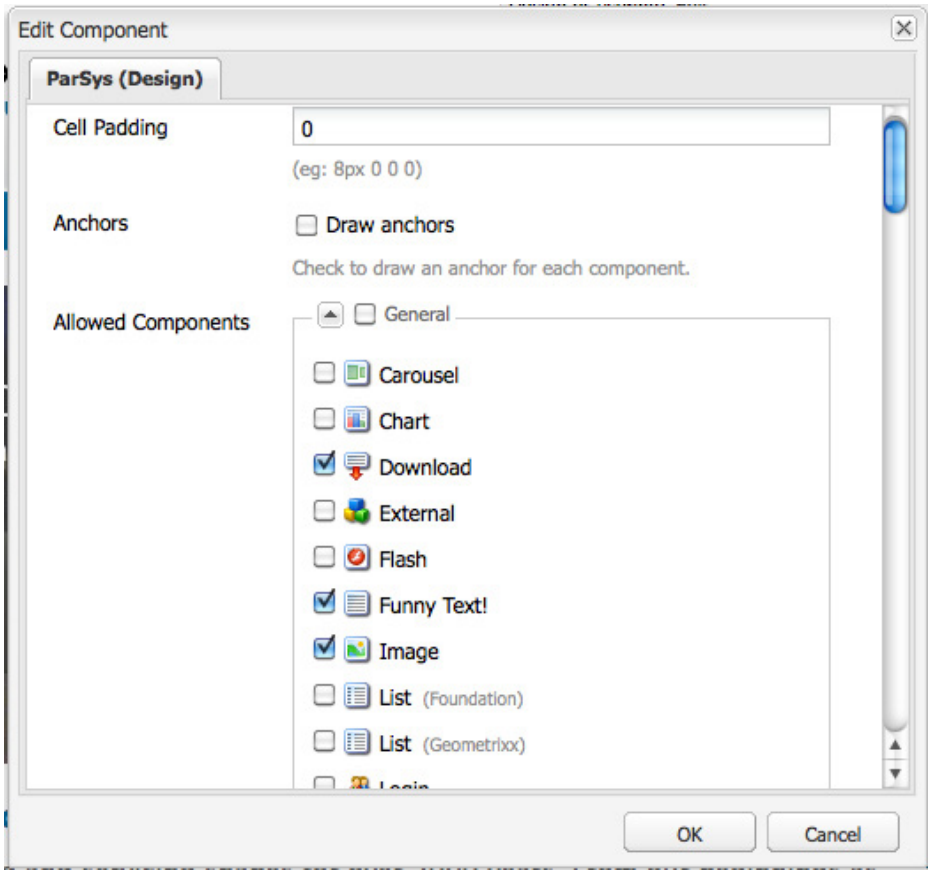
Steps

1. In CRXDE create a new component in the Geomterixx app that inherits from the foundation text component.

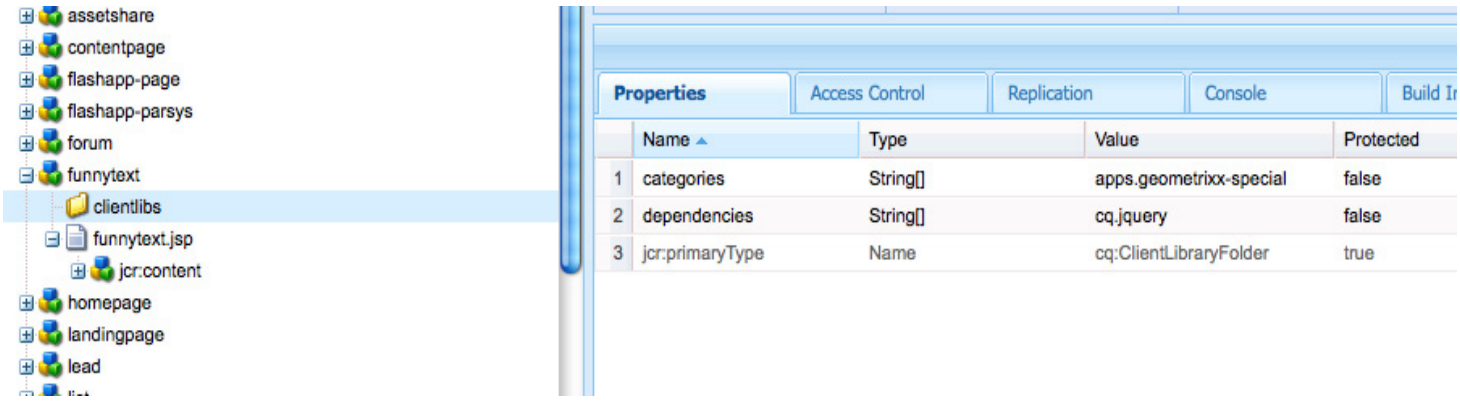


2. Do not forget to add some default output in funnytext.jsp (so that something gets rendered) and to enable the component in Design Mode.

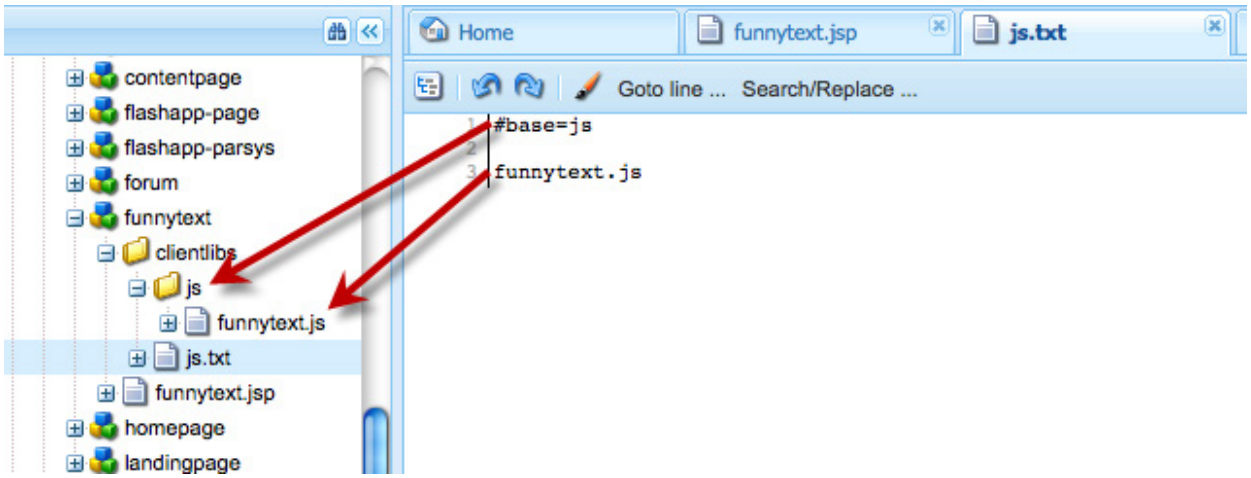
```
<%--

 Funny Text component.

--%><%
%><%@include file="/libs/foundation/global.jsp"%><%
%><%@page session="false" %>

  <span class="funny">
    <cq:text property="text"/>
  </span>
```



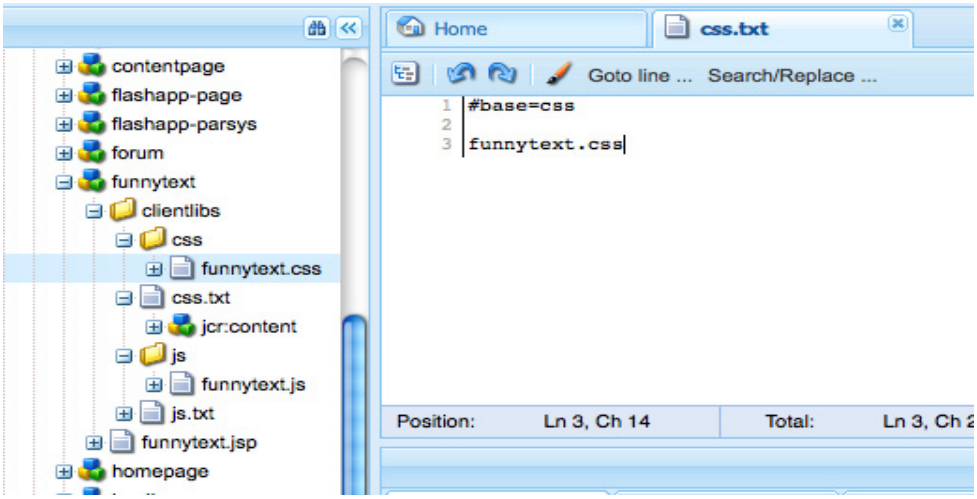
3. Create a clientlibs node (note the resource type!). Give it a new category and let it depend on the ootb jQuery:



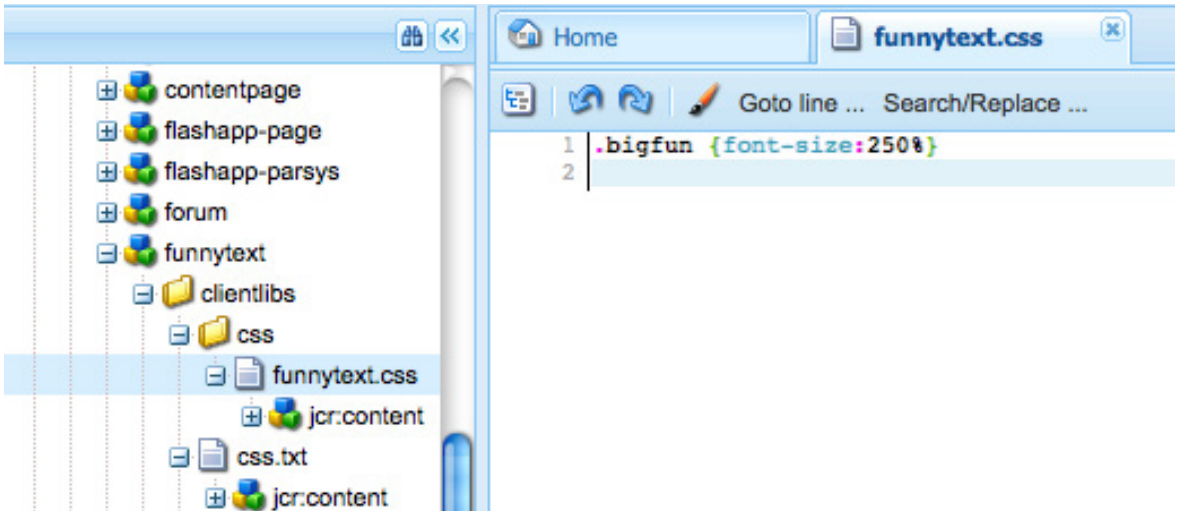
4. Create a js.txt file that references all relevant js files for this component and create the file funnytext.js to hold the actual js code



5. Analogous for css:



6. In funnytext.css add a css class:



7. Edit funnytext.js and add a jQuery effect to the rendered text:

```
jQuery(function($) {
    $('.funny').hover(function() {
    console.log(this);
      $('.funny p').addClass("bigfun");
  }, function() {
     $('.funny p').removeClass("bigfun");
  });
});
```

(remember that you added a dependency to jQuery so you can assume it to be defined)

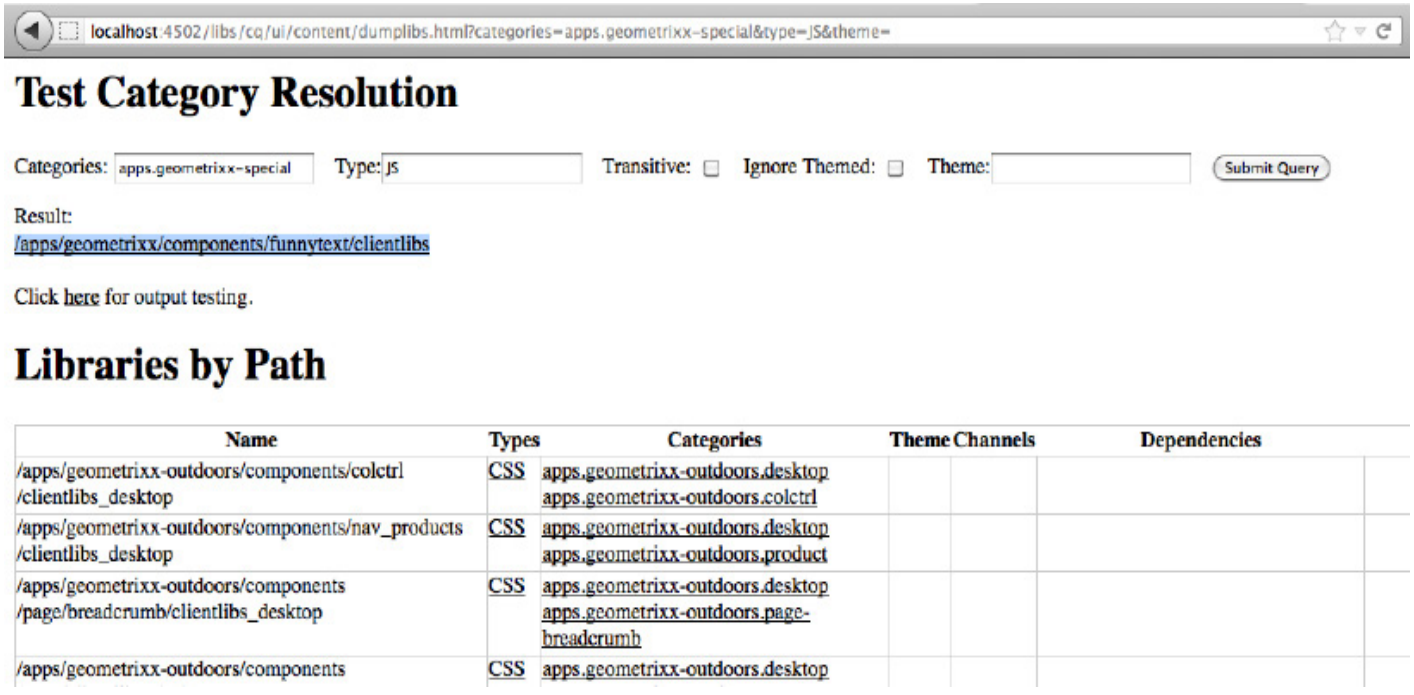
8. Edit the headlibs.jsp in the Geometrixx page local Super Type: ***/apps/geometrixx/components/page/headlibs.jsp***. Add the category (name) of your new clientlib: apps.geometrixx-special.

```
<%--
  Copyright 1997-2008 Day Management AG
  Barfuesserplatz 6, 4001 Basel, Switzerland
  All Rights Reserved.

  This software is the confidential and proprietary information of
  Day Management AG, ("Confidential Information"). You shall not
  disclose such Confidential Information and shall use it only in
  accordance with the terms of the license agreement you entered into
  with Day.

  ==============================================================
==================

  Includes the scripts and css to be included in the head tag

  ==============================================================
==================

--%><%@include file="/libs/foundation/global.jsp" %><%
%><%@ page import="java.util.Date,
                  java.text.SimpleDateFormat,
                  com.day.cq.commons.Doctype,
                  org.apache.commons.lang3.StringEscapeUtils,
                  org.apache.commons.lang3.StringUtils" %><%
String xs = Doctype.isXHTML(request) ? "/" : "";

if(!properties.get("cq:lastModified", "").equals("")) {
    SimpleDateFormat sdf = new SimpleDateFormat("d MMM yyyy HH:mm:ss
z");
    String date = sdf.format(properties.get("cq:lastModified", Date.
class) );
    %><meta http-equiv="Last-Modified" content="<%= StringEscapeUtils.
escapeHtml4(date) %>"<%=xs%>><%
}

if(!properties.get("cq:lastModifiedBy", "").equals("")) {
    %><meta name="author" content="<%= StringEscapeUtils.escapeHtm-
l4(properties.get("cq:lastModifiedBy", "")) %>"<%=xs%>><%
}

if(!properties.get("jcr:title", "").equals("")) {
    %><meta name="title" content="<%= StringEscapeUtils.escapeHtm-
l4(properties.get("jcr:title", "")) %>"<%=xs%>><%
}
```

```
if(!properties.get("subtitle", "").equals("")) {
   %><meta name="subtitle" content="<%= StringEscapeUtils.escapeHtm-
l4(properties.get("subtitle", "")) %>"<%=xs%>><%
}

if(properties.get("cq:tags", new String[0]).length > 0) {
   %><meta name="tags" content="<%= StringEscapeUtils.escapeHtml4(
StringUtils.join(properties.get("cq:tags", new String[0]), ",") )
%>"<%=xs%>><%
}
%>
<cq:includeClientLib categories="apps.geometrixx-main,
apps.geometrixx-special"/>
<%  currentDesign.writeCssIncludes(pageContext); %>
```

9. Add the component to a Geometrixx page and verify the rollover effect.

10. Verify that your Clientlib appears in the clientlib console at localhost:4502/libs/cq/ui/content/dumplibs.html



Appendix 1

Useful API definitions

EXERCISE - Useful API definitions

Goal

- During this training you will use several packages and APIs, here is a list of the most important ones and a short description of them, you are of course invited to take a closer look to all the methods available to you

Package log.slf4j: The Simple Logging Facade for Java or (SLF4J) serves as a simple facade or abstraction for various logging frameworks.

- *Interface org.slf4j.Logger*: The main user entry point of SLF4J API. It is expected that logging takes place through concrete implementations of this interface.

- *Class org.slf4j.LoggerFactory*: A utility class producing Loggers for various logging APIs, most notably for log4j.

Package org.apache.felix.scr.annotations: The maven-scr-plugin uses the SCR annotations from the corresponding subproject at Apache Felix, they are all defined in here.

- *Annotation org.apache.felix.scr.annotations.Component:* This annotation is used to declare the <component> element of the OSGi component declaration.

- *Annotation org.apache.felix.scr.annotations.Activate*: This annotation is used to declare the method that it is executed at component's activation.

- *Annotation org.apache.felix.scr.annotations.Deactivate:* This annotation is used to declare the method that it is executed at component's de-activation.

- *Annotation org.apache.felix.scr.annotations.Reference*: The Reference annotation defines references to other services made available to the component by the Service Component Runtime.

- *Annotation org.apache.felix.scr.annotations.Property*: The Property annotation defines properties which are made available to the component through the

- *Annotation org.apache.felix.scr.annotations.Property*: The Property annotation defines properties which are made available to the component through the ComponentContext. getProperties() method. Additionally properties may be set here to identify the component if it is registered as a service.

- *Annotation org.apache.felix.scr.annotations.Service*: The Service annotation defines whether and which service interfaces are provided by the component.

- *Annotation org.apache.felix.scr.annotations.sling.SlingServlet*: The SlingServlet annotation marks servlet classes as felix SCR component, and allows to configure sling resource resolving mapping.

Package javax.jcr: Provides all the well-know interfaces and classes for the Content Repository for Java Technology.

- *Interface javax.jcr.Repository*: The entry point into the content repository. The Repository object is usually acquired through the RepositoryFactory.

- *Interface javax.jcr.Session*: The Session object provides read and (in level 2) write access to the content of a particular workspace in the repository.

- *Interface javax.jcr.Node:* The Node interface represents a node in a workspace.

- *Interface javax.jcr.Value*: A generic holder for the value of a property. A Value object can be used without knowing the actual property type (STRING, DOUBLE, BINARY etc.)

- I*nterface javax.jcr.ValueFactory:* The ValueFactory object provides methods for the creation of Value objects that can then be used to set properties.

Package javax.jcr.query.qom: Provides interfaces and classes for content repository Query Object Model.

- *Interface javax.jcr.query.qom.QueryObjectModel:* The JCR query object model describes the queries that can be evaluated by a JCR repository independent of any particular query language, such as SQL.

- *Interface javax.jcr.query.qom.QueryObjectModelFactory:* A QueryObjectModelFactory creates instances of the JCR query object model.

- *Interface javax.jcr.query.qom.Constraint:* Filters the set of node-tuples formed by evaluating the query's selectors and the joins between them. To be included in the query results, a node-tuple must satisfy the constraint.

- *Interface javax.jcr.query.qom.Selector*: Selects a subset of the nodes in the repository based on node type.

Package javax.jcr.observation: Provides interfaces and classes for content repository event observation functionality.

- *Interface javax.jcr.observation.Event:* An event fired by the observation mechanism.

- *Interface javax.jcr.observation.EventIterator:* Allows easy iteration through a list of Events with nextEvent as well as a skip method inherited fromRangeIterator.

- *Interface javax.jcr.observation.EventListener:* An EventListener can be registered via the ObservationManager object. Event listeners are notified asynchronously, and see events after they occur and the transaction is committed.

- *Interface javax.jcr.observation.ObservationManager:* Acquired via Workspace. getObservationManager(). Allows for the registration and deregistration of event listeners.

Package javax.jcr.security: Provides interfaces and classes for content repository access control management functionality.

- *Interface javax.jcr.security.AccessControlList:* The AccessControlList is an AccessControlPolicy representing a list of access control entries.

- *Interface javax.jcr.security.AccessControlManager :* The AccessControlManager object provides methods for access control discovery and for assigning access control policies.

- *Interface javax.jcr.security.AccessControlPolicyIterator:* The AccessControlPolicyIterator allows easy iteration through a list of AccessControlPolicys with nextAccessControlPolicy as well as a skip method.

- *Interface javax.jcr.security.Privilege:* A privilege represents the capability of performing a particular set of operations on items in the JCR repository.

Package org.apache.jackrabbit.api.security: Package with different interfaces providing jackrabbit specific extensions to the javax.jcr.security package

- *Interfaceorg.apache.jackrabbit.api.security.JackrabbitAccessControlList*: JackrabbitAccessControlList is an extension of the AccessControlList.

Package org.apache jackrabbit.api.security.user: Package with different interfaces used to manage users and groups within the repository.

- *Interface org.apache.jackrabbit.api.security.Authorizable:* The Authorizable is the common base interface for User and Group. It provides access to the Principals associated with an Authorizable and allow to access and modify additional properties such as e.g. full name, e-mail or address.

- *Interface org.apache.jackrabbit.api.security.user.UserManager:* The UserManager provides access to and means to maintain authorizable objects i.e. users and groups.

Package com.day.cq.replication: Package for replication objects within AEM

- *Class com.day.cq.replication.ReplicationAction:* It is the class that is used for control information of a replication.

- *Enum com.day.cq.replication.ReplicationActionType:* The type of replication action.

Package org.apache.sling.api: Provides several Sling network orientedinterfaces

- *Interfaceorg. apache.sling.api.SlingHttpServletRequest:* The SlingHttpServletRequest defines the interface to provide client request information to a servlet.

- *Interface org.apache.sling.api.SlingHttpServletResponse:* The SlingHttpServletResponse defines the interface to assist a servlet in creating and sending a response to the client.

- *Class org.apache.sling.api.servlets.SlingSafeMethodsServlet:* Helper base class for read-only Servlets used in Sling.

- *Interface org.apache.sling.api.resource.ResourceResolver:* The ResourceResolver defines the service API which may be used to resolve Resource objects. The resource resolver is available to the request processing servlet through the SlingHttpServletRequest. getResourceResolver() method.

- *Interface org.apache.sling.api.resource.ResourceResolverFactory:* The ResourceResolverFactory defines the service API to get and create ResourceResolvers.

Package org.apache.sling.jcr.api: Useful jcr related classes

- *Interface org.apache.sling.jcr.api.SlingRepository:* The SlingRepository extends the standard JCR repository interface with two methods: getDefaultWorkspace() and loginAdministrative(java.lang.String). This method ease the use of a JCR repository in a Sling application in that the default (or standard) workspace to use by the application may be configured and application bundles may use a simple method to get an administrative session.

Package org.apache.sling.commons.json: Library for JSON objects used in Sling

- *Class org.apache.sling.commons.json.JSONObject:* A JSONObject is an unordered collection of name/value pairs.

- *Class org.apache.sling.commons.json.JSONArray:* A JSONArray is an ordered sequence of values.

Package org.apache.sling.commons.osgi: Library for OSGi objects used in Sling.

- *Class org.apache.sling.commons.osgi.OsgiUtil:* The OsgiUtil is a utility class providing some useful utility methods, mostly used to retrieve service reference properties.

Package org.osgi.service.event: The OSGi event Admin Package.

- *Class org.osgi.service.event.Event:* The Event class defines objects that are delivered to EventHandler services which subscribe to the topic of the event.

- *Interface org.osgi.service.event.EventHandler*: Listener for events.

Package org.osgi.service.component: The OSGi component Admin Package.

- *Interface org.osgi.service.component.ComponentContext:* A Component Context object is used by a component instance to interact with its execution context including locating services by reference name. Each component instance has a unique Component Context.

Package org.junit: Provides JUnit core classes and annotations.

- *Annotation org.junit.Test:* The Test annotation tells JUnit that the public void method to which it is attached can be run as a test case.

Package org.junit.runner: Provides classes used to describe, collect, run and analyze multiple tests.

- *Annotation org.junit.runner.RunWith:* When a class is annotated with @RunWith or extends a class annotated with @RunWith, JUnit will invoke the class it references to run the tests in that class instead of the runner built into

  JUnit.
  org.junit.Assert

Package org.apache.sling.junit.annotation: Provides Sling Testunner classes to use with JUnit.

- *Class org.apache.sling.junit.annotations.SlingAnnotationsTestRunner:* TestRunner which uses a TestObjectProcessor to handle annotations in test classes. A test that has RunWith=SlingAnnotationsTestRunner can use @TestReference, for example, to access OSGi services.

- *Class org.apache.sling.junit.annotations.TestReference:* Annotation used to inject services in test classes. Similar to the Felix @Reference annotation.