ADOBE® CQ 5.5
Advanced Developer

User Training Student Workbook

ADOBE® TRAINING SERVICES

Adobe® CQ Advanced Developer Training Student Workbook

Disclaimer

Adobe reserves the right to revise this publication from time to time, changing content without notice.

Preface – Formatting Conventions

Style	Description	Example
Cross-reference	Cross-reference to external documents.	See the *Microsoft Manual of Style* for *Technical Publications*
GUI Item	User interface items.	Click **Save**.
Keyboard shortcut	Keyboard shortcuts	Press **Ctrl+A**
Mouse Button	Mouse buttons	**Secondary-mouse** button (usually the **right-mouse** button).
Link	Link to anchor-points within the current document and/or external sources.	http://www.day.com
`Code`	Example of programming code.	`If (weather==sunny) smile;`
User Input	Example of text, or commands, that you type.	**ls *.xml**
<Variable User Input>	Example of variable text – you type the actual value needed.	**ls <cq-installation-dir>**
[Optional Parameter]	An optional parameter.	**ls [<option>] [<filename>]**
Computer Output	Logging and error messages, computer responses.	The output is ls: cannot access error.log:
Page reference	Reference to a page, file, or component.	Open the *Company* page

When you see this...	It means do this...
Ctrl+A	Hold down the **Ctrl** key, then press the **A** key.
Right-click	Press the **right-mouse** button (or the **left-mouse** button if your mouse has been configured for left-handed use).
Drag	Hold down the left mouse button while moving the item, then release the mouse button at the new location (or the right mouse button if your mouse has been configured for left-handed use).

Table of Contents

Chapter One

1 OSGi Components, Annotations

OSGi

The Open Services Gateway Initiative (OSGi), also known as the Dynamic Module System 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, who 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.

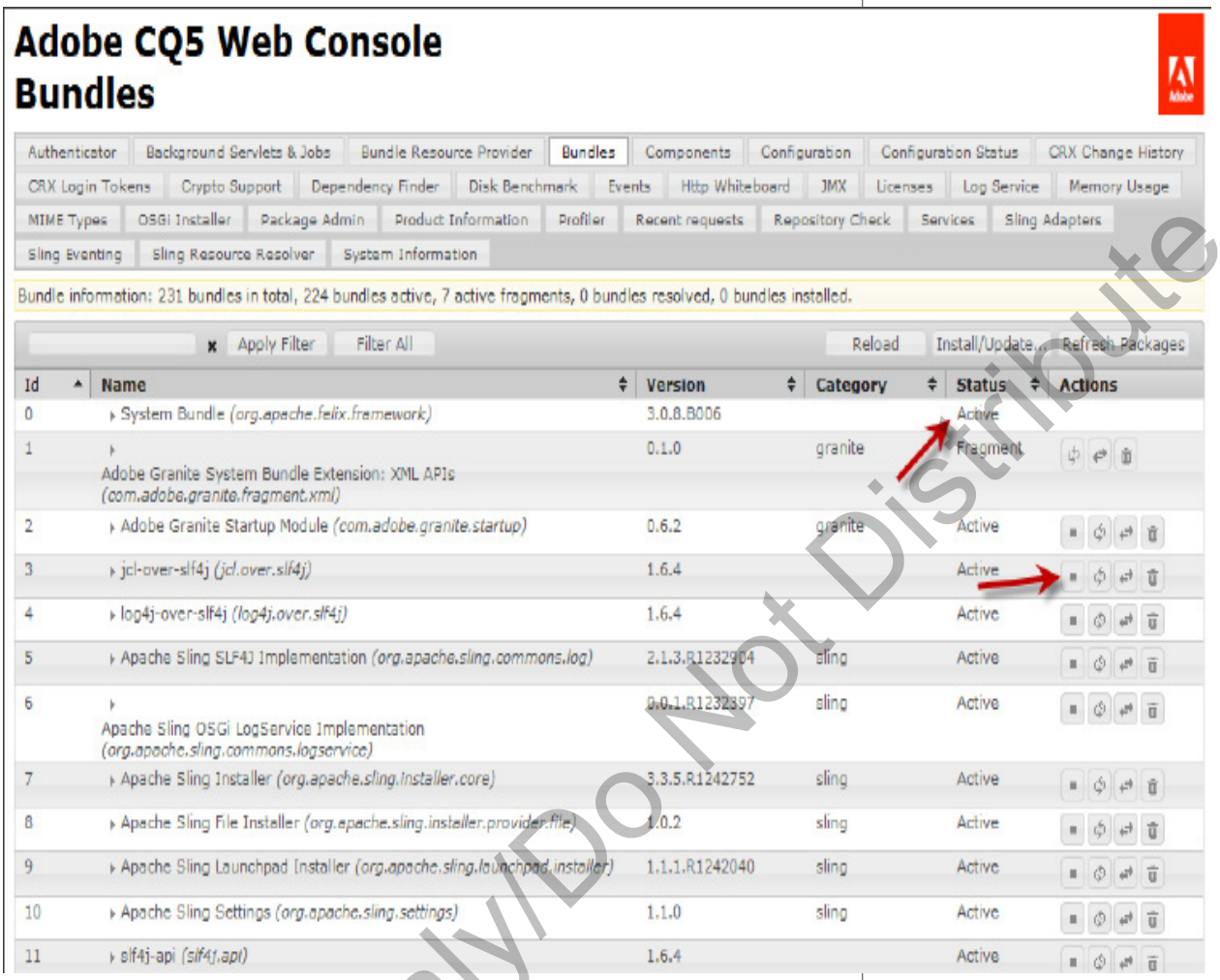
The CQ5 and the CRX applications are shipped and deployed as OSGi bundles. We use Apache Felix as our OSGi container.

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 CQ5 Web Console provides for the dynamic management and configuration of OSGi bundles. You can see the figure below the Bundles tab of the Adobe CQ5 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.

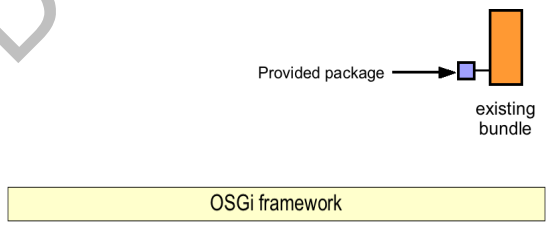
Bundles in OSGi can be installed, updated and uninstalled without taking down the entire application. This makes modular application deployment possible, for example, upgrading parts of a server application — either to include new features or to fix bugs found in production — without affecting the running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

The bundles are loosely coupled.

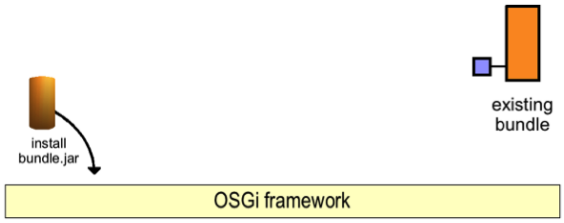
- Package imports and exports with versions
- Dependencies are independent from bundle organization
- "Someone" provides the self-describing package
- OSGi provides error management for unresolved bundles
- Requires modular thinking during application design
- Requires proper meta data and consistent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

Dependency Management Resolution

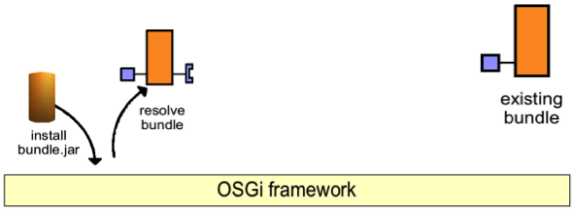
A bundle is present in the container:



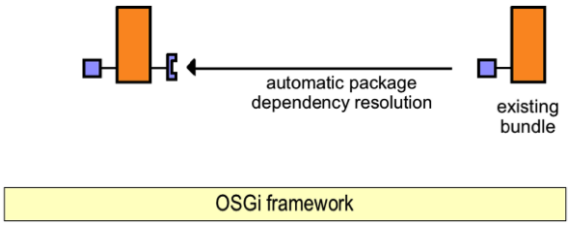
"Someone" provides a new bundle. The bundle is installed into the OSGi container:



The OSGi container resolves the new bundle:



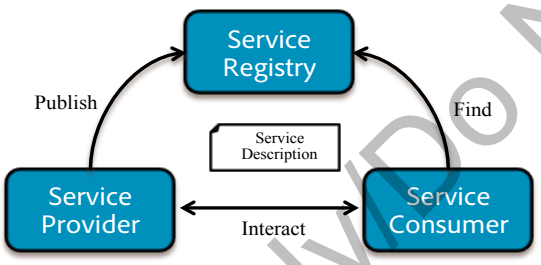
Automatic dependency resolution is provided by the container:



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.

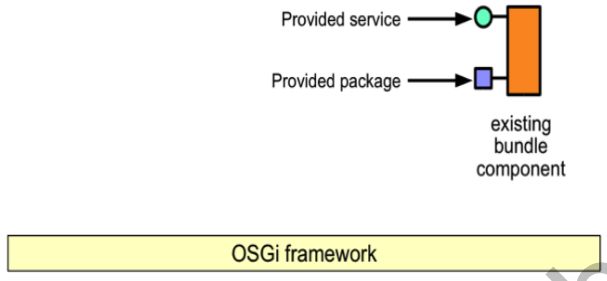
Because you cannot depend on any particular listener or a consumer being present in the container at any particular time, OSGi provides dynamic service look up using the Whiteboard registry pattern. Briefly defined, the Whiteboard registry pattern works as follows:



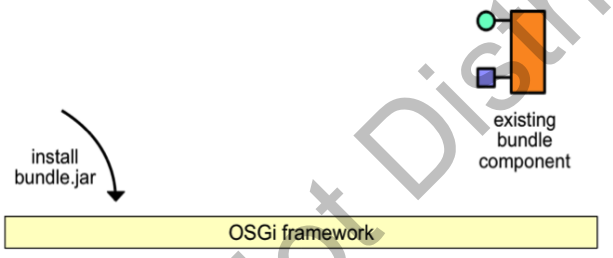
Instead of registering a listener/consumer object C with the service provider S, the consumer creates an object LI that implements OSGi listener interface, providing a method that should be called when the event of interest occurs. When the desired event occurs, S requests a list of all of the services of type LI, 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

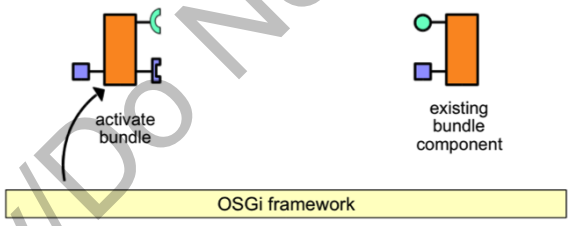
Step 1 - Existing service



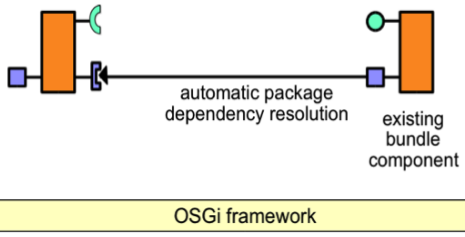
Step 2 - New bundle install



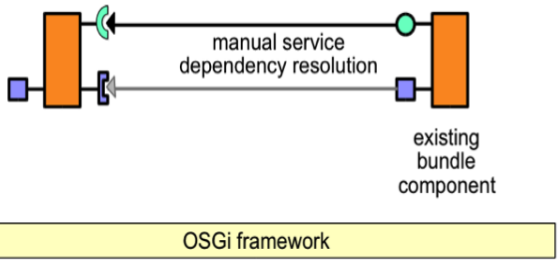
Step 3 - New bundle activation



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 rather easy 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.

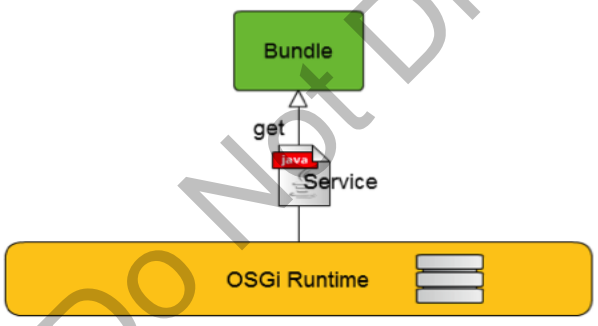
Declarative Services are a good alternative to

- 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



NOTE: A Service is by default only started if someone else uses it. The immediate flag forces a service start

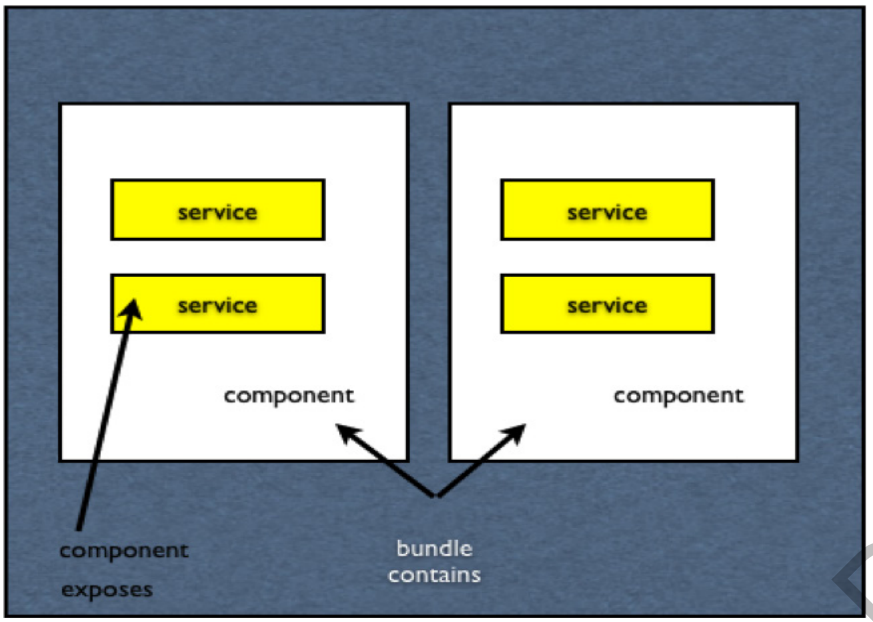
Components

NOTE:The OSGi Runtime calls the bundle and not the other way around

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.

NOTE: Do not confuse OSGi components with CQ 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".



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 service is:

- Component providing a service
- Component implementing one or more Java interfaces (= 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 .

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.

Bundles

Bundles can be uploaded into the Felix container through the Adobe CQ5 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.

NOTE: See section 112.4.3, Component Element, In the OSGi Service Platform Service Compendium Specification for more information.

```
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.felic.scr.annotations.Deactivate;
@Component
public class MyComponent {
      @Activcate
      protected void activate() {
        // do something
      }
      @Deactivate
      protected void deactivate() {
        // do something
      }
}
```

@Service

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

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

```
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 {
```

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.

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

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

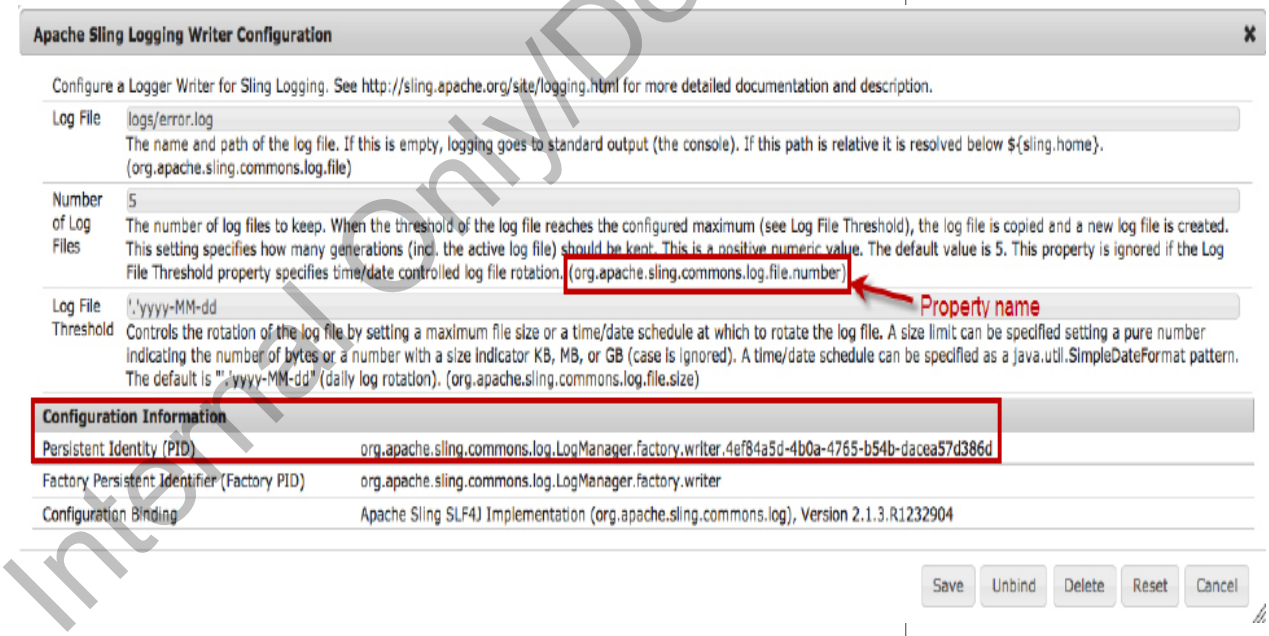
The **@Overide** annotation informs the compiler that the element is meant to override an element declared in a superclass.

@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.

EXERCISE - OSGi Component in CRXDE Lite

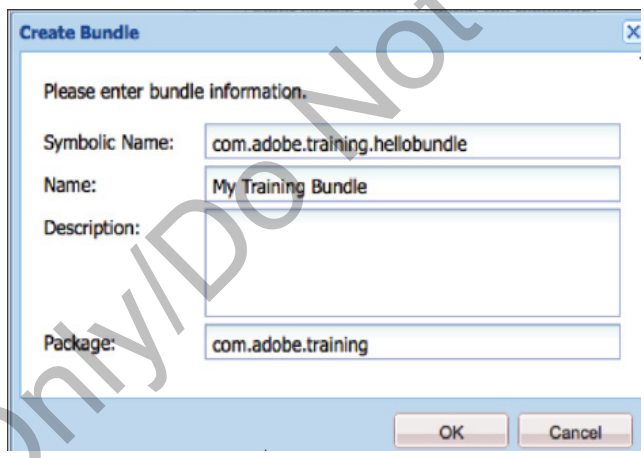
In this exercise we will

- 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

For this exercise, we will use CRXDE Lite to build our OSGi bundle, component, and service. Later, we will explore Maven as an OSGi build tool.

1. Start CQ5, go to http://localhost:4502/crx/de/

2. Right-**click** on /apps/geometrixx -> create bundle

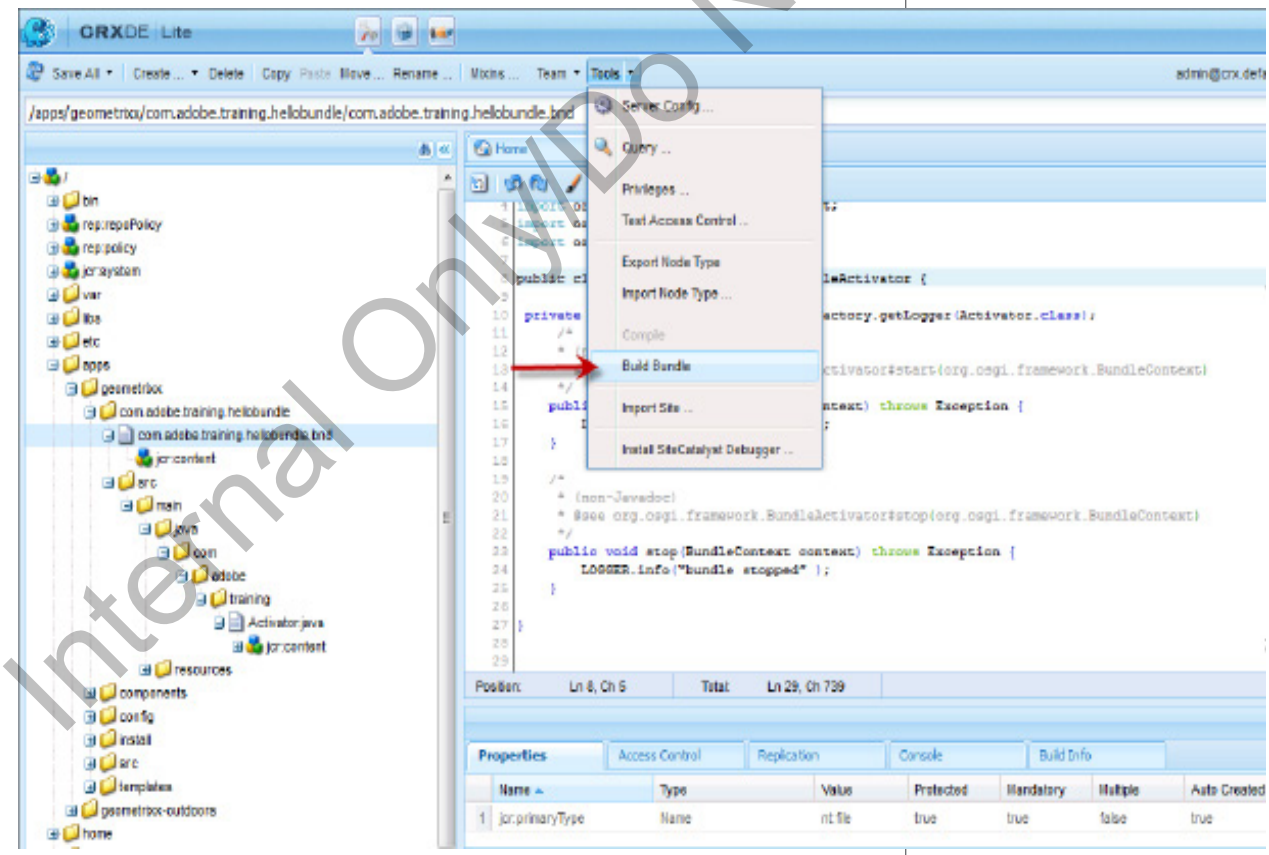


3. Add logging to the Activator. Make the following changes to the Activator.java file:

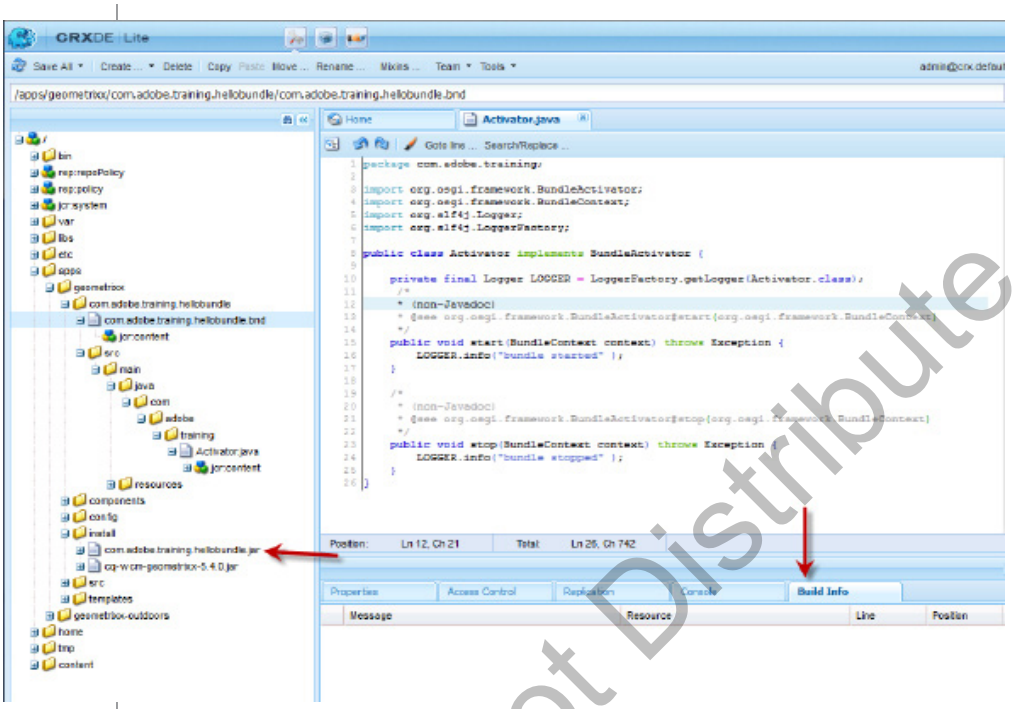
```
package com.adobe.training;

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

public class Activator implements BundleActivator {
  private 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" );
  }

}
```

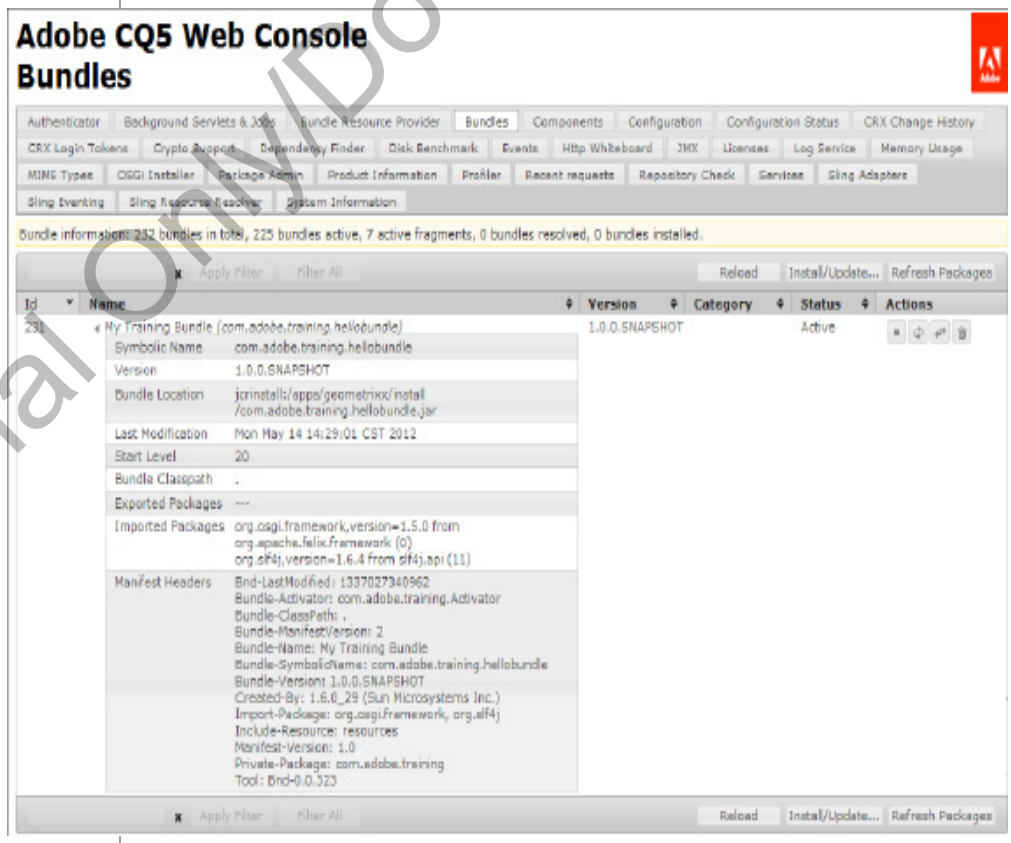
4. Build and deploy bundle. Select the .bnd file and then choose Build Bundle from the Tools menu in the top Toolbar.



Notice the jar file appearing in the install folder under /apps/geometrixx.



5. Go to http://localhost:4502/system/console/bundles and check that the bundle is deployed.

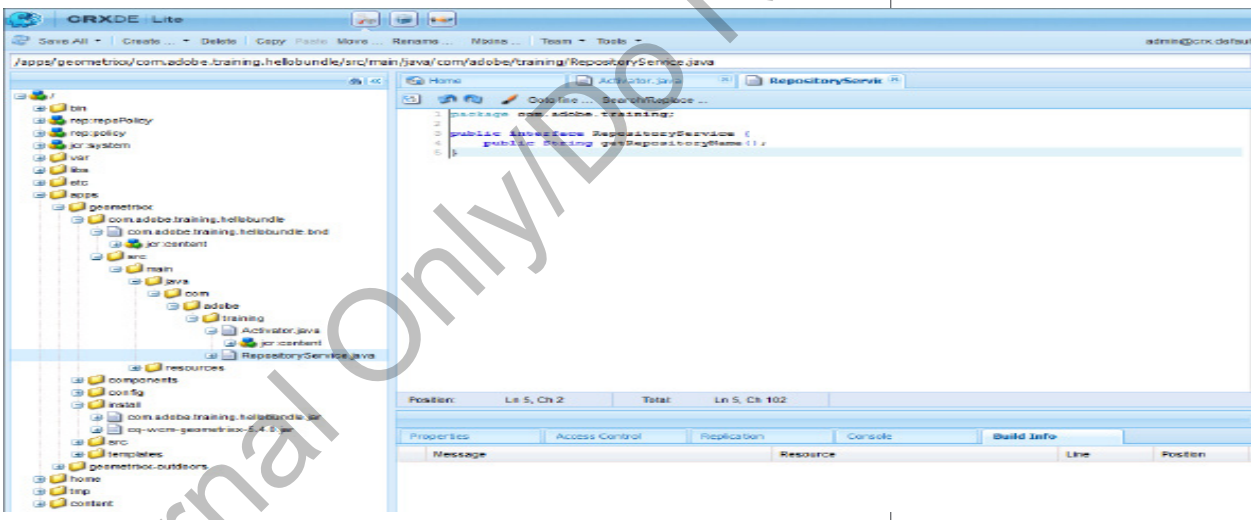


6. Start and stop this bundle. Check the error.log for the log message

```
14.05.2012 15:19:41.845 *INFO* [FelixDispatchQueue] org.apache.felix.framework FrameworkEvent PACKAGES REFRESHED
14.05.2012 15:19:46.403 *INFO* [JcrInstaller.2] org.apache.sling.installer.provider.jcr.impl.JcrInstaller Registering resource with OSGi installer: [InstallableResource, prior
14.05.2012 15:19:46.421 *INFO* [OsgiInstallerImpl] org.apache.sling.audit.osgi.installer Installed bundle com.adobe.training.hellobundle [235] from resource TaskResource(url=j
14.05.2012 15:19:46.423 *INFO* [OsgiInstallerImpl] com.adobe.training.Activator bundle started
14.05.2012 15:19:46.423 *INFO* [OsgiInstallerImpl] org.apache.sling.installer.core.impl.tasks.BundleStartTask Bundle started (bundle ID=235) : com.adobe.training.hellobundle
14.05.2012 15:19:46.551 *INFO* [FelixDispatchQueue] com.adobe.training.hellobundle BundleEvent INSTALLED
14.05.2012 15:19:46.551 *INFO* [FelixDispatchQueue] com.adobe.training.hellobundle BundleEvent RESOLVED
14.05.2012 15:19:46.552 *INFO* [FelixDispatchQueue] com.adobe.training.hellobundle BundleEvent STARTED
14.05.2012 15:19:57.999 *INFO* [0:0:0:0:0:0:0:1 [1337030397996] POST /system/console/bundles/235 HTTP/1.1] com.adobe.training.Activator bundle stopped
14.05.2012 15:19:58.000 *INFO* [FelixDispatchQueue] com.adobe.training.hellobundle BundleEvent STOPPED
14.05.2012 15:20:00.268 *INFO* [0:0:0:0:0:0:0:1 [1337030400264] POST /system/console/bundles/235 HTTP/1.1] com.adobe.training.Activator bundle started
14.05.2012 15:20:00.268 *INFO* [FelixDispatchQueue] com.adobe.training.hellobundle BundleEvent STARTED
14.05.2012 15:20:02.172 *INFO* [0:0:0:0:0:0:0:1 [1337030402169] POST /system/console/bundles/235 HTTP/1.1] com.adobe.training.Activator bundle stopped
14.05.2012 15:20:02.174 *INFO* [FelixDispatchQueue] com.adobe.training.hellobundle BundleEvent STOPPED
14.05.2012 15:20:03.541 *INFO* [0:0:0:0:0:0:0:1 [1337030403377] POST /system/console/bundles/235 HTTP/1.1] com.adobe.training.Activator bundle started
```

7. Create the service interface. Create a new java class file named ***RepositoryService.java*** and add the following code.

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



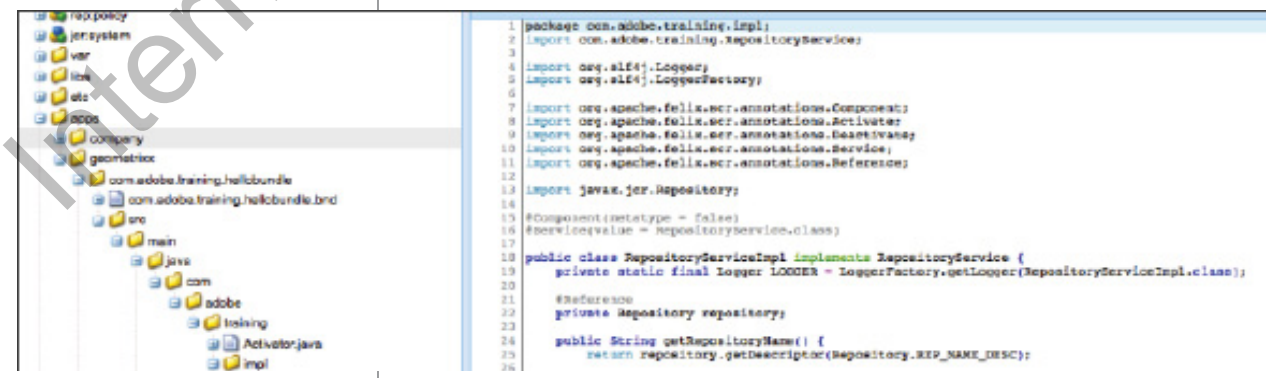
8. Implement the service interface. Create a folder named **impl** under the package directory.

/.../com.adobe.training.hellobundle/src/main/java/com/adobe/training/impl

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

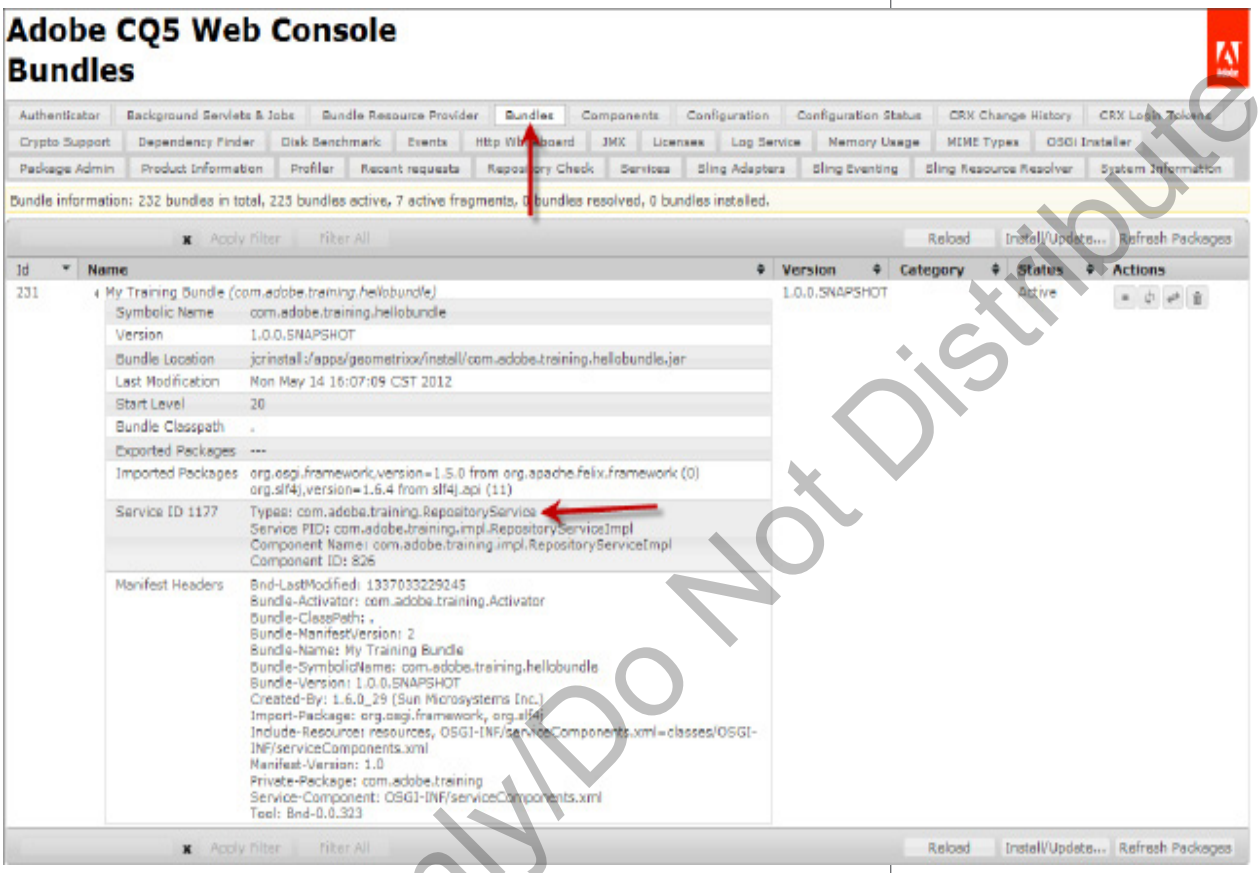
```
package com.adobe.training.impl;
import com.adobe.training.RepositoryService;
import org.slf4j.Logger;
```

```
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(metatype = false)
@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");
    }
}
```



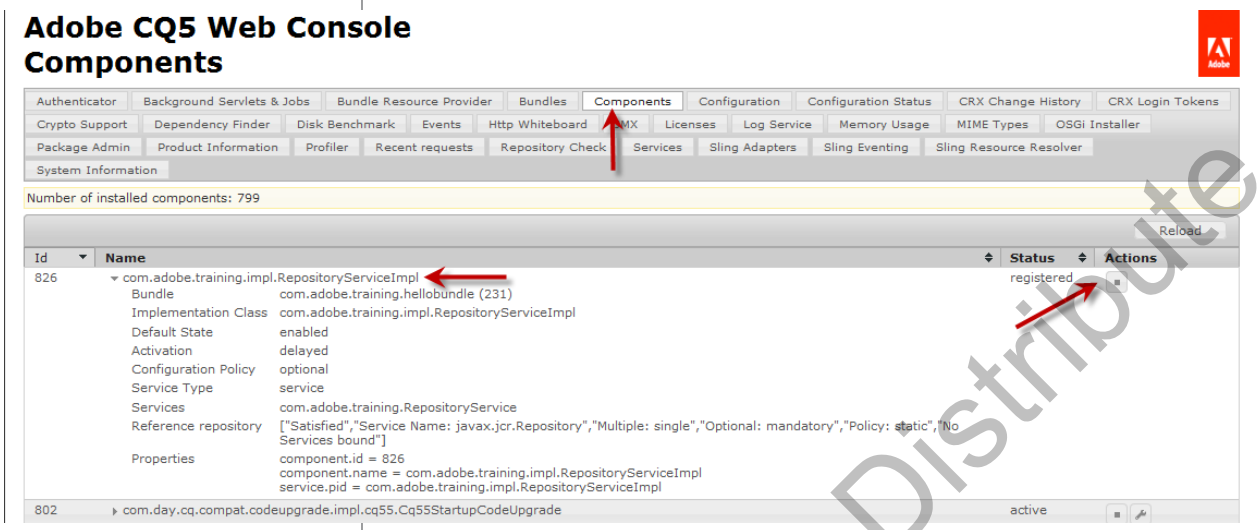
10. Notice the use of the @Component, @Service and @Reference annotations.

11. Deploy the bundle and inspect the Adobe CQ5 Web Console. Your modified bundle should now show the new service in bundle manifest.



NOTE: The bundle gets started, but the event is not yet activated, because you did not set the @Component(immediate=true) annotation. It will get activated when we actually use the service

The new component should show up in the Adobe CQ5 Web Console Components tab.



The service should show up in the Services tab.

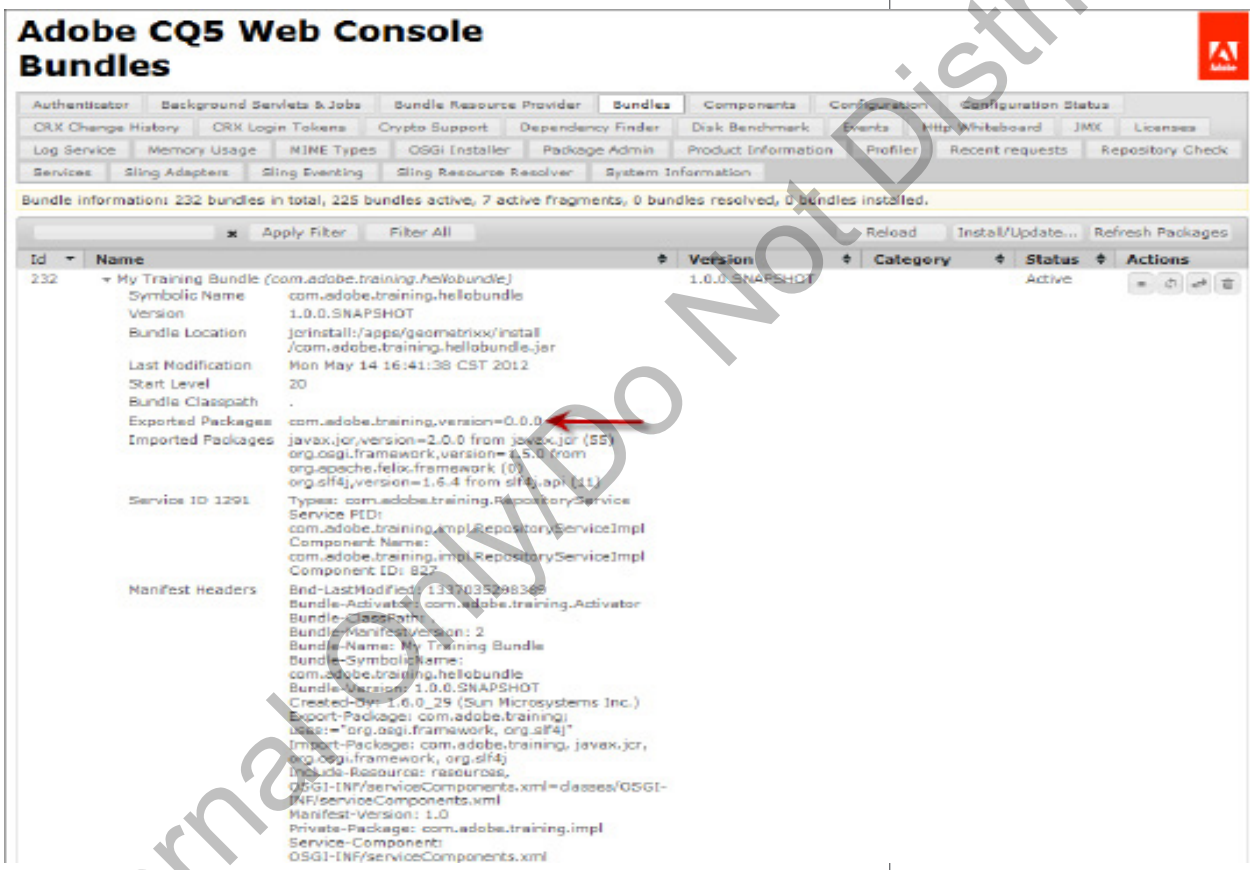
and see the error.log output:

```
14.05.2012 16:07:09.701 *INFO* [OsgiInstallerImpl] com.adobe.training.Activator bundle stopped
14.05.2012 16:07:09.745 *INFO* [OsgiInstallerImpl] org.apache.sling.audit.osgi.installer Updated bundle com.adobe.training.hellobundle [231]
14.05.2012 16:07:09.748 *INFO* [FelixDispatchQueue] com.adobe.training.hellobundle BundleEvent STOPPED
14.05.2012 16:07:09.748 *INFO* [FelixDispatchQueue] com.adobe.training.hellobundle BundleEvent UNRESOLVED
14.05.2012 16:07:09.748 *INFO* [FelixDispatchQueue] com.adobe.training.hellobundle BundleEvent UPDATED
14.05.2012 16:07:09.748 *INFO* [FelixDispatchQueue] org.apache.felix.framework FrameworkEvent PACKAGES REFRESHED
14.05.2012 16:07:09.753 *INFO* [FelixDispatchQueue] org.apache.felix.framework FrameworkEvent PACKAGES REFRESHED
14.05.2012 16:07:10.009 *INFO* [FelixDispatchQueue] com.adobe.training.hellobundle BundleEvent RESOLVED
14.05.2012 16:07:10.011 *INFO* [OsgiInstallerImpl] com.adobe.training.Activator bundle started
14.05.2012 16:07:10.017 *INFO* [OsgiInstallerImpl] com.adobe.training.hellobundle Service [com.adobe.training.impl.RepositoryServiceImpl,1177
14.05.2012 16:07:10.017 *INFO* [OsgiInstallerImpl] org.apache.sling.installer.core.impl.tasks.BundleStartTask Bundle started (bundle ID=231)
14.05.2012 16:07:10.018 *INFO* [FelixDispatchQueue] com.adobe.training.hellobundle BundleEvent STARTED
14.05.2012 16:17:23.051 *INFO* [pool-5-thread-4] com.day.crx.security.token.impl.TokenCleanupTask TokenCleanupTask: Removed 0 token(s) in 50
```

12. You may not want the service to be publicly available, so make the implementation classes private by editing the *com.adobe.training.hellobundle.bnd* file.

```
Export-Package: com.adobe.training
Import-Package: *
Private-Package: com.adobe.training.impl
# Include-Resource:
Bundle-Name: My Training Bundle
Bundle-Description:
Bundle-SymbolicName: com.adobe.training.hellobundle
Bundle-Version: 1.0.0-SNAPSHOT
Bundle-Activator: com.adobe.training.Activator
```

13. Redeploy the bundle.

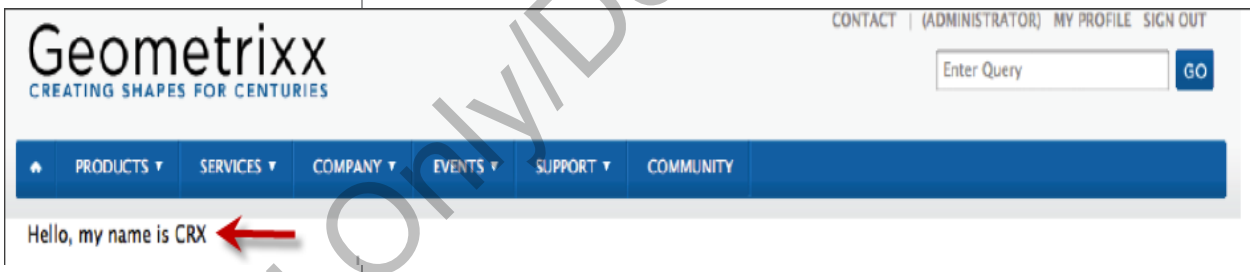


14. Consume the service in a JSP by editing content.jsp in the geometrixx homepage component.

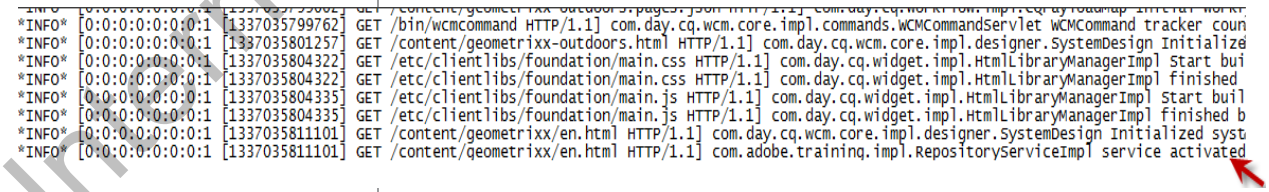
```
<%@include file="/libs/foundation/global.jsp" %>
<div class="container _ 16">
    <%
        com.adobe.training.RepositoryService repositoryService
=
        sling.getService(com.adobe.training.RepositoryService.
class);
```

```
    %>

    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>
```

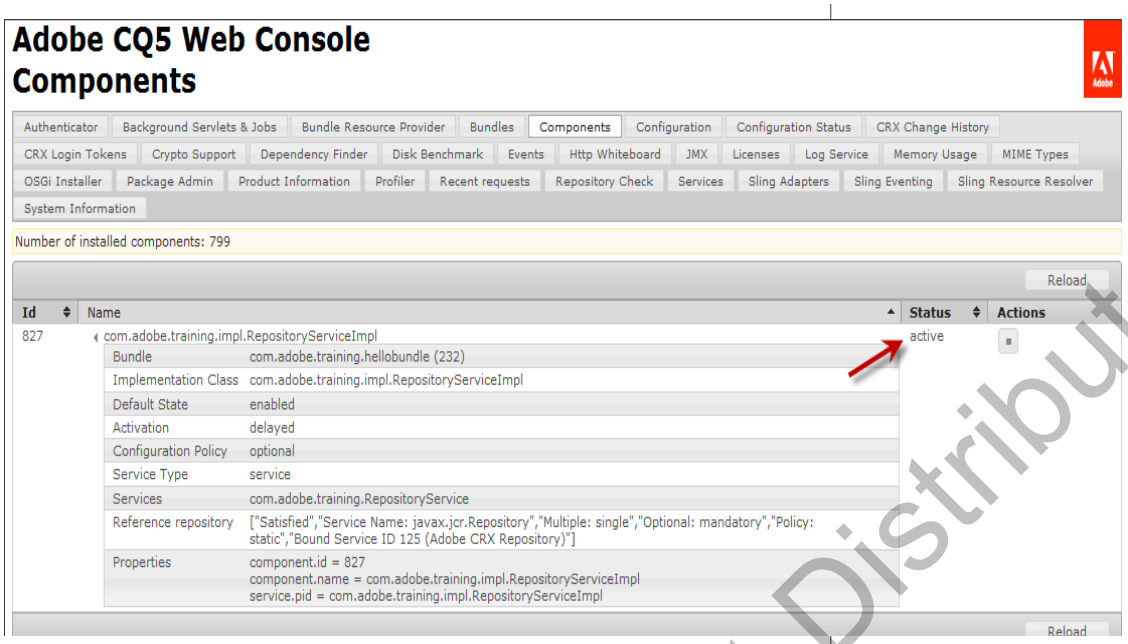
15. You should see the following on the Geometrixxx home page:



16. By now you should see in the error.log that the service is active:



17. Also in the Adobe CQ5 Web console you now see that the component is active:



Congratulations! You have used CRXDE Lite to create an OSGi service. Now you have a working knowledge of the annotations. Next we will explore using Maven as a build tool.

Maven-scr-plugin

We used CRXDE Lite to build our bundle, which has included the Felix annotations component in its library at:

/etc/crxde/profiles/default/libs/org.apache.felix.scr.annotations-1.6.0.jar

The Apache Felix Maven SCR Plugin is a great tool to ease the development of OSGi components and services when you are using Maven as a build tool. If you want to use the annotations with an IDE other than CRXDE and with Maven as a build tool, you must use the *maven-scr-plugin* and add the appropriate dependency to your POM.

```
<dependency>
  <groupId>org.apache.felix</groupId>
  <artifactId>org.apache.felix.scr.annotations</artifactId>
  <version>1.6.0</version>
</dependency>
```

Support for automatic generation of the component and metadata descriptors is embedded in the *org.apache.felix:maven-scr-plugin*. The *maven-scr-plugin* creates the necessary descriptors, from the annotations, for the OSGi Declarative Services, Config Admin and Metatype services.

2 Development Models for Teams

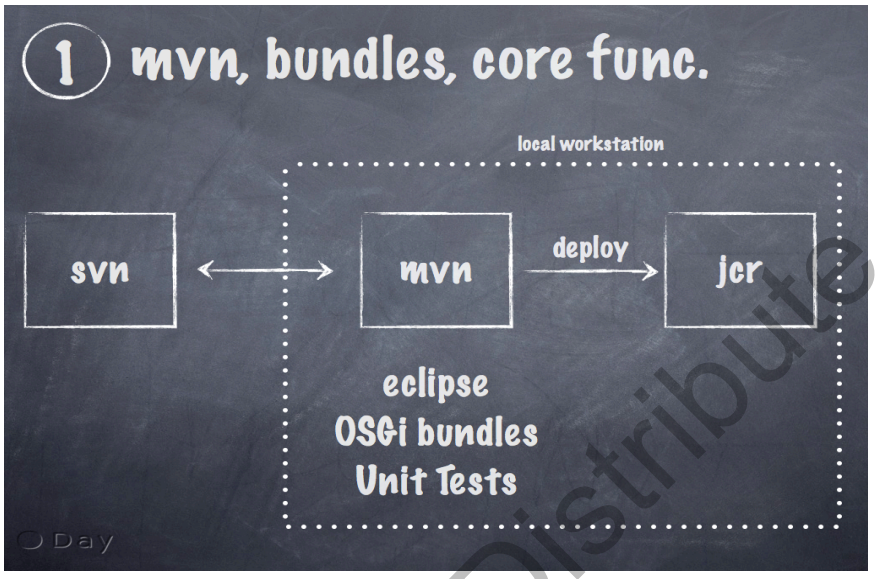
Team Development Models

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

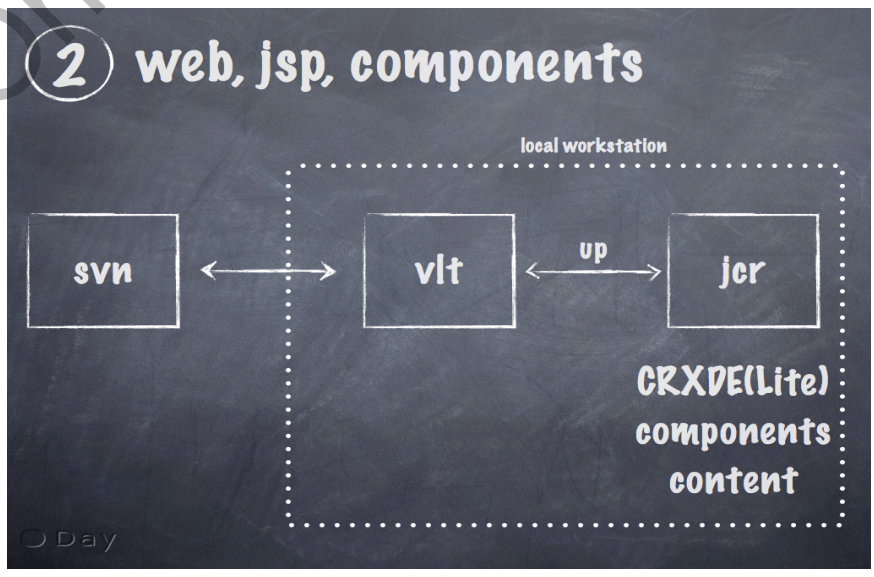
- Inspect the anatomy of the project
- Compile the project
- Deploy into a local CQ instance
- Use vlt to change locally and in the repository

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/mvnref-book/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. 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.



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. ${project.groupId} would give the groupId value,`?com`.adobe in the earlier simple `pom.xml` example.
- ***settings*** - exposes Maven settings information. For example, `${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.

EXERCISE - Use Maven to Create a Bundle

Before installing Maven, check that you already have java 1.6 installed - at a command prompt type `java -version` to display the version that is installed. If you do not see java version 1.6, 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

Installing Maven - Instructions for Mac

1. Uncompress the file apache-maven-3.0.0-bin.tar.gz into the /usr/localfolder. 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`

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_login 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\apache-maven-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\CQ54AdvDev\
   and unzip sampleproject.zip, (on the USB stick in the Exercises folder) saving the files in the newly created directory.

   On MAC, create a new folder for the project called /Adobe/CQ54AdvDev and unzip, sampleproject.zip (on the usb stick in the Exercises folder) saving the files in the newly created folder

   It contains the skeleton for a project named "company" with 2 Maven modules: 1 OSGi bundle company-core and 1 CRX package company-ui (that will contain JSPs, content nodes etc)

2. Inspect pom.xml. 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>2.3.4</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>
```

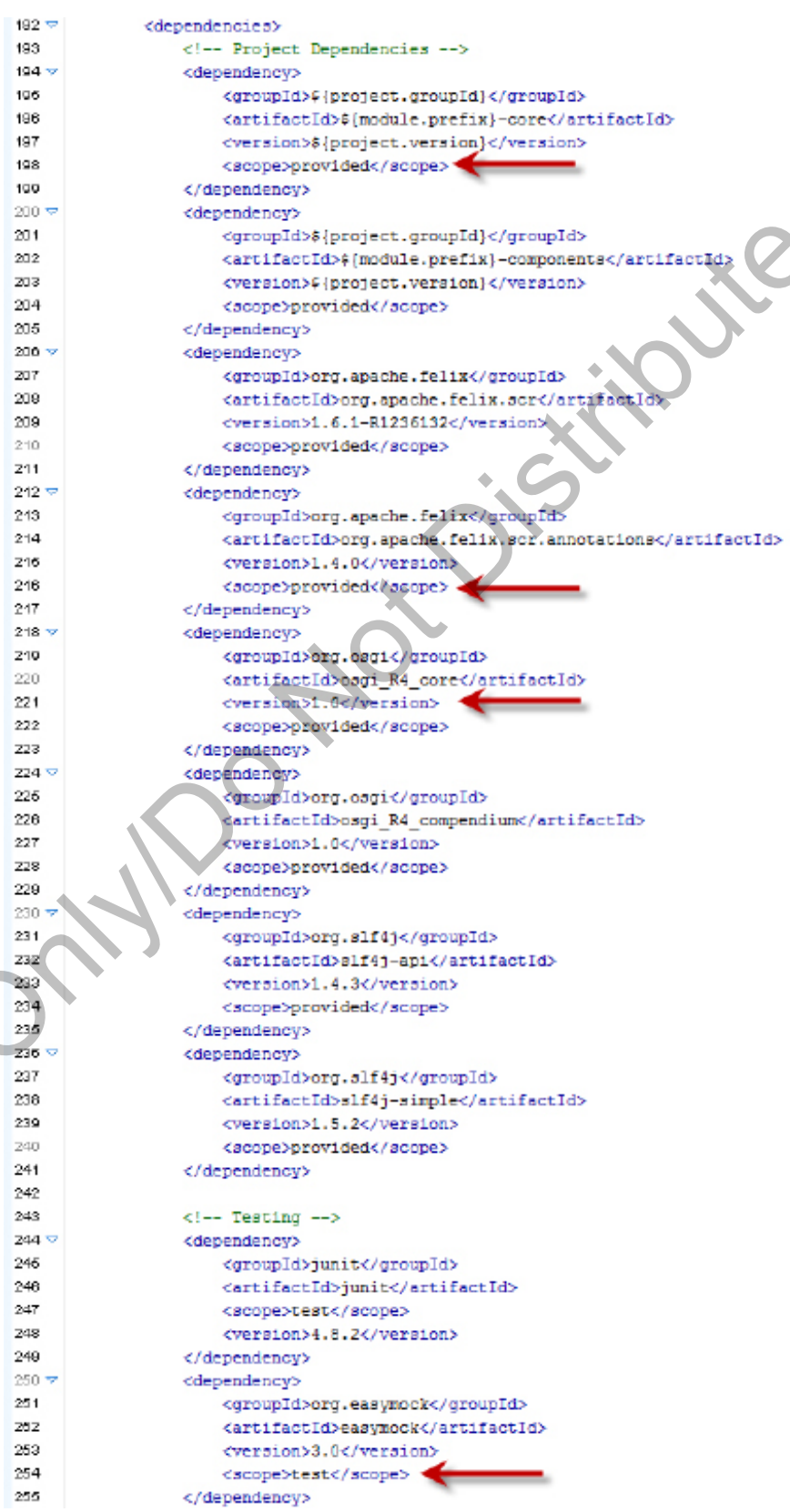
Take a look at these and other plugins in the pom.xml file:

```
<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:



```
192        <dependencies>
193            <!-- Project Dependencies -->
194            <dependency>
195                <groupId>${project.groupId}</groupId>
196                <artifactId>${module.prefix}-core</artifactId>
197                <version>${project.version}</version>
198                <scope>provided</scope>
199            </dependency>
200            <dependency>
201                <groupId>${project.groupId}</groupId>
202                <artifactId>${module.prefix}-components</artifactId>
203                <version>${project.version}</version>
204                <scope>provided</scope>
205            </dependency>
206            <dependency>
207                <groupId>org.apache.felix</groupId>
208                <artifactId>org.apache.felix.scr</artifactId>
209                <version>1.6.1-R1236132</version>
210                <scope>provided</scope>
211            </dependency>
212            <dependency>
213                <groupId>org.apache.felix</groupId>
214                <artifactId>org.apache.felix.scr.annotations</artifactId>
216                <version>1.4.0</version>
216                <scope>provided</scope>
217            </dependency>
218            <dependency>
219                <groupId>org.osgi</groupId>
220                <artifactId>osgi_R4_core</artifactId>
221                <version>1.0</version>
222                <scope>provided</scope>
223            </dependency>
224            <dependency>
225                <groupId>org.osgi</groupId>
226                <artifactId>osgi_R4_compendium</artifactId>
227                <version>1.0</version>
228                <scope>provided</scope>
229            </dependency>
230            <dependency>
231                <groupId>org.slf4j</groupId>
232                <artifactId>slf4j-api</artifactId>
233                <version>1.4.3</version>
234                <scope>provided</scope>
235            </dependency>
236            <dependency>
237                <groupId>org.slf4j</groupId>
238                <artifactId>slf4j-simple</artifactId>
239                <version>1.5.2</version>
240                <scope>provided</scope>
241            </dependency>
242
243            <!-- Testing -->
244            <dependency>
246                <groupId>junit</groupId>
246                <artifactId>junit</artifactId>
247                <scope>test</scope>
248                <version>4.8.2</version>
249            </dependency>
250            <dependency>
251                <groupId>org.easymock</groupId>
252                <artifactId>easymock</artifactId>
253                <version>3.0</version>
254                <scope>test</scope>
255            </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:

3. Now compile the project. Open a command prompt window and change to the directory containing the pom.xml file, i.e.

```
C:\Adobe\CQ54AdvDev\sampleproject\sampleproject on
Windows
/Adobe/CQ54AdvDev/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.

4 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:

and in the <pluginRepositories> section:
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.

5. Compile the project and deploy to CQ5; on the command line run

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

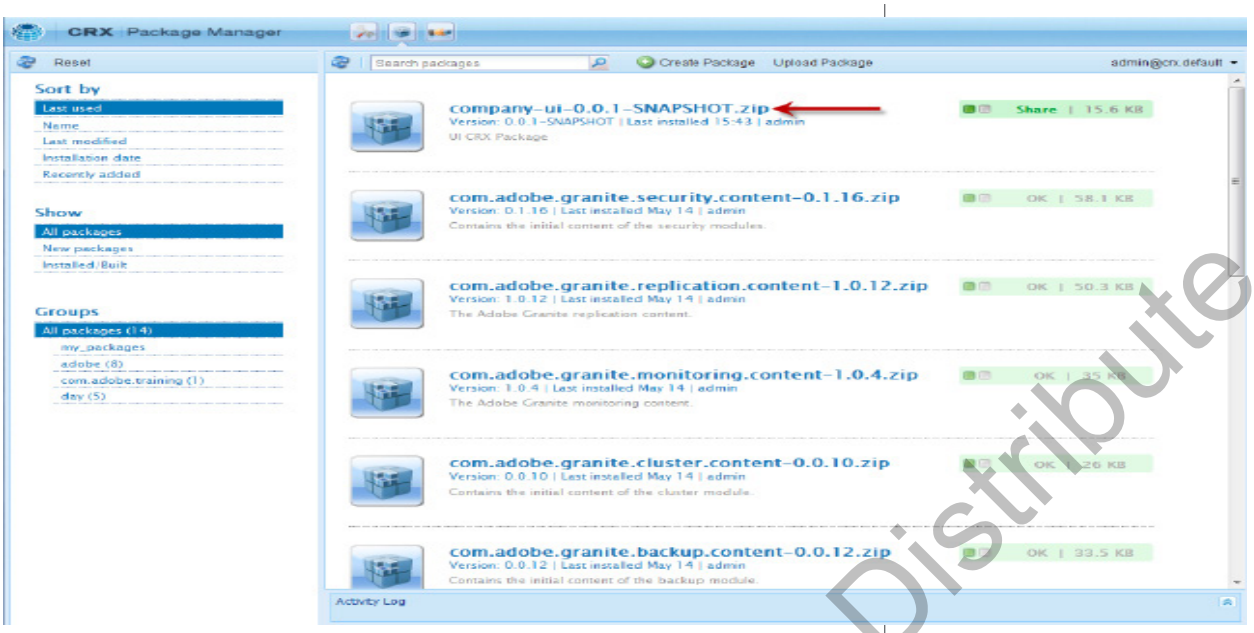
The -P option is for activating build profiles, so here we are using a profile called full. This builds the OSGi bundle and creates a CRX Package that gets uploaded to the local CRX instance. Find the package in

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

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

NOTE

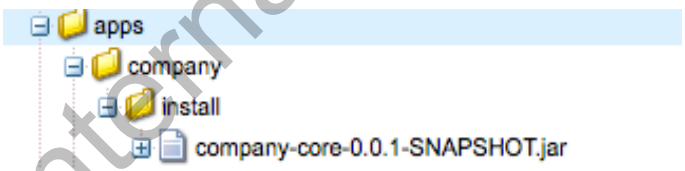
You will get the following Build Error: Failed to resolve artifact. It is because we have not defined our Adobe's public mvn repository yet.



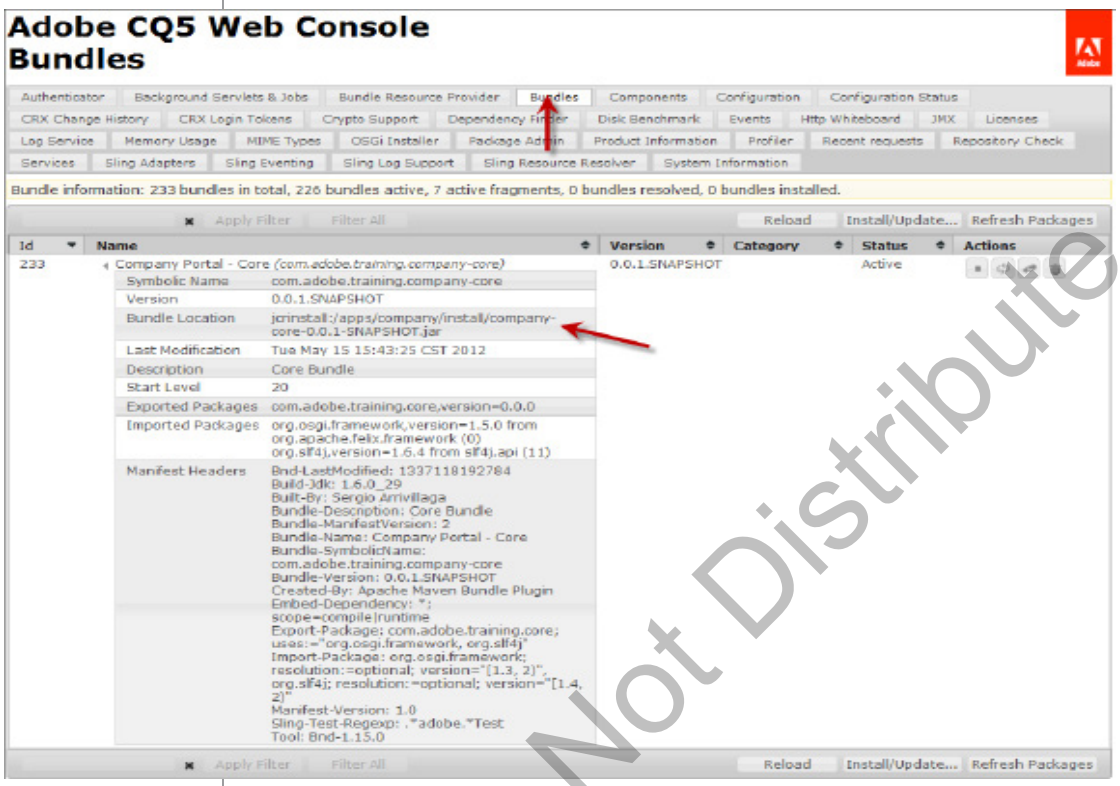

```
<repository>
      <id>central</id>
      <name>Adobe Proxy Repository</name>
      <url>http://repo.adobe.com/nexus/content/groups/public</url>
</repository>
```

In CRXDE Lite you should see

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



This bundle is now installed, check in Adobe CQ5 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 - 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 - 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**

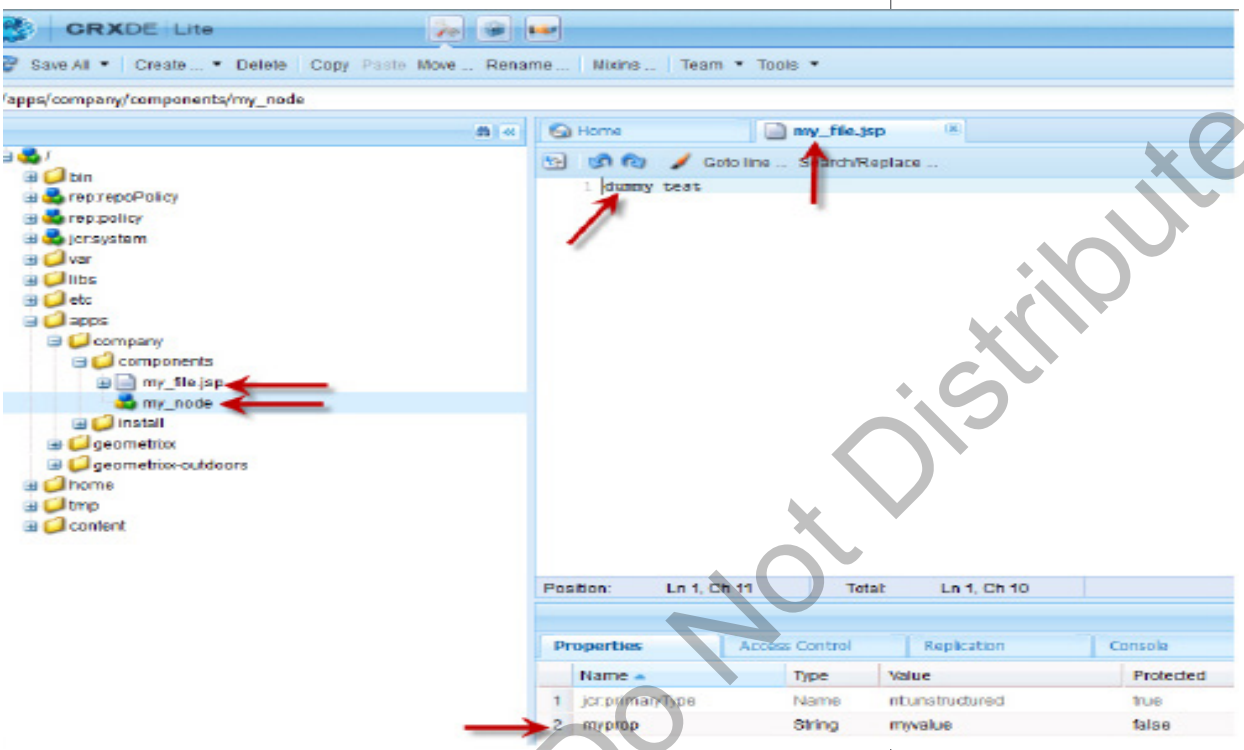
This will perform a vlt checkout of the JCR repository to your local filesystem. You will now have file serializations of the JCR content, e.g. in `company-ui/src/main/content/jcr_root/.content.xml`

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 (node type sling:Folder) and add an nt:unstructured node with a property, and a file.

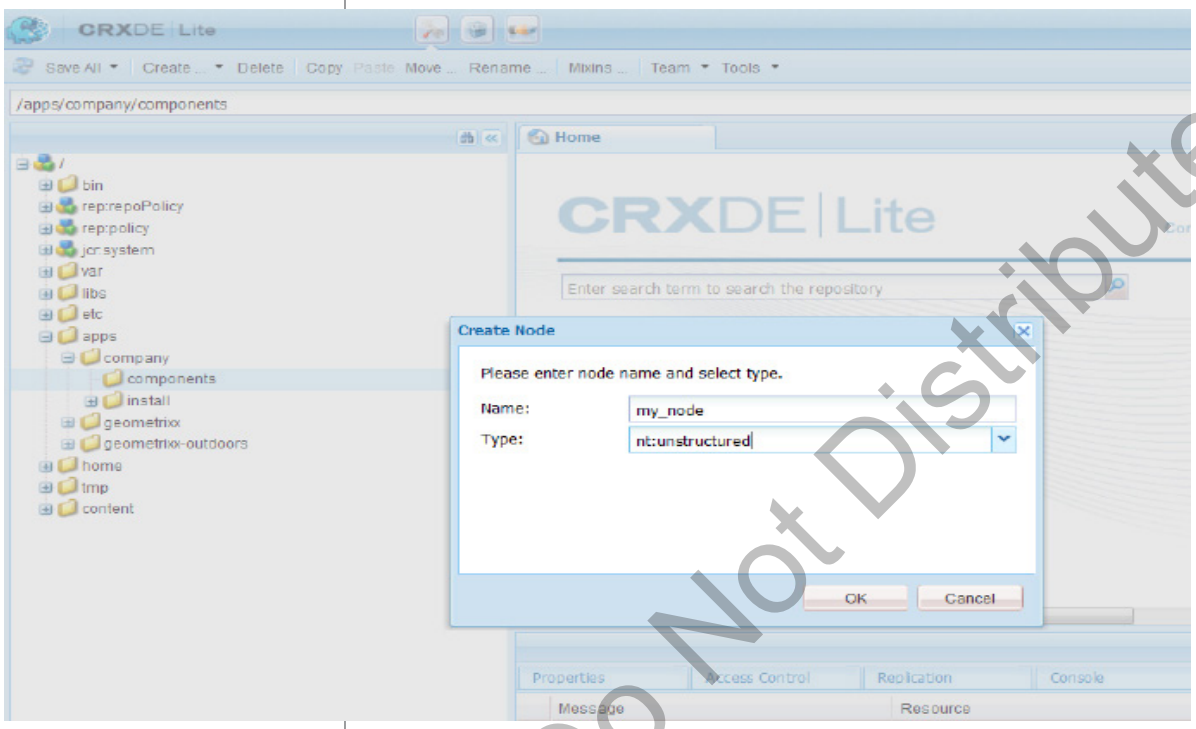


On the command line cd to company-ui/src/main/content/jcr_root and execute

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/src/main/content/jcr_root/company-ui/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 - 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.

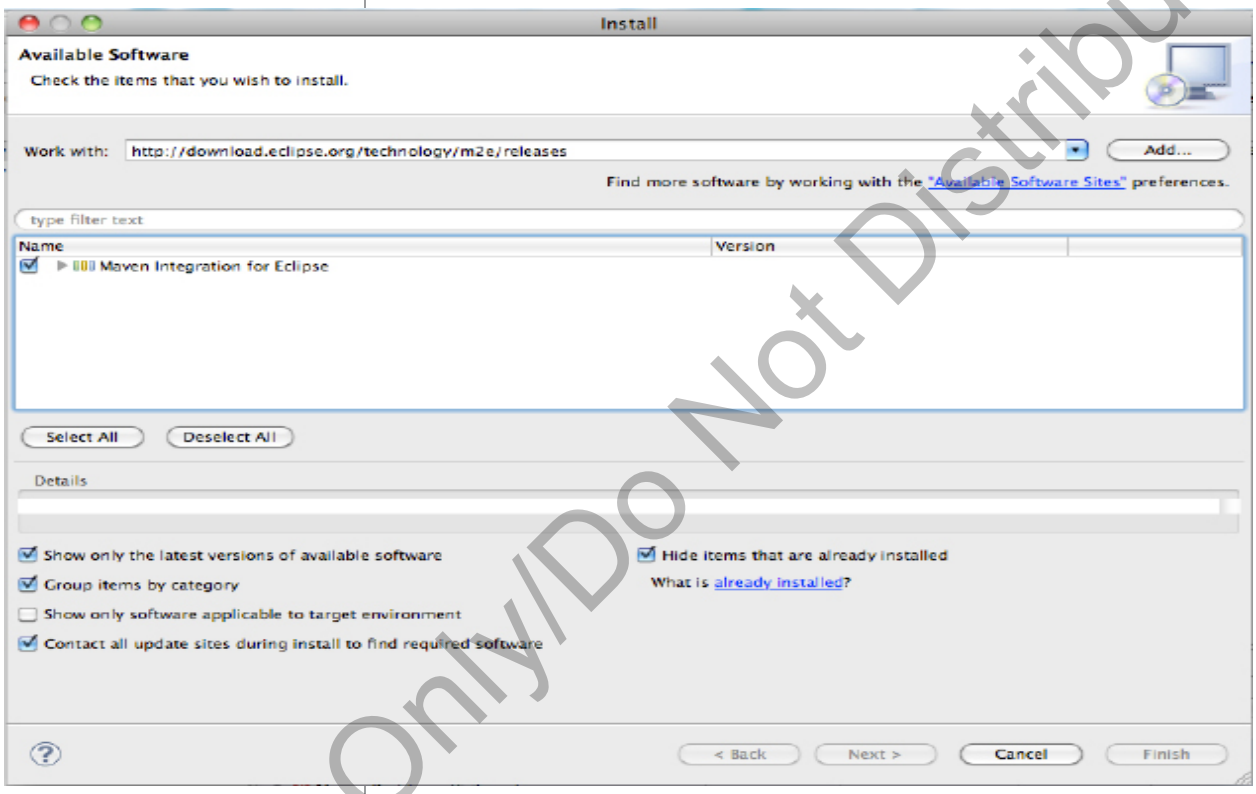
2. Install Eclipse; extract from the downloaded zip file to your destination directory (usually under Program Files on Windows, or Applications on the Mac).

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.

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

4. After installing Eclipse you need to install the M2e plugin via Eclipse installation at http://eclipse.org/m2e/ Follow the steps below, and note that m2e is NOT THE SAME as the Maven eclipse plugin! If you do web searches be aware of this distinction when filtering through search results.

   - In eclipse, navigate to Help - `Install New Software...` and in the Work with: dialog box, enter the url http://download.eclipse.org/technology/m2e/releases Press `Enter`, then check the box to the left of the new entry "`Maven Integration for Eclipse`". Check that the dialog looks like this:



   - Click on Next > then Finish to install the m2e plugin.

5. Generate Eclipse project files.

from the project root execute:

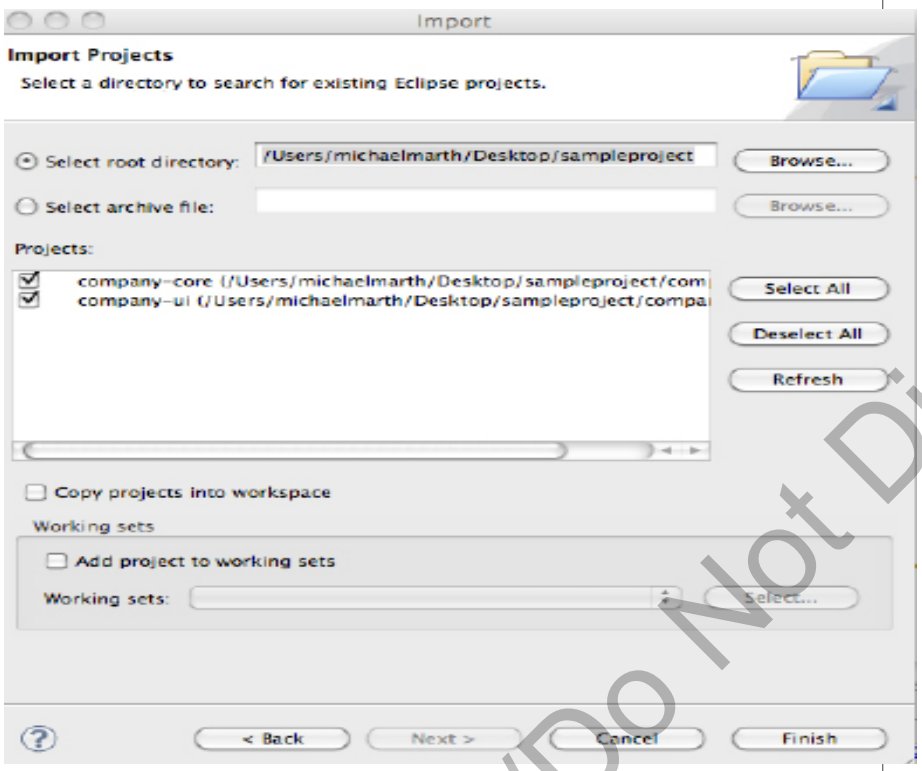
```
mvn eclipse:eclipse -DdownloadSources=true
        -DdownloadJavadocs=true
```

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

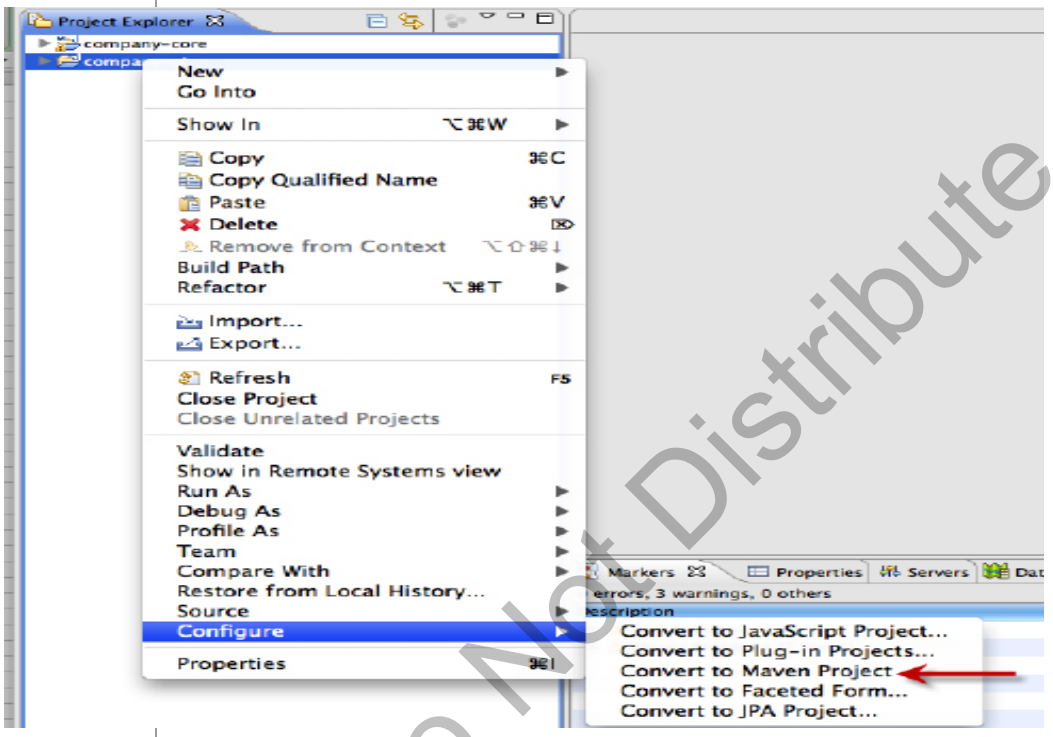
`sampleproject/company-core/.project`
`sampleproject/company-ui/.project`

Import these into Eclipse using the menu option:

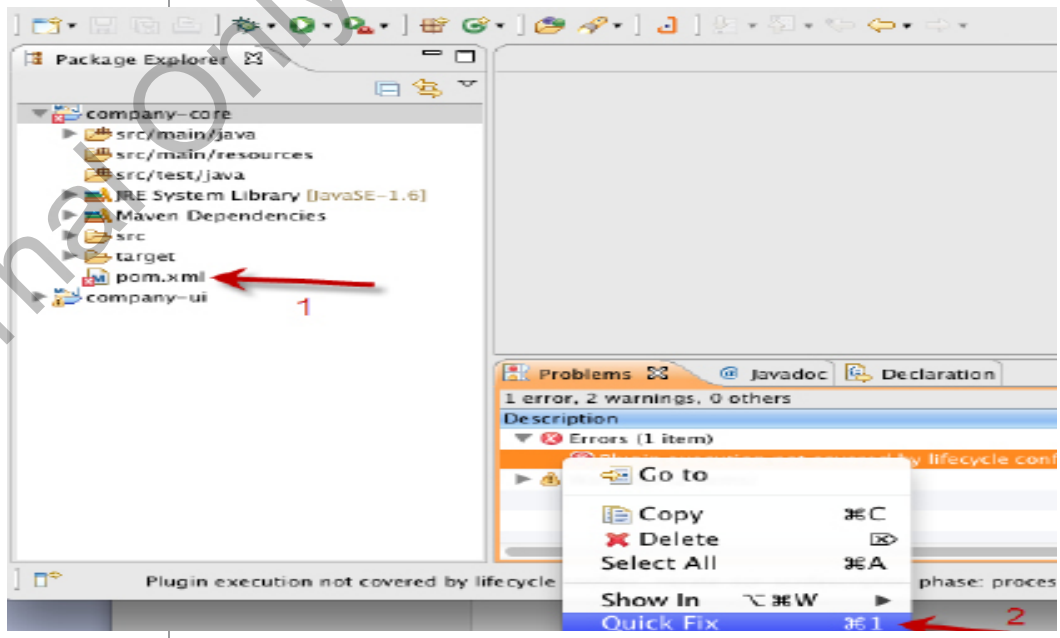
```
File > Import > General > Existing Projects Into Workspace
```



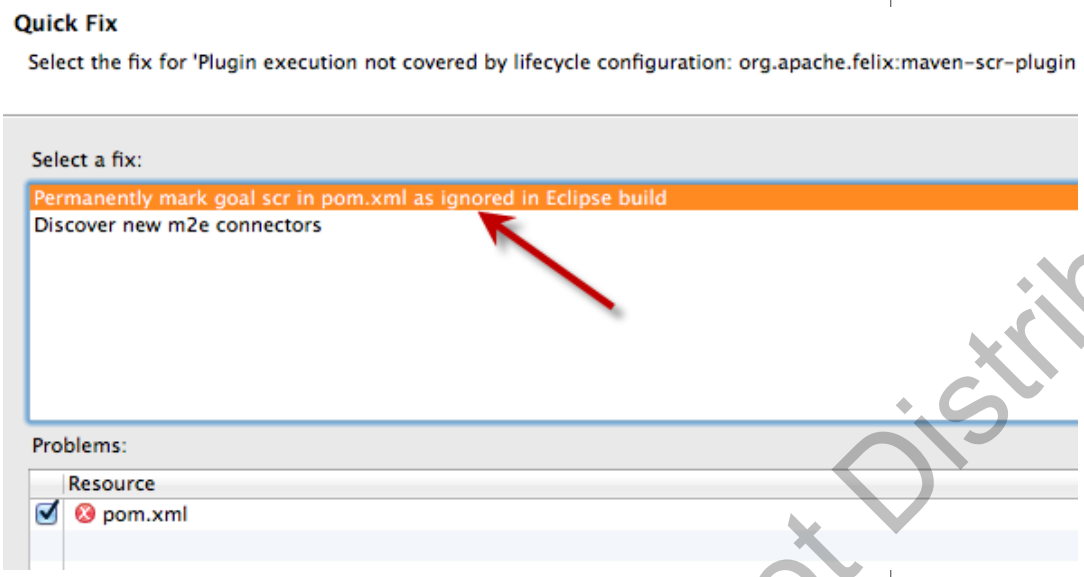
6. Right-clik on both projects (one by one) and select Configure > Convert to Maven Project



7. Right click on the error shown in the Problems panel 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

3 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 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

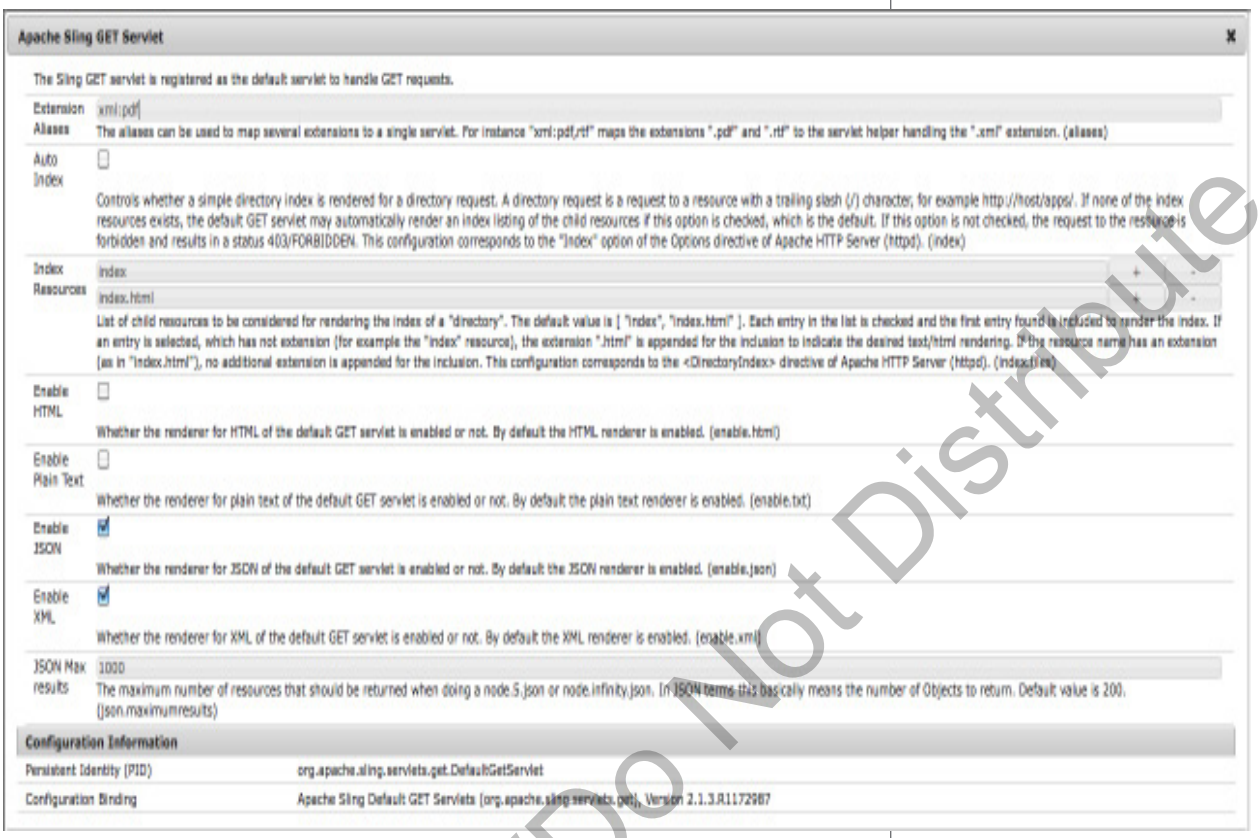
Apache Sling is a web framework that uses a Java Content Repository, such as Apache Jackrabbit or Adobe CRX to store and manage content. Sling applications use either scripts or Java servlets, selected based on simple name conventions, to process HTTP requests in a RESTful way

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 CQ5 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.

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

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

```
curl -F"jcr:primaryType=nt:unstructured" -Ftitle="some title
text" -Ftext="some body text content" http://host/some/new/
content
curl -F":operation=delete" http://host/content/sample
```

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

You can configure the Apache Sling POST Servlet using the Adobe CQ5 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

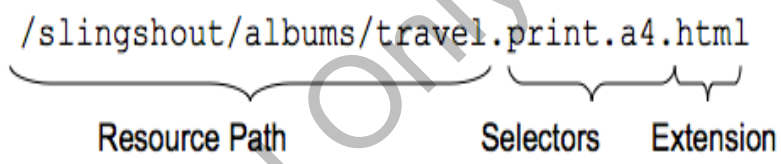
Resource-First Request Processing

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 CQ5, 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 use 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

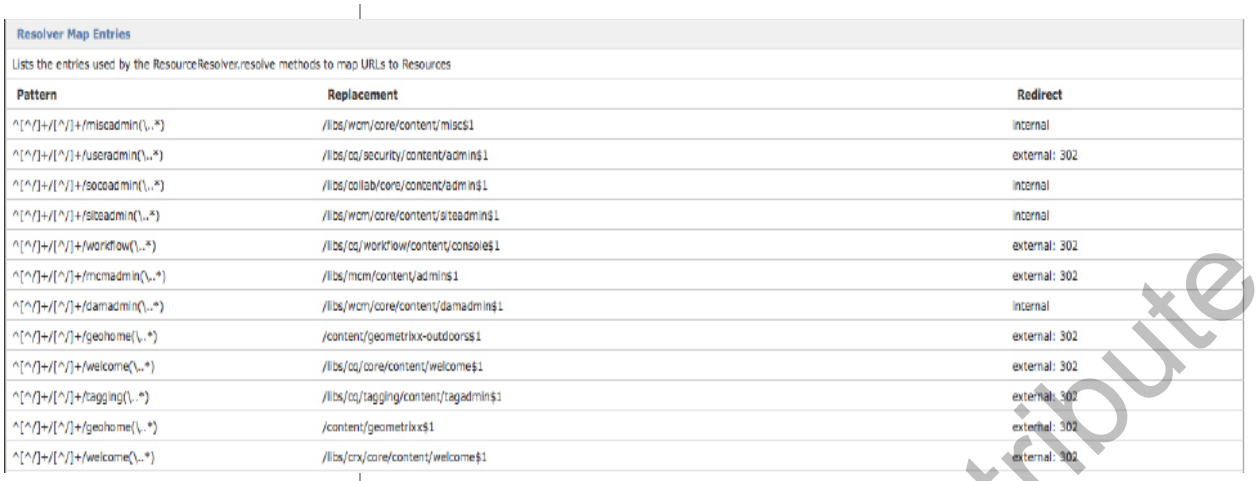
The Sling Resource Resolver is the gateway for finding/accessing (resolving) resources. The Resource Resolver

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

The Resource Resolver is configurable, through the CQ5 Web Console and also through standard OSGi configuration mechanisms in the CRX.

In CQ5, 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/admins1	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 - 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

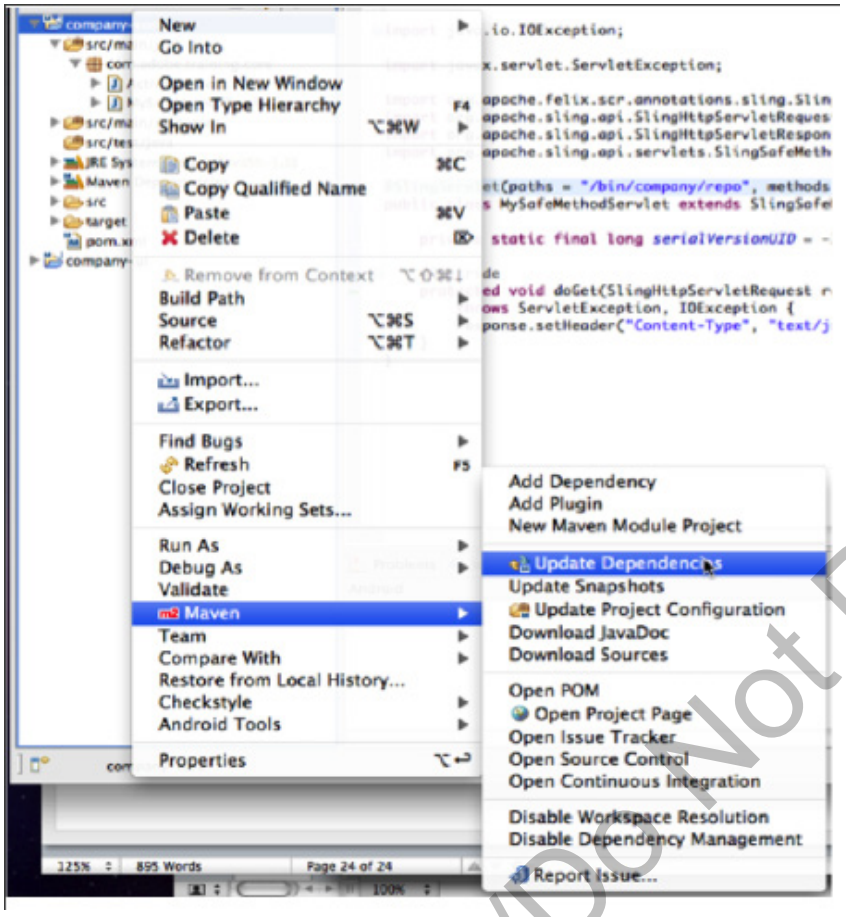
1. Add the necessary dependencies to the parent **pom.xml**
   In **pom.xml** , in the <dependency management> section, just before the <!-- Testing --> comment, add:

```
        </dependency>
    <dependency>
            <groupId>org.apache.sling</groupId>
            <artifactId>org.apache.sling.api</artifactId>
            <version>2.2.4</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>
    <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

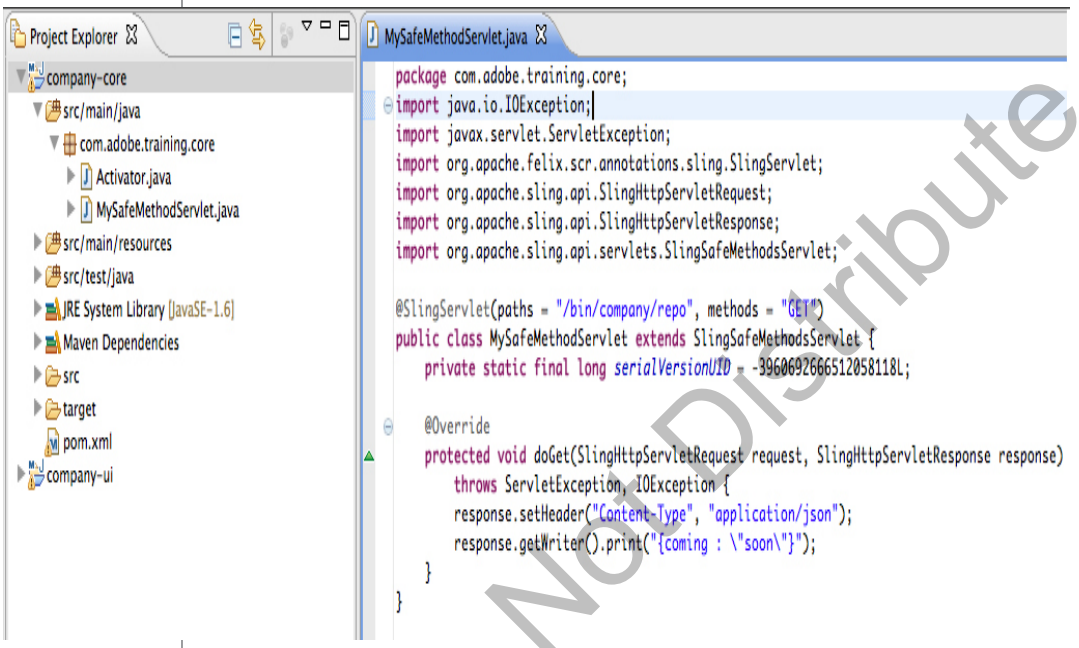


Alternatively you could have executed on the command line

```
$mvn eclipse:clean
$mvn eclipse:eclipse
```

and refreshed the project in Eclipse.

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”);
```

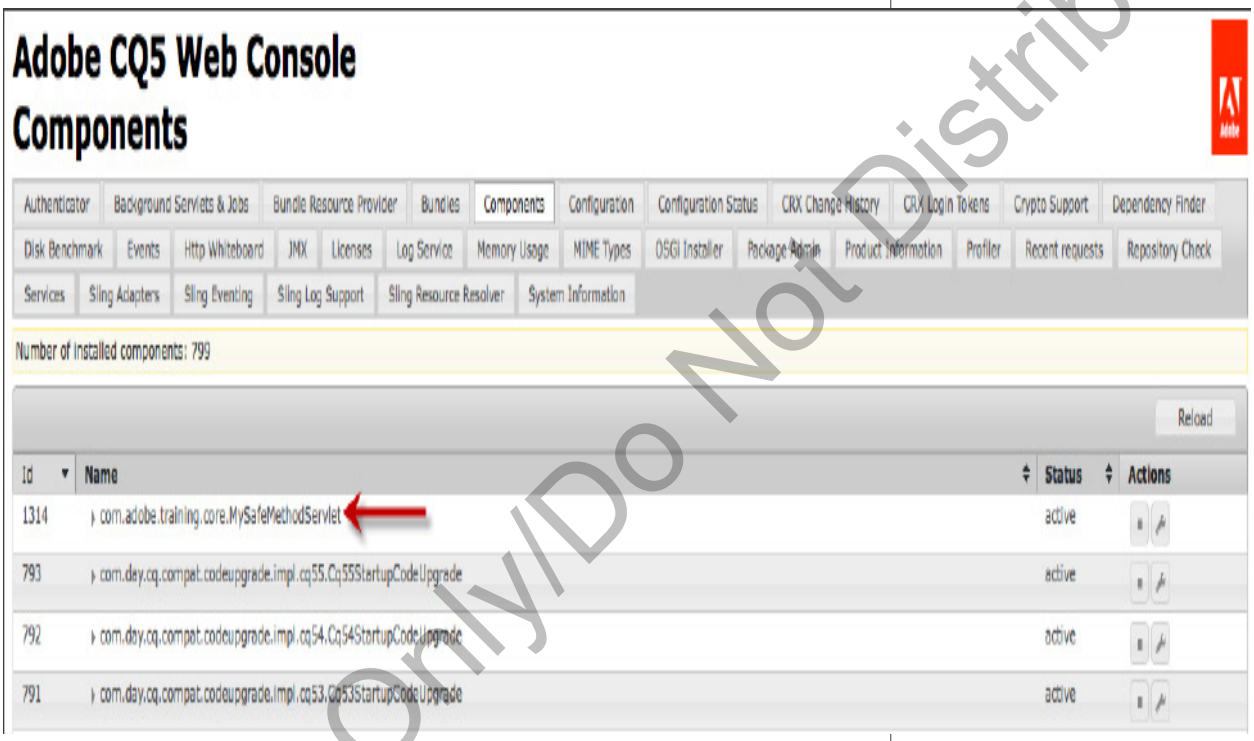
NOTE: This mVn command will not work if you are running the parent pom.xml.

```
    response.getWriter().print("{\"coming\" : \"soon\"}");
}
        }
```

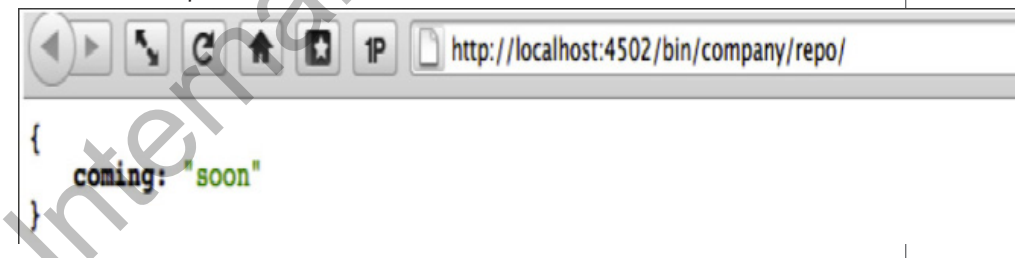
3. Deploy the code to your local CQ instance by executing in company-core directory:

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

Check that the component is installed:



and works as expected:



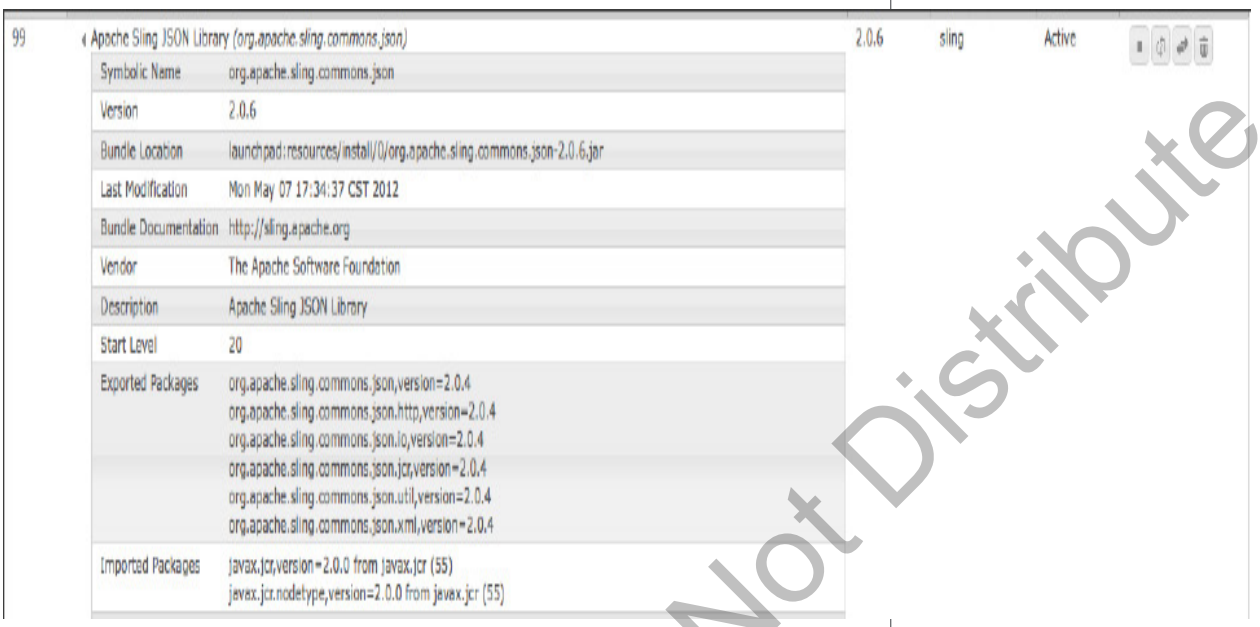
Note:
Maven options:
clean = remove class files-P bundle = create a bundle, i.e. the OSGi bundle class.
-P full = create a content package in CRX

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 CQ5 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.xmll



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
```

Notice the Exported Packages section. These are the relevant version numbers.

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

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

8. 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());
    }
}
```

9. Examine the results:

4 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. That is, the listener regularly polls the registry for any events it is listening for.

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 cq 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.

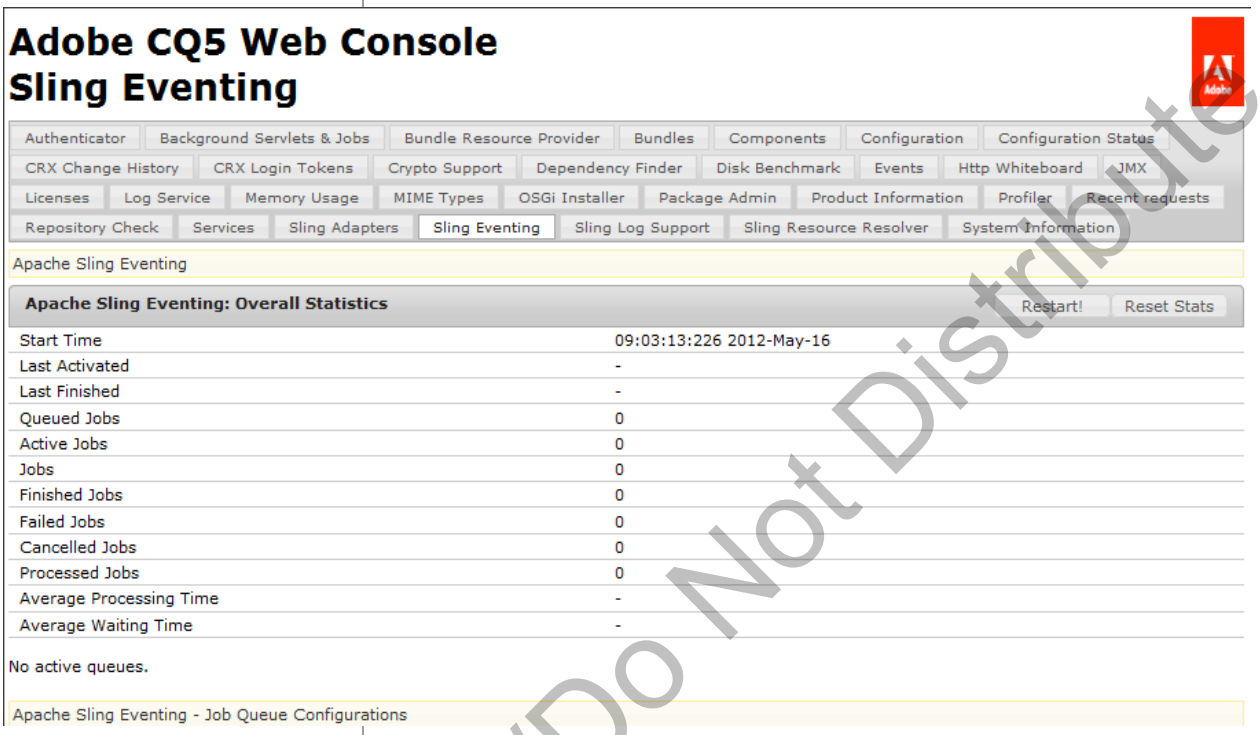
Sending Job Events

To send a job event the service needs to implement:

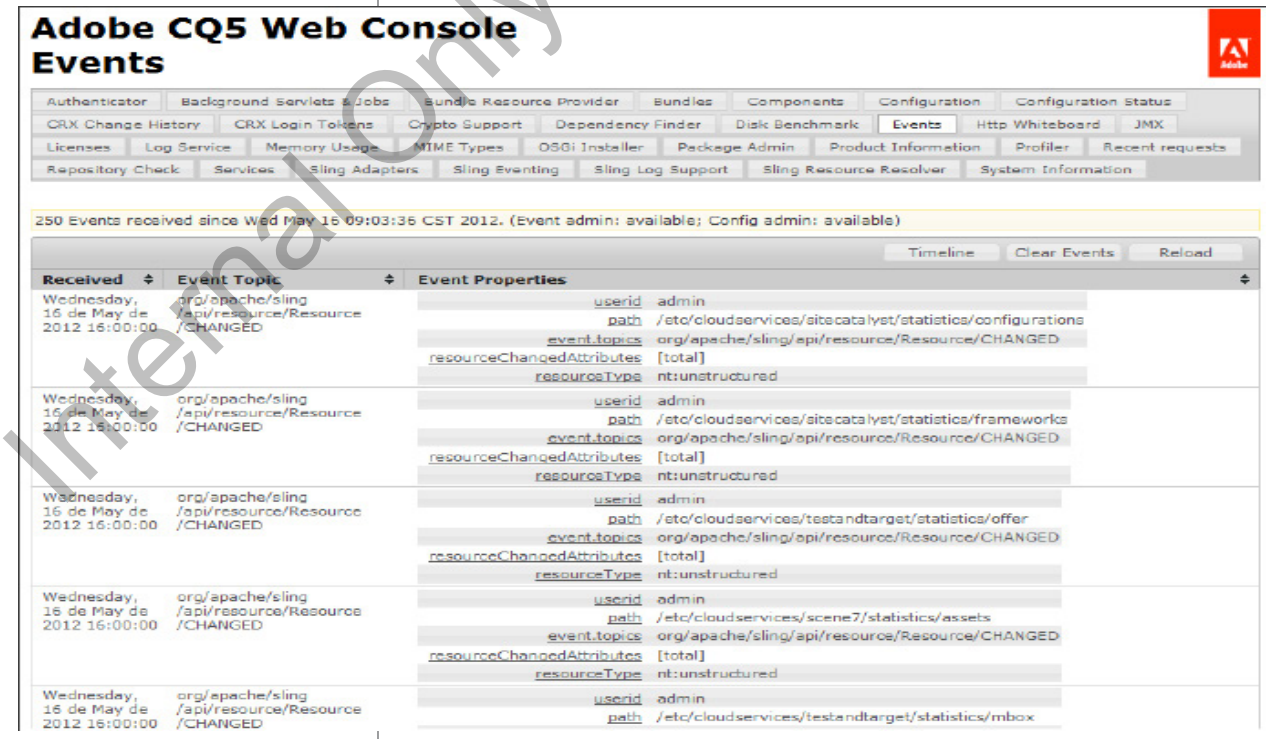
- the `org.osgi.service.event.EventHandler` interface.
- the `org.apache.sling.event.JobProcessor` interface.

Admin User Interface

Two tabs of the Adobe CQ5 Web Console contain information relating to events which may be helpful for debugging. Event statistics are shown under "Sling Eventing":



...and details of individual Events under "Events":



EXERCISE - Event Handling

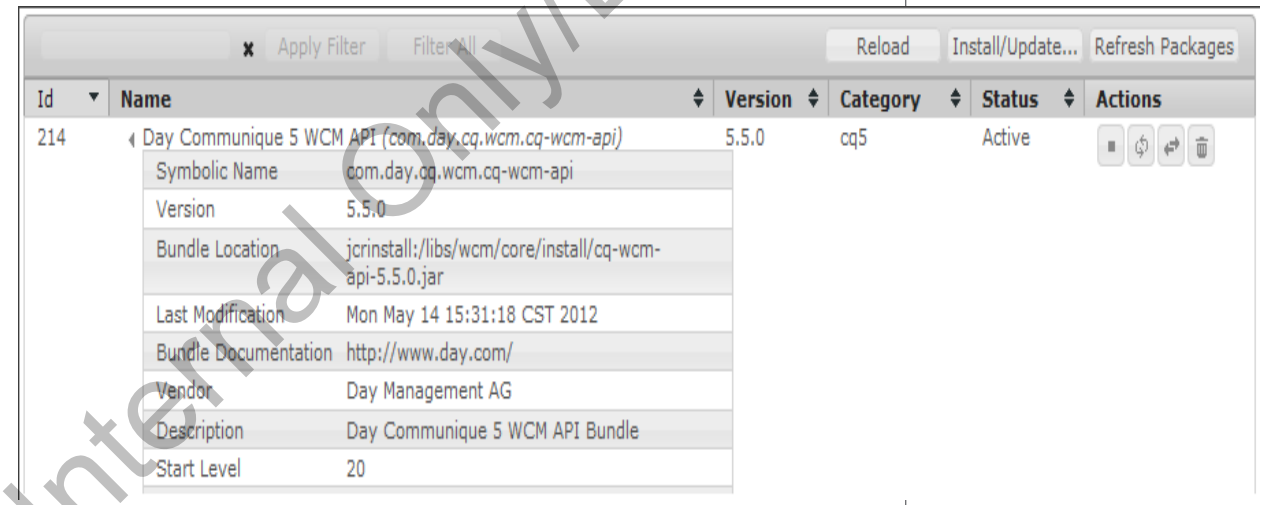
Goal

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.

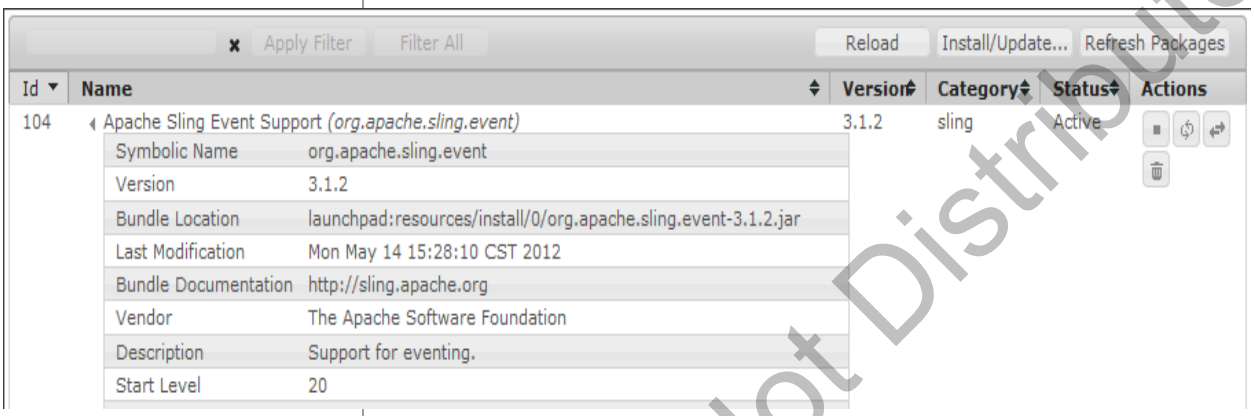
Steps

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

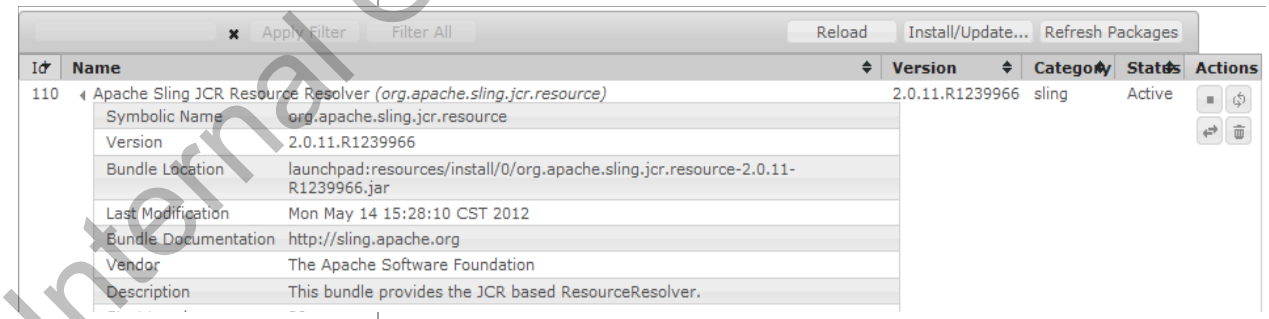
2. For this exercise you will need the bundles `com.day.cq.cq-replication`, `com.day.cq.wcm.cq-wcm-api`, `com.day.cq.cq-commons`, `org.apache.sling.jcr.resource` and `org.apache.sling.event`. These have already been added to the pom.xml files for your convenience and to save time. If you needed other bundles, you would find them in the Adobe CQ5 Web Console and add them as dependencies to your pom.xml files. Notice that the version number can be obtained from the Adobe CQ5 Web Console, and that when specifying the version number for org.apache.sling.jcr.resource, the version number in CQ5 Web Console is shown as 2.0.11.R1239966 but when written into the POM this is translated to 2.0.11-R1239966.



```
<dependency>
      <groupId>com.day.cq.wcm</groupId>
      <artifactId>cq-wcm-api</artifactId>
      <version>5.5.0</version>
      <scope>provided</scope>
</dependency
```



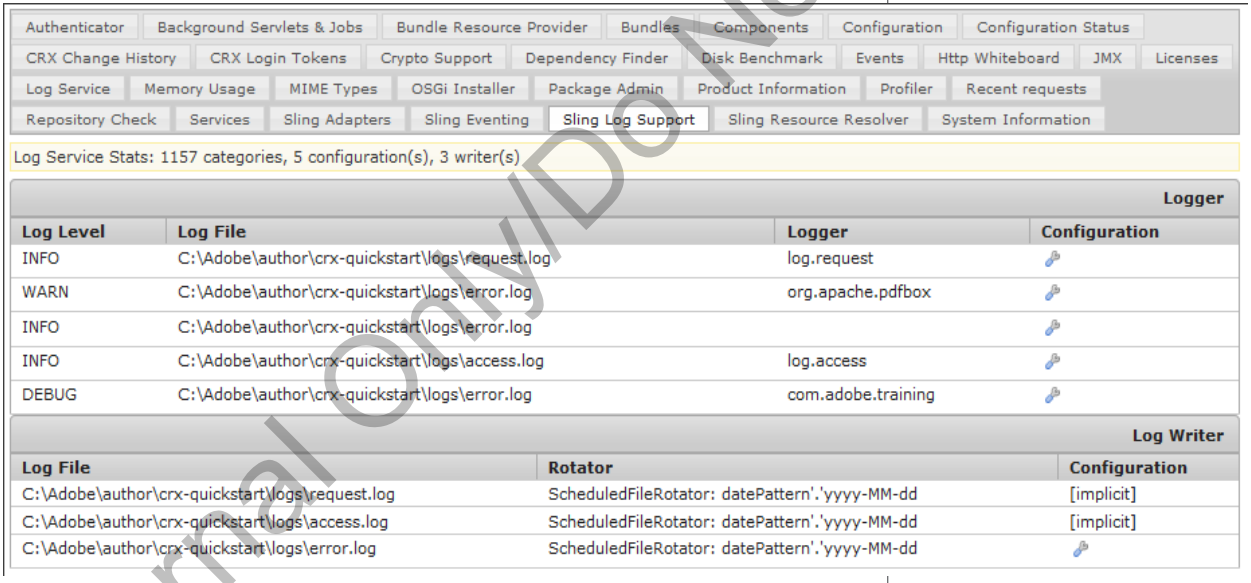

```
<dependency>
      <groupId>org.apache.sling</groupId>
      <artifactId>org.apache.sling.event</artifactId>
      <version>3.1.2</version>
      <scope>provided</scope>
</dependency>
```



```
<dependency>
      <groupId>org.apache.sling</groupId>
      <artifactId>org.apache.sling.jcr.resource</artifactId>
      <version>2.0.11-R1239966</version>
      <scope>provided</scope>
</dependency
```

NOTE: For the purposes of this exercise, we are using the Adobe CQ5 Web Console to define and configure the Logging Logger. Best Practices suggest configuring OSGi Bundles in the JCR as described in the documentation.

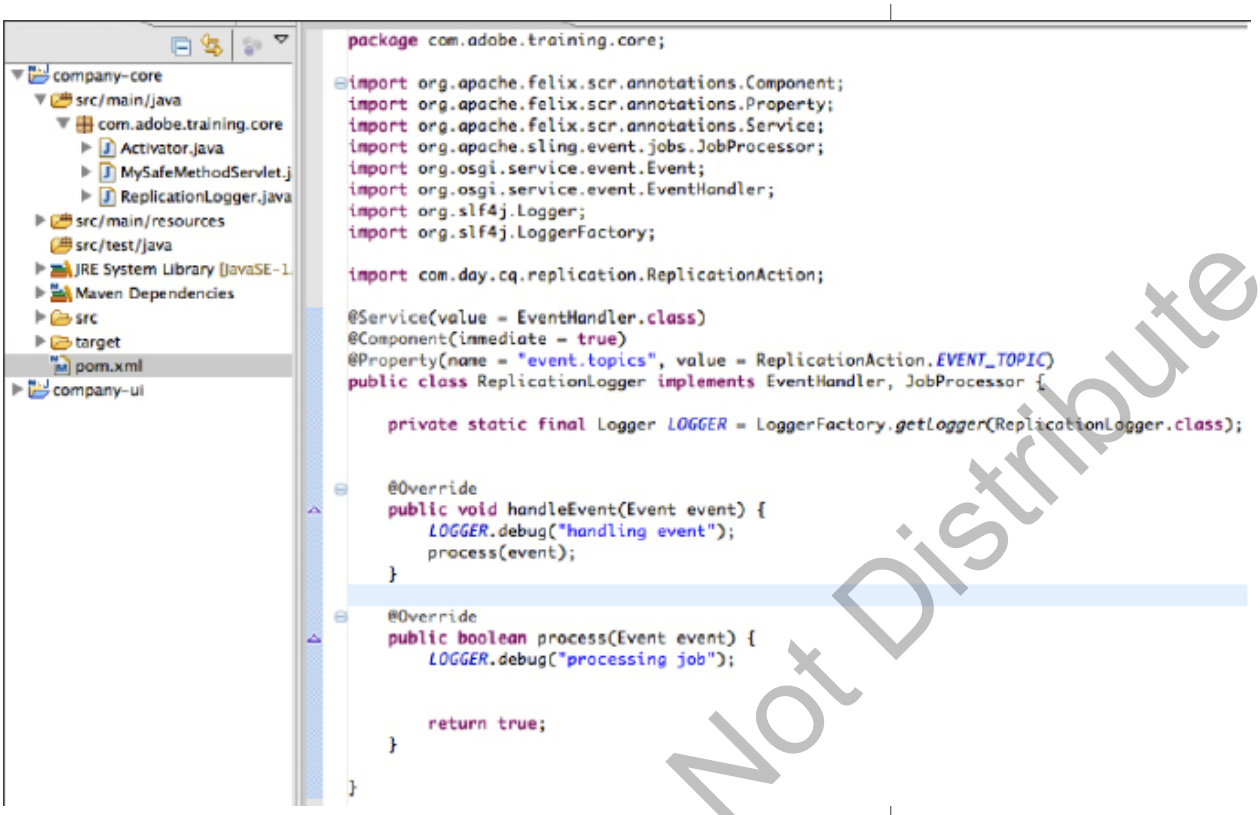
3. Set the log level for `com.adobe.training` to debug. This can be done in the Adobe CQ5 Web Console (though Adobe recommends that permanent configuration changes are normally made directly in the repository - see http://dev.day.com/docs/en/cq/current/deploying/configuring_cq.html for more details). 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.

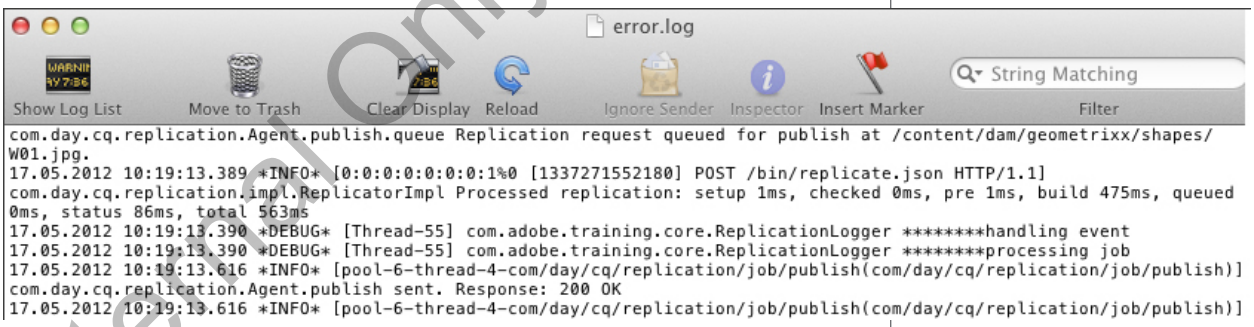
ReplicationLogger.java

```
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.info("********handling event");
     process (event);
  }
```

```
      @Override
public boolean process(Event event) {
      LOGGER.info("********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.info
      ("********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: Note the resource resolver and how it is used with `adaptTo` to obtain a PageManager.

NOTE:

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.

5 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 ? * MON-FRI", 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 - 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.

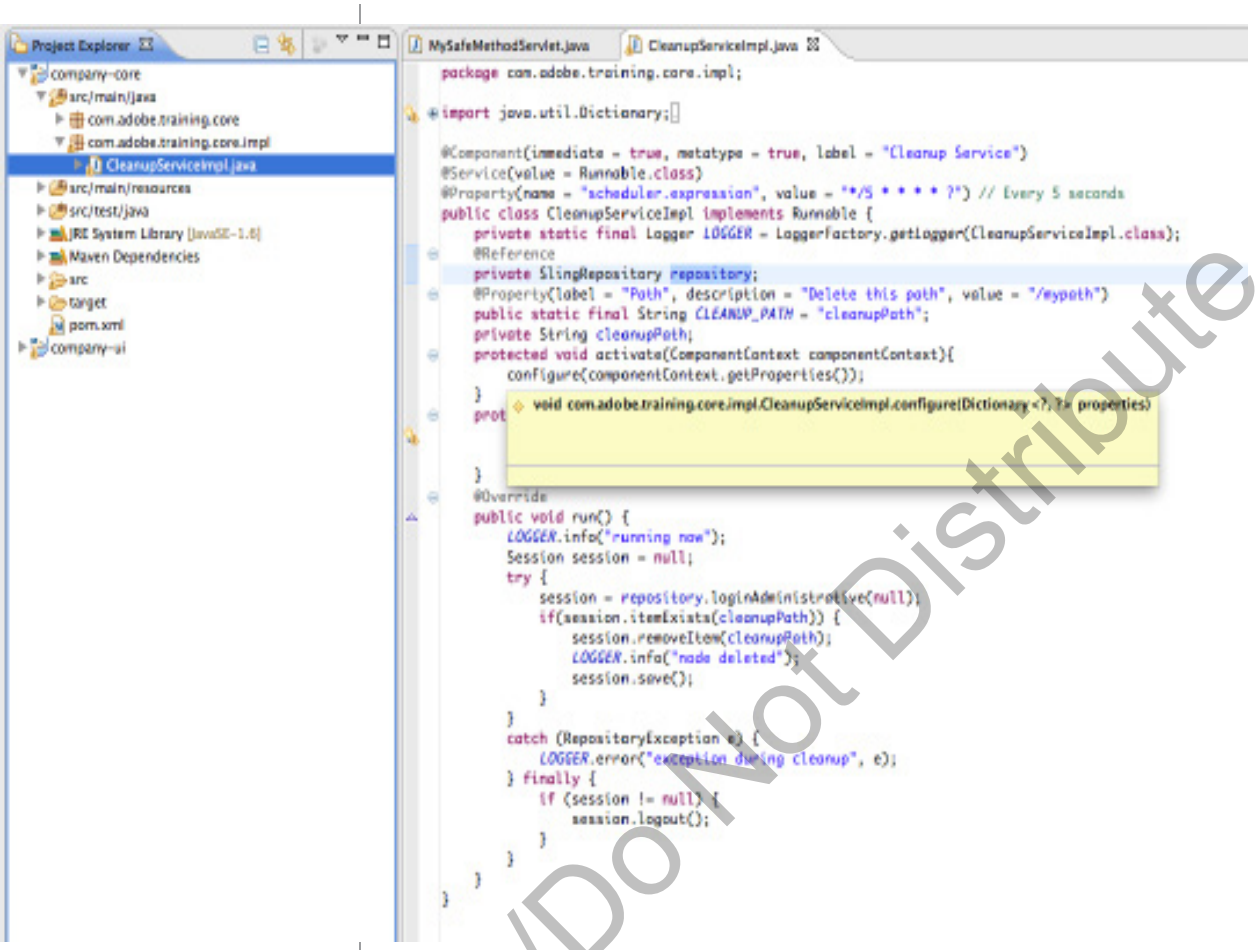
Steps

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);
```

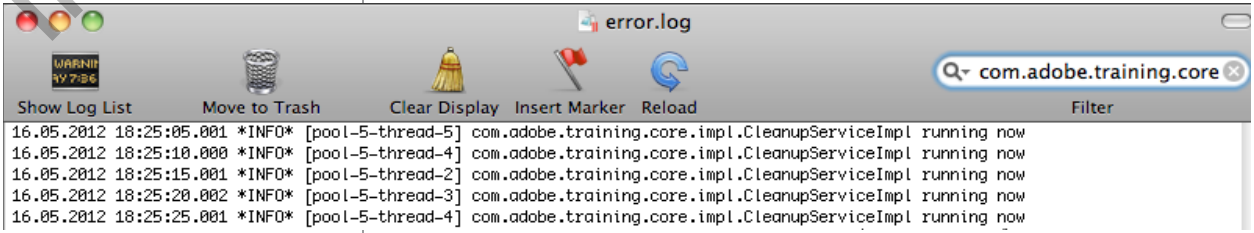
```
@Reference
private SlingRepository repository;

@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:

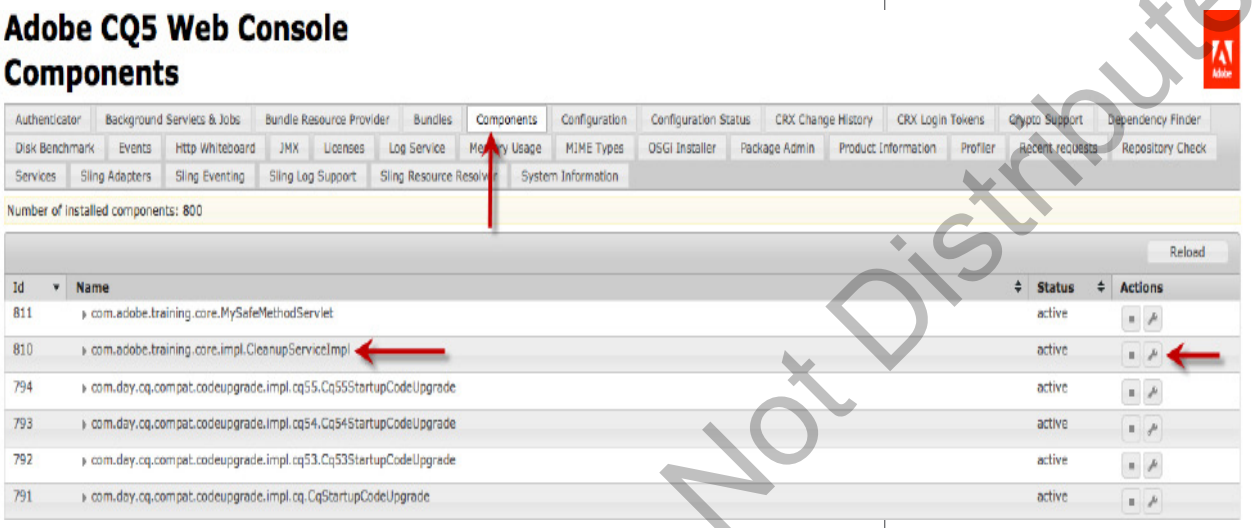
- `metatype` = true enables configuration in CQ5 Web Console.
- `SlingRepository` can be used for administrative logins, using loginAdministrative(null). Never forget to logout afterwards (in `finally`).
- `configure` will be called when the config changes. This is used to obtain the new value for the 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).

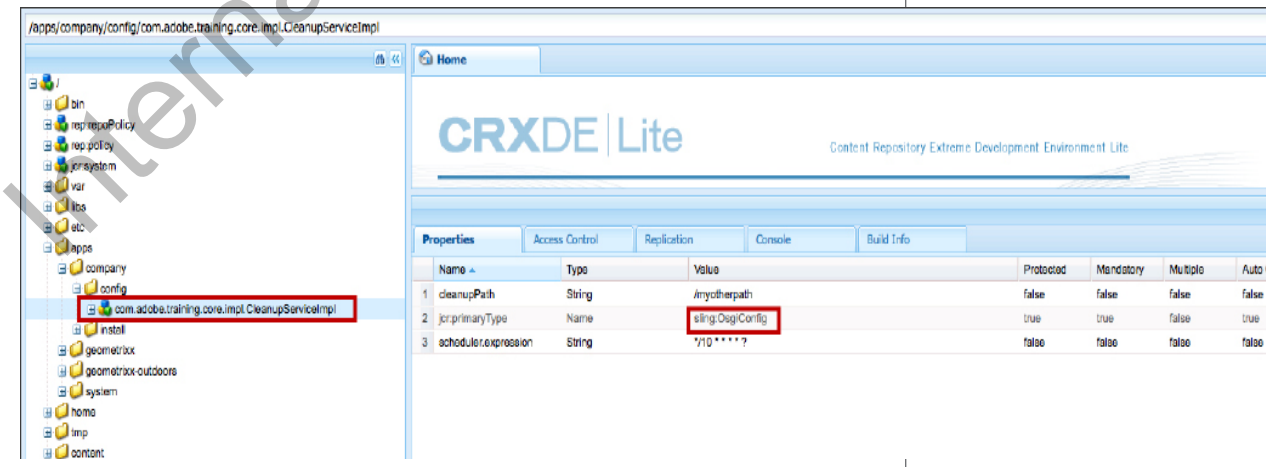
5. Click on the wrench to the right of the com.adobe.training.core.impl entry to open the Configuration pop up window. Configure a different path:



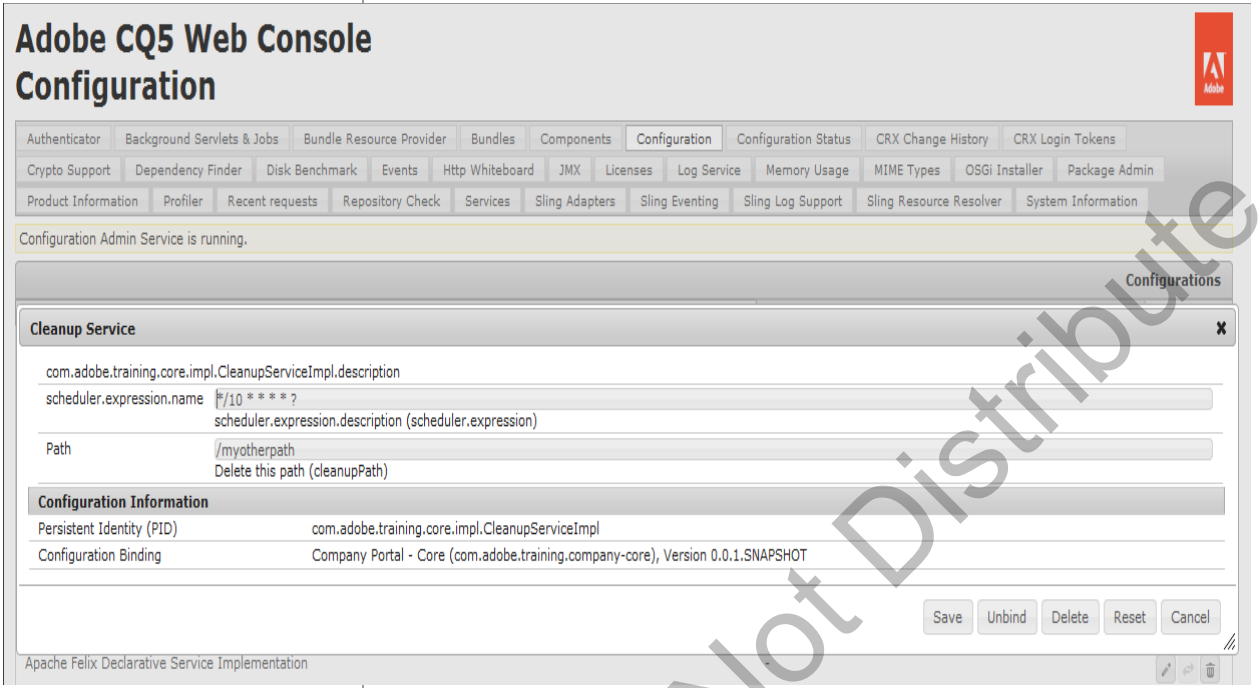
6. Now create a node with the new path name and confirm that this is now deleted.

7. Finally we will set the configuration in the repository. In CRXDE Lite create a folder (type nt:folder) `/apps/company/config`, 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 CQ5 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.

Congratulations! You have successfully created an 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.

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.

Chapter Six

6 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 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, CQ5 developers will use the node types defined for the CQ5 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 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

Data Store

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 to the Persistence Manager (PM) store. 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 minimum object length default is 100 bytes;; smaller objects are stored inline (not in the data store). The maximum value is 32000 because Java does not support strings longer than 64 KB in writeUTF.

Characteristics of the Data Store:

- Content-addressed storage for large binary properties
  - Arbitrarily sized binary streams
  - Addressed by MD5 hash
  - String properties not included in the data store
- Fast delivery of binary content
  - Read directly from disk
  - Can also be read in ranges
- Improved write throughput
  - data store operations do not block other users because they are performed outside the persistence manager
- Multiple uploads can proceed concurrently (within hardware limits)
  - Cheap copies
    - only the identifier is copied
- Garbage collection used to reclaim disk space

Cluster Journal

Whenever CRX writes data it first records the intended change in the journal. Maintaining the journal helps ensure data consistency and helps the system to recover quickly from crashes. As with the PM and data stores, the journal can be stored in a file system (the default) or in a database.

Journal of all persisted changes in the repository, including:

- Content changes
- Namespace, nodetype registrations, etc.

The Journal is used to keep all cluster nodes in sync

- Observation events to all cluster nodes (see JackrabbitEvent.isExternal)
- Search index updates
- Internal cache invalidation

Old events need to be discarded eventually

- No notable performance impact, just extra disk space
- Keep events for the longest possible time so a node can be offline without getting completely recreated

Persistence Manager

Each workspace in the repository can be separately configured to store its data through a specific persistence manager (the class that manages the reading and writing of the data). Similarly, the repository-wide version store can also be independently configured to use a particular persistence manager. A number of different persistence managers are available, capable of storing data in a variety of file formats or relational databases.

Identifier-addressed physical storage for nodes and properties

- Each node has a UUID, even if not *mix:referenceable*
- Essentially a key-value store, even when backed by a RDBMS
- Also keeps track of node references

Think about bundles as units of content

- Bundle persistence managers store each node together with all the properties as one unit
- Bundle = UUID, type, properties, child node references, etc.
- Only large binaries stored elsewhere in the data store
- Designed for balanced content hierarchies, avoid too many child nodes

The Persistence Manager (PM) performs atomic updates. A save() call persists the entire transient space as a single atomic operation.

Each workspace has its own Persistence Manager (PM) (and one for the shared version store. The default PM for CQ5 is TarPM, which stores data in .tar files (append only).

Query Index

CRX's inverse index is based on Apache Lucene. This allows for:

- Flexible mapping from terms to node identifiers
- Special handling for the path structure

Most index updates are synchronous. Long full text extraction tasks are handled in background. Other cluster nodes will update their indexes at next cluster sync

Everything indexed by default. You can tweak the indexing configuration for improvements in indexing functionality, performance and disk usage.

There is one index per workspace (and one for the shared version store) Indexes are not shared across a cluster, indexes are local to each cluster node.

Jackrabbit

The Apache Jackrabbit™ content repository is a fully conforming implementation of the Content Repository for Java Technology API (JCR, specified in JSR 170 and JSR 283. Note the next release of the JCR specification is JSR 333, which is currently under work.

Apache Jackrabbit is a project of the Apache Software Foundation.

Repository Configuration

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.xml*** configuration file for each new workspace.

Main elements for configuration using ***repository.xml*** are:

- clustering,
- workspace,
- versioning,
- security,
- persistence,
- search index,
- data store,
- file system

Jackrabbit Persistence

The Persistence Manager (PM) is an "internal" Jackrabbit component that handles the persistent storage of content nodes and properties. Property values are also stored in the persistence manager, with the exception of large binary values (those are usually kept in the Data Store).

The PM sits at the very bottom layer of the Jackrabbit system architecture. Reliability, integrity and performance of the PM are crucial to the overall stability and performance of the repository. If e.g. the data that a PM is based upon is allowed to change through external means, the integrity of the repository would be at risk.

NOTE: If you use a database persistence manager, the configured database connection must not be under the control of an external transaction manager. Jackrabbit implements distributed XA transaction support on a higher level, and expects to be in full control of the underlying database connection.

In practice, a persistence manager is any Java class that implements the PersistenceManager interface and the associated behavioral contracts. Jackrabbit contains a set of built-in persistence manager classes that cover most of the deployment needs. There are also a few contributed persistence managers that give additional flexibility.

Jackrabbit provides for a set of pluggable Persistence Managers

e.g. Apache Derby Persistence Manager, MySQL, MS SQLServer, Oracle.

CRX ships with Tar Persistence Manager, which stores content in tar files.

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

Jackrabbit Search

JCR specifies search capabilities, but not how it shall be implemented.

Jackrabbit includes Apache Lucene as its Search and Indexing engine. Lucene synchronously indexes content in the repository.

Search Index

Per-workspace search indexes are based on Apache Lucene. By default everything is indexed. Use index configuration to customize Lucene's behavior. The default configuration file is embedded in the repository bundle deployed in the OSGi container.

Do not modify, but examine the procedure in
http://dev.day.com/content/kb/home/cq5/CQ5SystemAdministration/SearchIndexingConfig.html

Configuration (examine all the documentation at
http://wiki.apache.org/jackrabbit/Search

- respectDocumentOrder
- resultFetchSize (number of results the query handler should initially fetch when a query is executed)

Indexing Configuration

If you wish to configure the indexing behavior of Lucene, you need to add a parameter to the SearchIndex element in your ***workspace.xml*** and ***repository.xml*** file.

Boosting

Any particular property or search term can be "boosted" either at indexing time or at runtime. Boosting allows you to control the relevance of a document by boosting its term. For example, boosting can specify that a match on the "Title" property of a node is more important than the match on the "Text" property.

At the same time, the terms of a full-text query can also be "boosted," which means that at runtime, a user or developer can specify which terms in the query are more important than other terms and therefore a match on these terms has to be ranked higher.

You control which properties of a node are indexed by defining index rules. By default, all properties of a node are indexed.

You configure a boost value for the nodes that match the index rule. Lucene provides the relevance level of matching documents based on the terms found. By default, the boost factor is 1. Although the boost factor must be positive, it can be less than 1 (for example, 0.2).

The higher the boost factor (a reasonable range is 1.0 - 5.0), the more relevant the term will be. For example, to define an index rule to index only properties named "Text" and boost nodes of type nt:unstructured:

```
<index-rule nodeType="nt:unstructured"
     boost="2.0"
     condition="@priority = high">
<property>Text</property>
</index-rule>
```

Aggregates

Sometimes it is useful to include the contents of descendant nodes into a single node to easier search on content that is scattered across multiple nodes.

Jackrabbit allows you to define index aggregates based on relative path patterns and primary node types.

```
<!-- CQ Page for jcr:contains(jcr:content, "...") searches -->
<aggregate primaryType="cq:PageContent">
<include>*</include>
<include>*/*</include>
```

```
<include>*/*/*</include>
<include>*/*/*/*</include>
</aggregate>
```

NOTE: Both index rules and index aggregates influence how content is indexed in CRX. If you change the configuration, the existing content is not automatically re-indexed according to the new rules. You therefore must manually re-index the content when you change the configuration.

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?

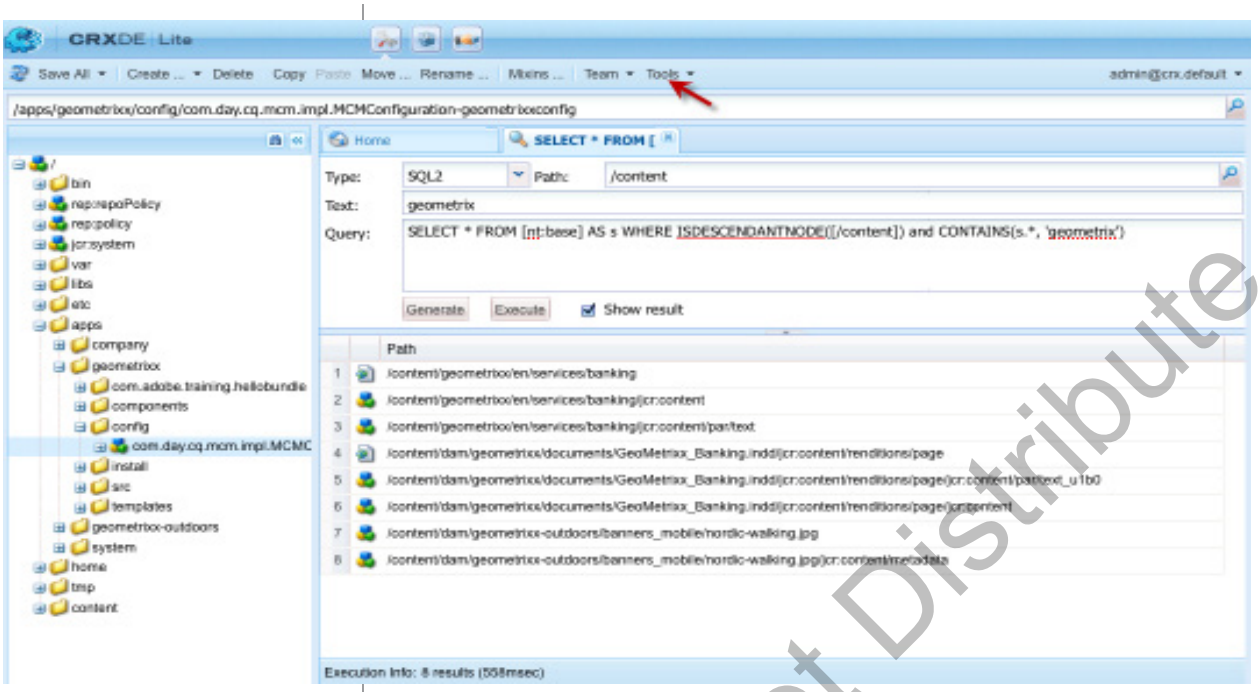
- Path constraints

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.



Debugging Queries

The CRX provides a mechanism to log query execution information. It can be done in the Adobe CQ5 Console.

1. Open the Adobe CQ5 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.core.query.QueryImpl

4. Then click on save and after refreshing.

5. Create a new instance of the Factory Configuration Apache Sling Logging Writer Configuration.

   Log File: logs/query.log
   Number of Log files: 5
   Log File Threshold: '.'yyyy-MM-dd

The new Logging Logger will direct its output to a new log file called query.log.

NOTE: For more information about logging see http://dev.day.com/docs/en/cq/current/deploying/configure_logging.html

EXERCISE - Search

Goal

- write a servlet that implements a Sling selector. The servlet shall perform a full text search below the requested node. (the results shall only be pages, but we will select for nt:unstructured, as the page content in jcr:content is of that type)
- The query results are returned as JSON

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"*.

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

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

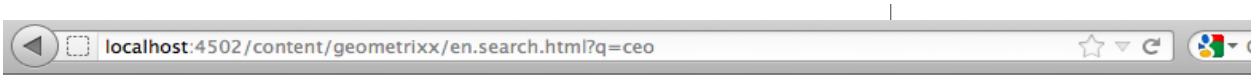
```
@Override
    public final void doGet|final SlingHttpServletRequest request, final SlingHttpServletResponse response|
      throws ServletException, IOException {
      response.setHeader("Content-Type", "application/json");
      JSONObject jsonObject = new JSONObject();
      JSONArray resulyArray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

      try {
          //this is the current node that is requested, in case of a page, it is the jcr:content node
          Node currentNode - request.getResource().adapt To (Node.class);
          PageManager pageManager = request.getResource().getResourceResolver().adaptTo(PageManager.class|;

          //node chat 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 = performSearchWithSQL(queryRoot, queryTerm);
              while (searchResults.hasNext()) resultArray.put(searchResults.nextNode() .getPAth());
            jsonObject.put(results - ", resultArray);
          }
      }
      catch (Exception e) {
         e.printStackTrace();
      }
      response.getWriter().print(jsonObject.toString());
    }
```

3. Implement actual search method NodeIterator performSearch(Node queryRoot, String queryTerm).

```
private NodeIterator performSearch(Node queryRoot, String
queryTerm)
        throws RepositoryException  {
        //JQOM infrastructure
           QueryObjectModelFactory qf = queryRoot.getSession().
getWorkspace().getQueryManager().getQOMFactory();
       ValueFactory vf = queryRoot.getSession().getValueFactory();



        //SELECTORS
        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);

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

    }
```

4. Build and deploy.

5. Results are available at e.g. http://localhost:4502/content/geometrixx/en.search.html?q=ceo




```
{
  - results: [
        "/content/geometrixx/en/company/management/jcr:content",
        "/content/geometrixx/en/company/management/jcr:content/par/title",
        "/content/geometrixx/en/company/bod/jcr:content/par/reference_0",
        "/content/geometrixx/en/company/management/jcr:content/par/james_doe_cto",
        "/content/geometrixx/en/company/management/jcr:content/par/john_doe_ceo",
        "/content/geometrixx/en/company/bod/jcr:content",
        "/content/geometrixx/en/company/jcr:content/par/our_management_team",
        "/content/geometrixx/en/company/jcr:content"
    ]
}
```


6. Implement same functionality using SQL2.

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

```
  private NodeIterator performSearchWithSQL(Node queryRoot,
String queryPerm

      throws RepositoryException {
```

7 JCR Versioning, Observation

EXERCISE - JCR Observation Listener

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<CR>`

or

`Repository.OPTION _ JOURNALED _ OBSERVATION _ SUPPORTED<CR>` 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

```
String Event.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()returnsaMapcontainingparameterinformationfrom the method that caused the event (see §12.4.3 *Event Information on Move and Order).*

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 l

```
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 that 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.

 Note that specifying an empty array instead of null results in no nodes being listened to.

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(Event Listener listener)`

User Data

Through: `void ObservationManager.setUserData(String userData)`

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 methos. 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 CQ5-level

In CQ5, 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.

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.annotaions.Component;
Import org.apache.felix.scr.annotaions.Reference;
Import org.apache.sling.jcr.opi.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.logginadministrative(null);
              observationManager = session getWorkspace().getObservationManager();

              observationManager.addEventListener(this, Event.PROPERTY _ ADDED | Event.
PROPERTY _ CHANGED,-/-, true, null,
                           null, true);
              LOGGER.infor('********added JCR event listener");
}
protected void deactivate(ComponentContext componentContext) {
              try {
                     if (observationManager !=null) {
                           observationManager.removeEventListener(this);
                           LOGGER info("********removed JCR event listener");
                     }
              }
              finally{
                     if (session != null){
                           session.logout();
                           session - null;
                     }
              }
}
```

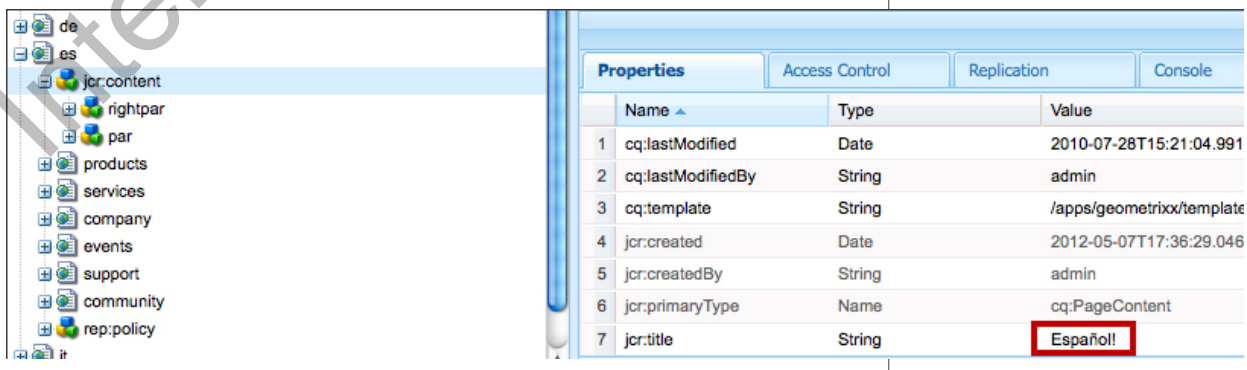
Note:
The method call `addEventListener`: Event types are bitwise OR'ed. The last parameter (noLocal) prevents changes done by this session to be sent to the listener (which would result in an endless loop in this case). You could also register for events sent by specific node types or at specific paths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

2. Implement the onEvent method of the EventListener interface:

```
public void onEvent(EventIterator ot) {
              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);
                     }
              }
}
```

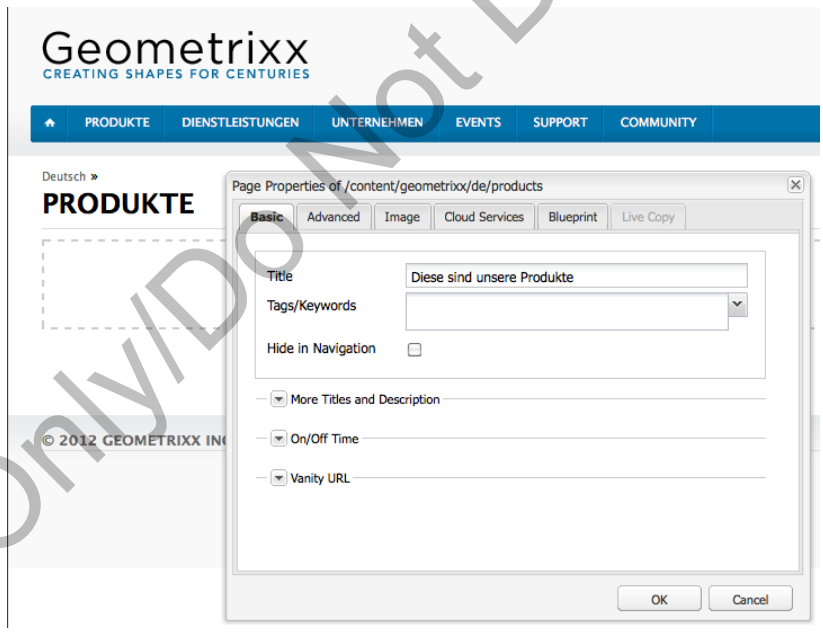
3. Deploy the component by executing:

mvn clean install -P bundle
Check that the component is installed:

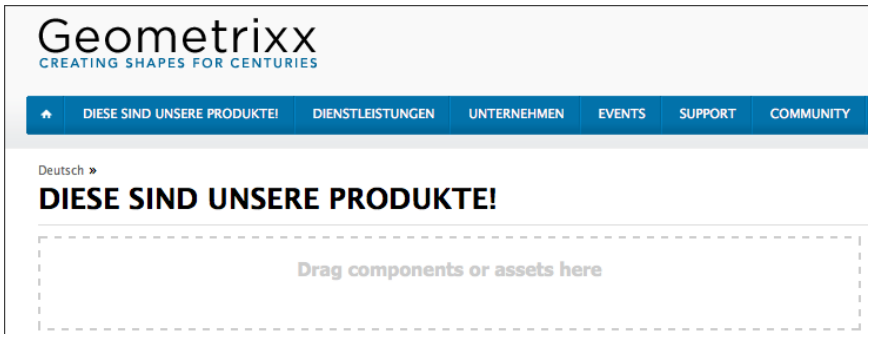


4. Change a property called "jcr:title" in CRXDE Lite. Refresh to see the added "!".

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



6. Click ok

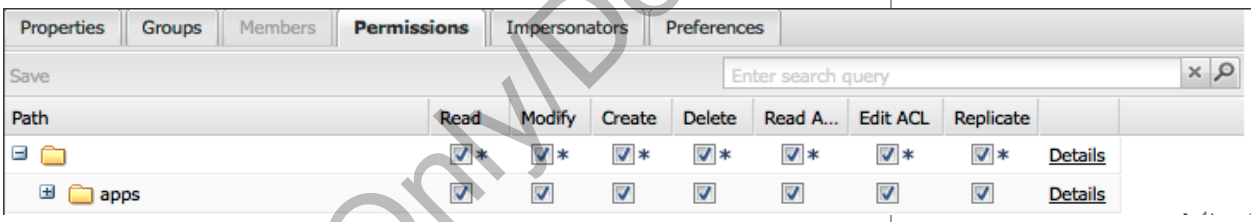


8 Users, Groups, Permissions

CQ 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 CQ 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 CQ (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.

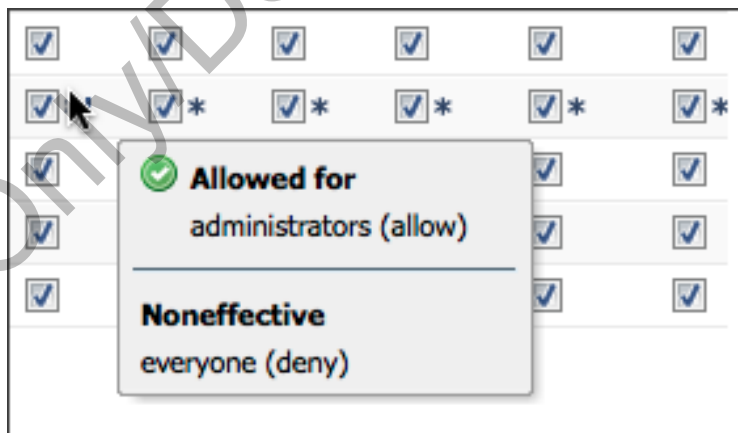
If a permission is not inherited from the parent node but has at least one local entry for it, then the following symbols are appended to the check box. A local entry is one that is created in the CRX 2.3 interface (Wildcard in the path of ACLs currently can only be created in CRX.)

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 jcr: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 jcr:content are specifically excluded because the creation of jcr:content and its child nodes are consierd a page modification. This only applies to nodes defining a jcr: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 jcr:content are specifically excluded as removing jcr:content and its child nodes is considered a page modification. This only applies to nodes defining a jcr: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

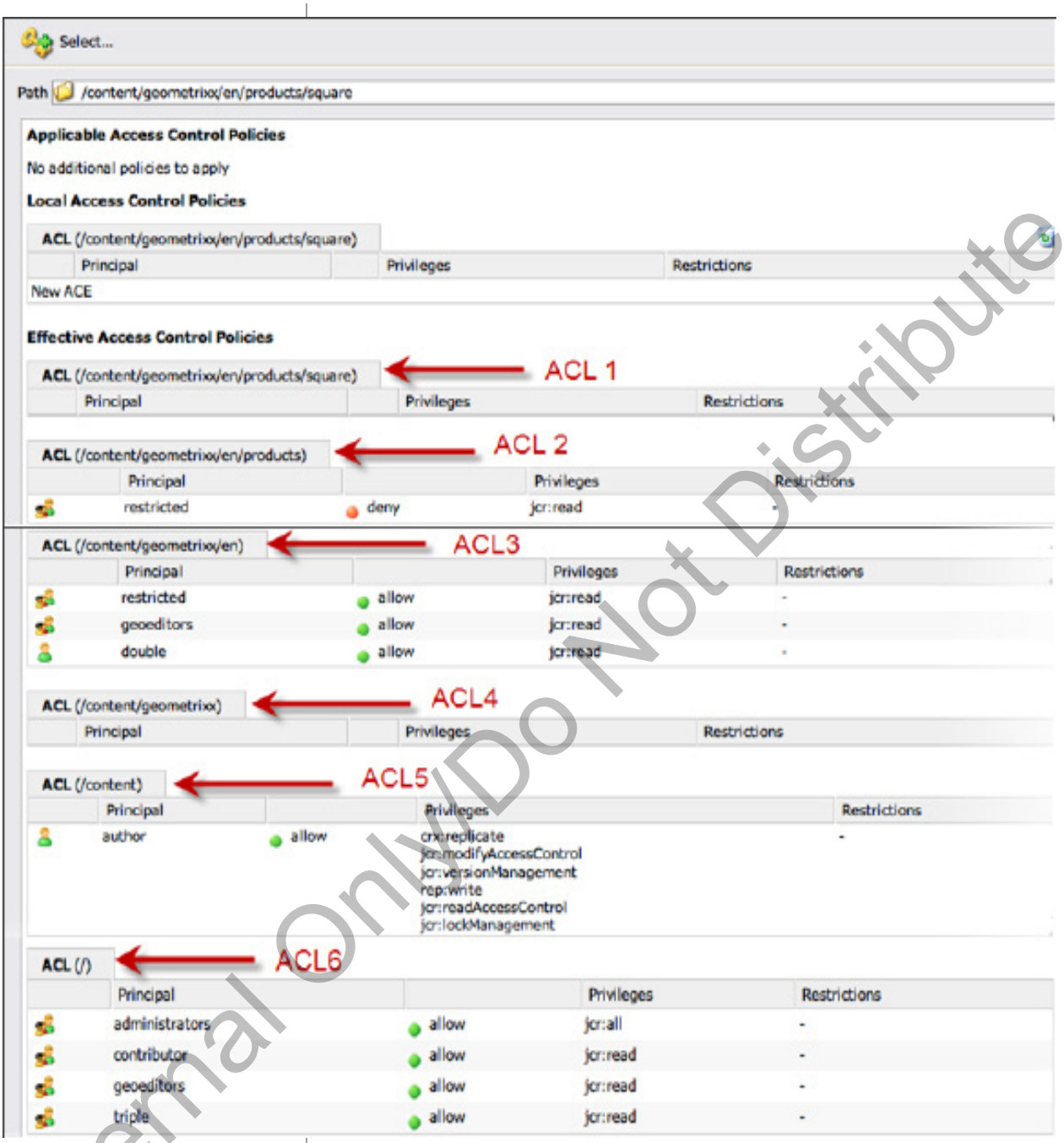
CQ WCM uses Access Control Lists (ACLs) to organize the permissions being applied to the various pages.

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 CQ 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 CQ will go up another level to ACL 3.

ACL 3 will be applied if the user belongs to the group **geoeditors** or **doble** 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 CQ will go up one level to ACL 4.

ACL 4 being empty, if CQ reach this level it will go up again one level to ACL 5.

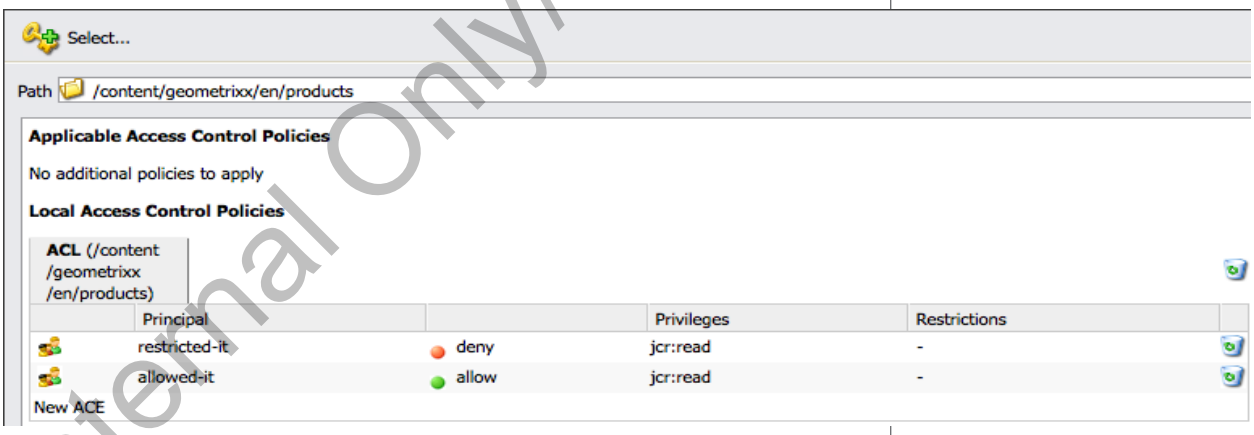
ACL 5 will be applied if the user belongs to the **author** group and then the user will have the access granted to the page. If the user is not part of the group name CQ will go up one level to ACL 6.

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 CQ 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 - ACLs

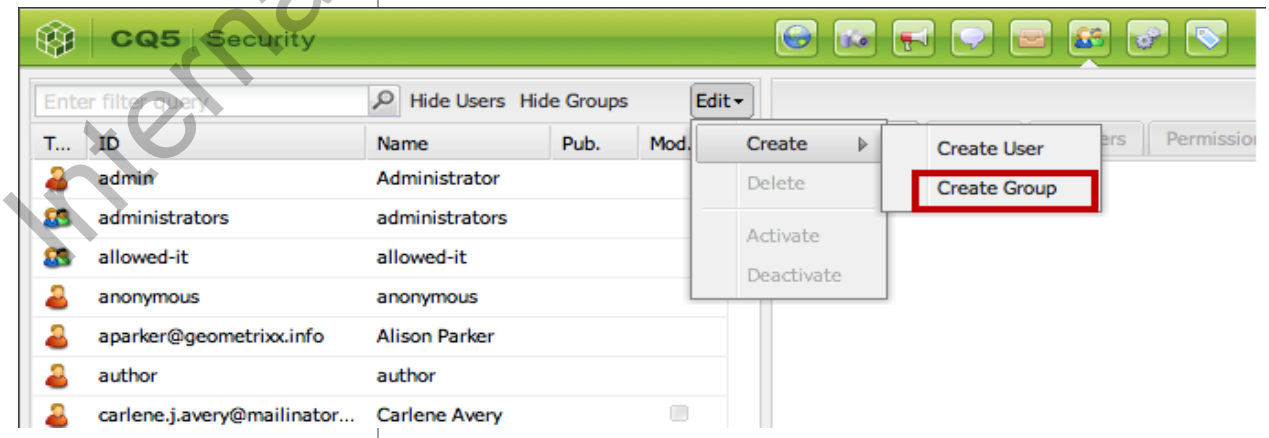
Goal

We are going to create a user which will be part of two groups with two different ACLs and we will modify them programmatically

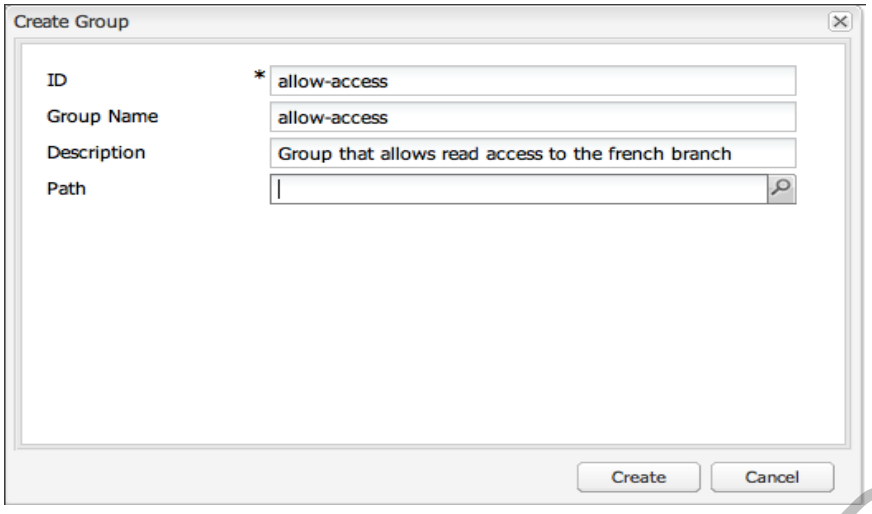
Steps

1. Create a group "allow-access" the group should have read access to /content/geometrixx/fr.

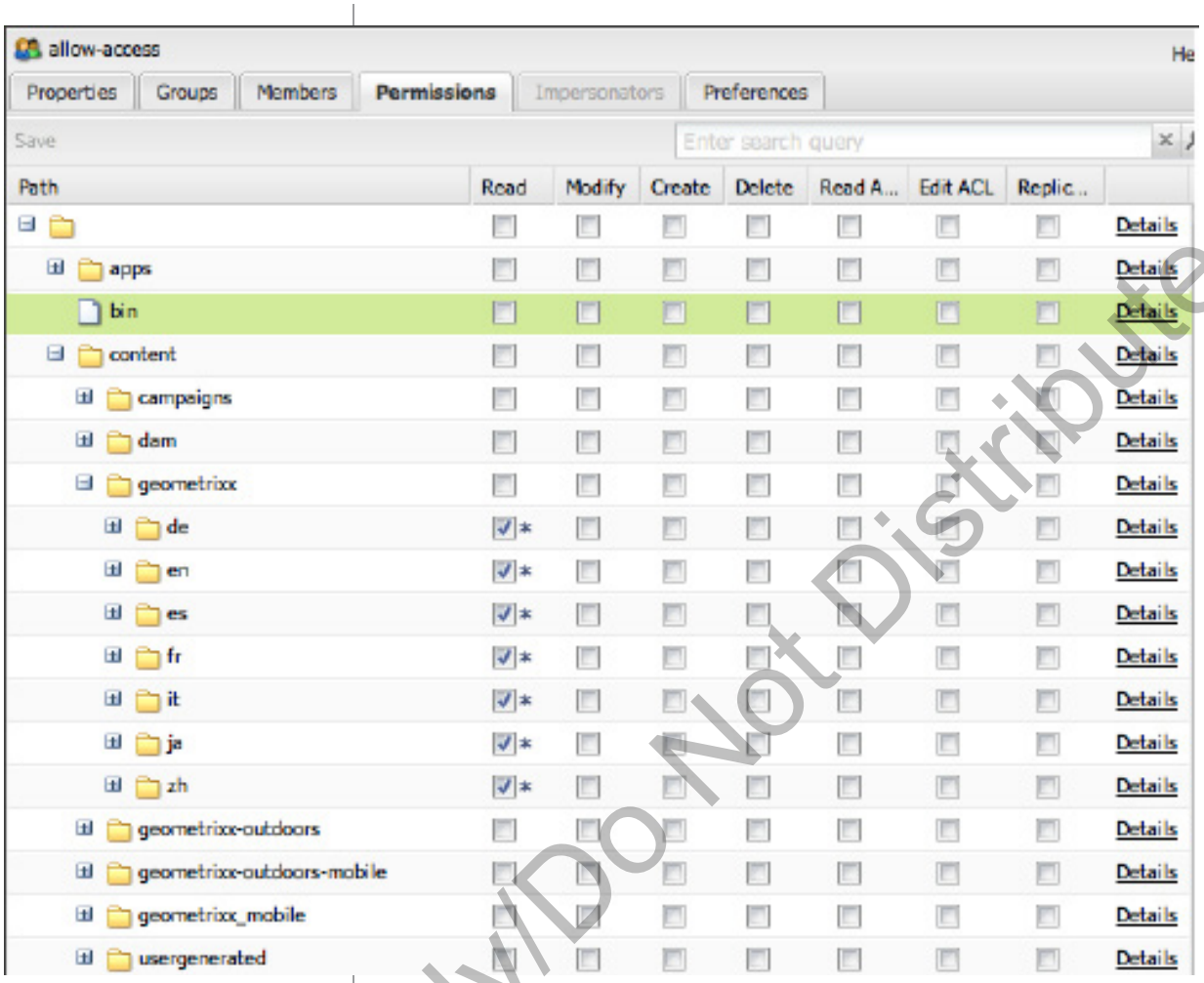
   Choose the user menu, then on the left pane click on the Edit button. A drop down menu will allow you to Create your group



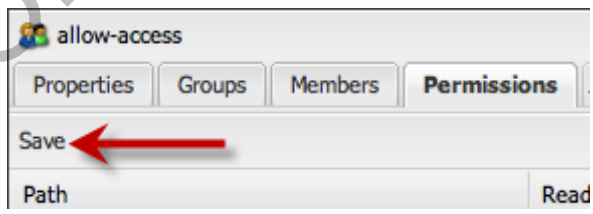
Create the group as following:



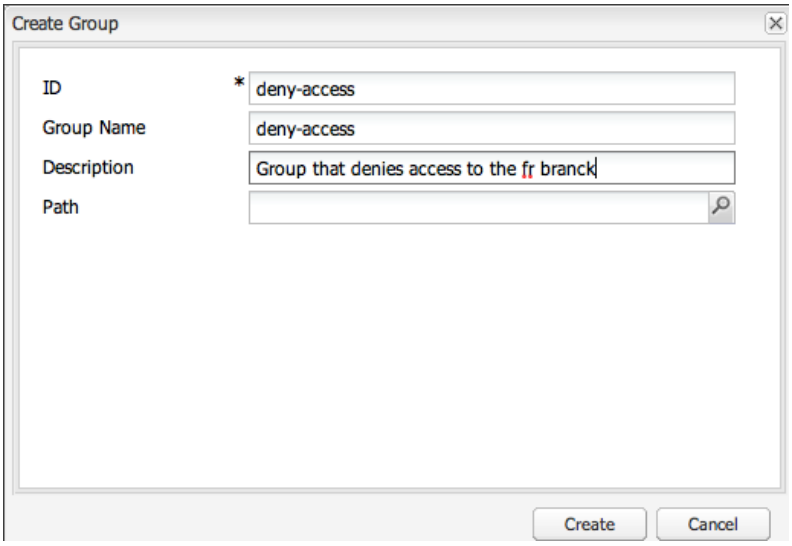
Open the newly created group and assign read rights to the all the languages branches, as following:



Don't forget to click on Save after you're done

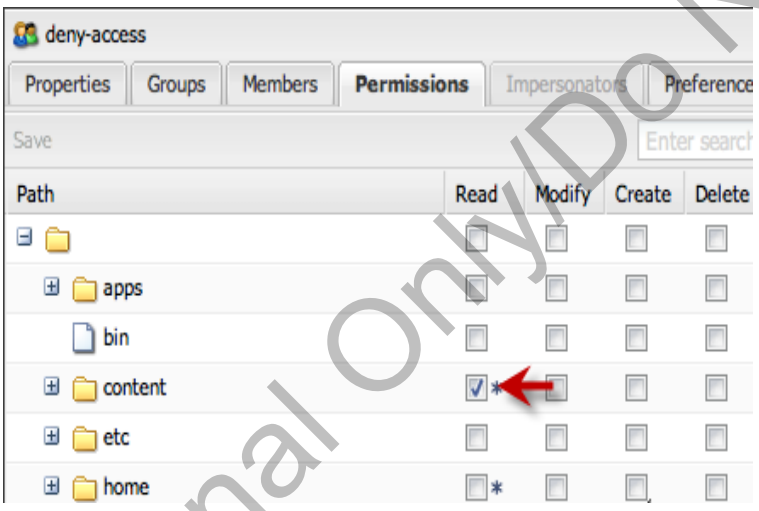


2. Create another group "deny-access", the group should not have read access to /content/geometrixx/fr.

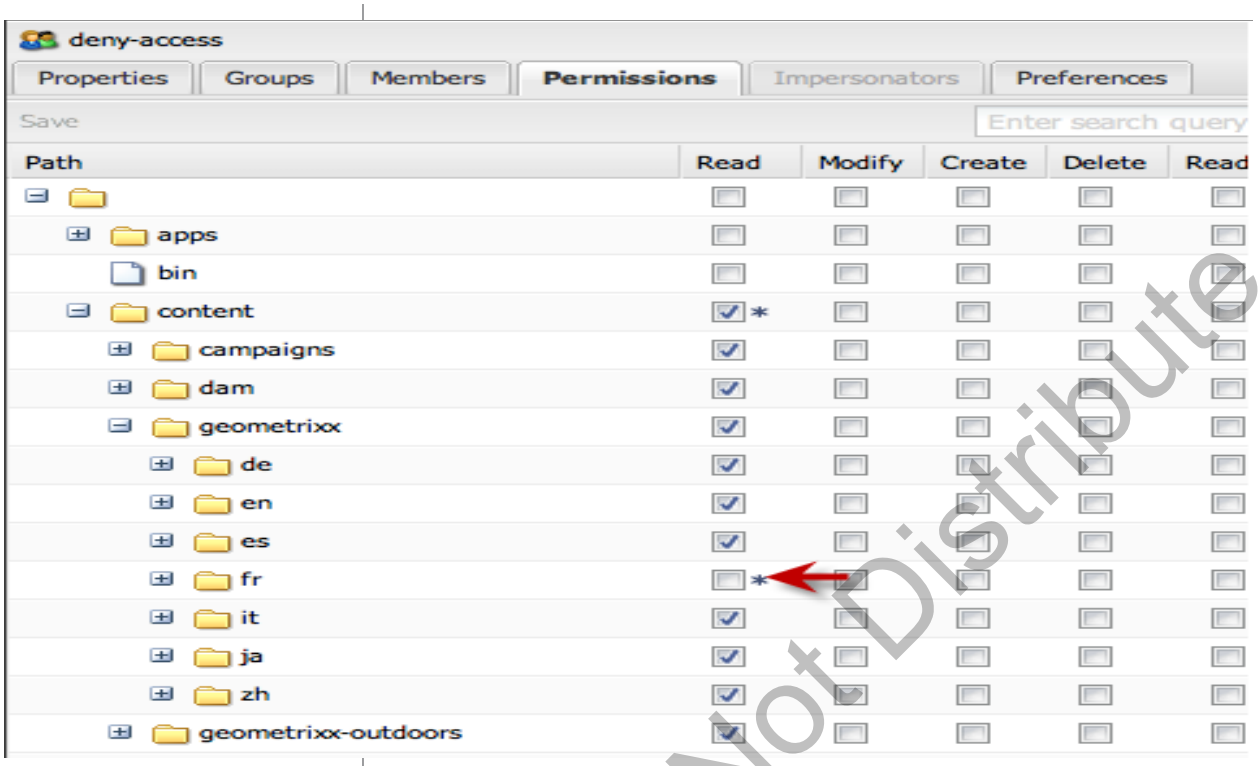


Open the newly created group and assign read rights to the all the languages branches, except for the French one as following.

For that first provide read access to the /content node

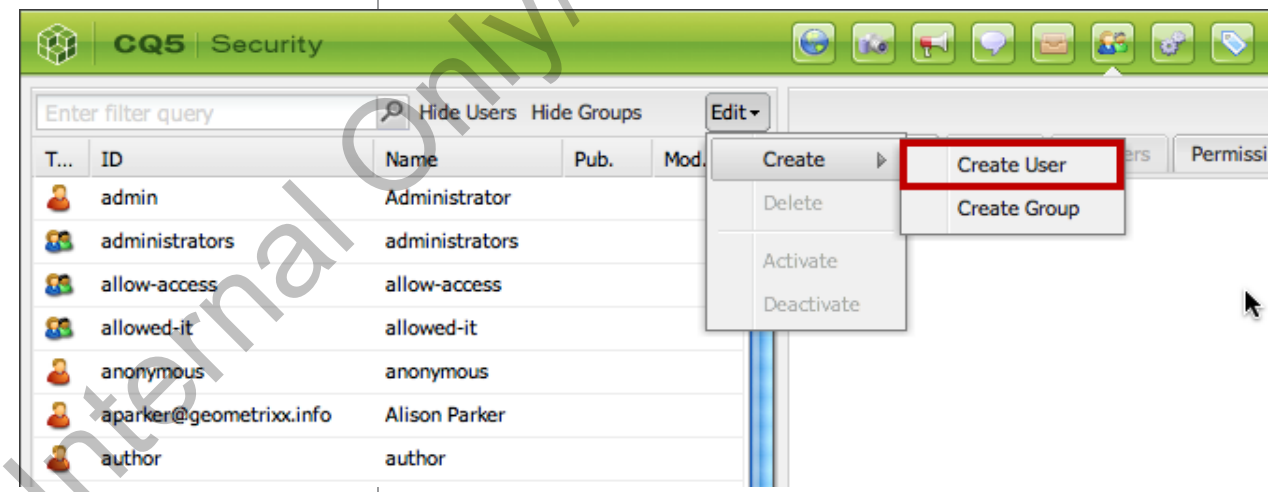


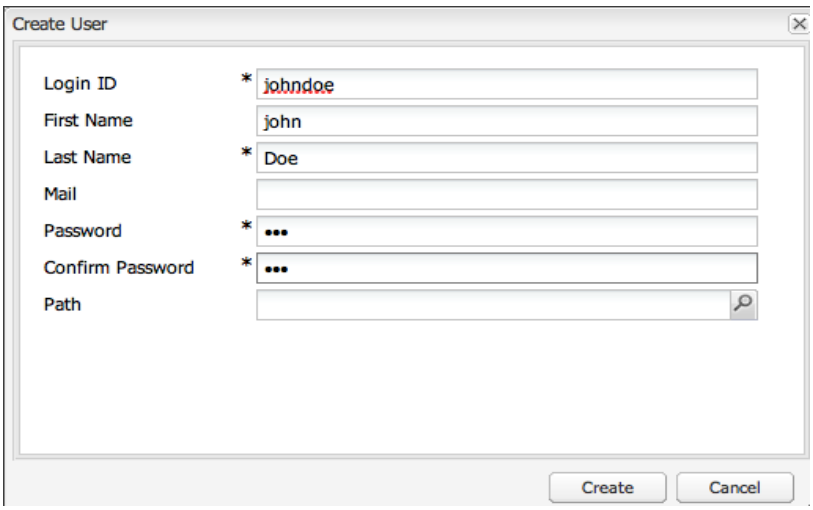
Then expand the content node and deselect the fr node



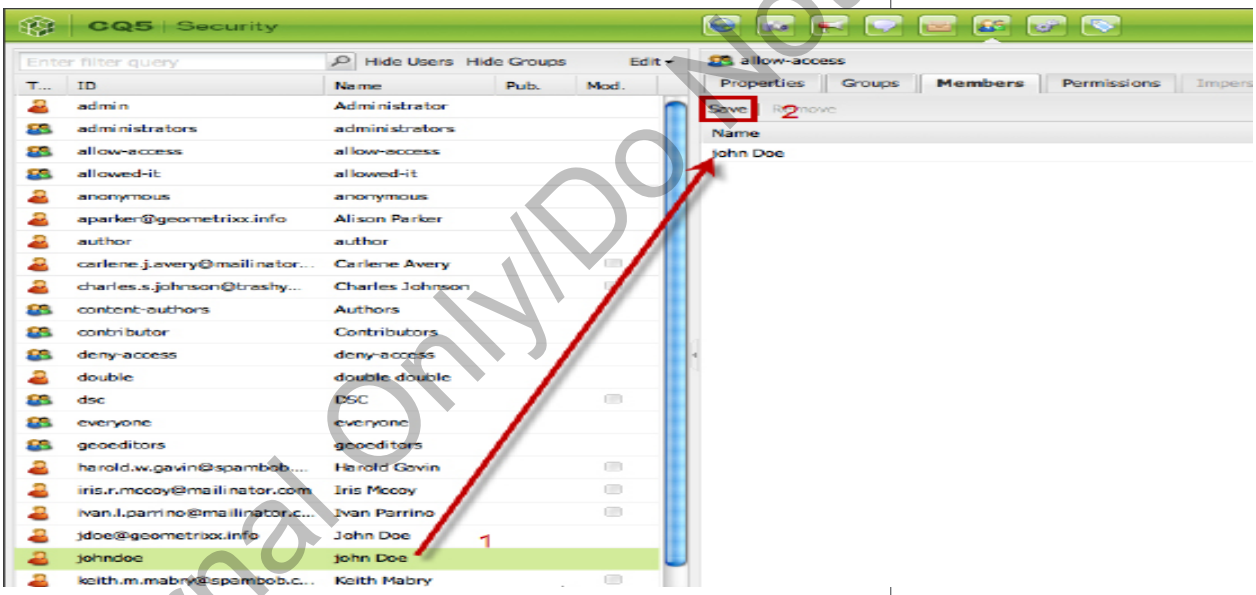
Don't forget to save the changes by clicking on Save!

3. Create the user John Doe, for that proceed the same way as you did when creating a new group, and choose "Create User"





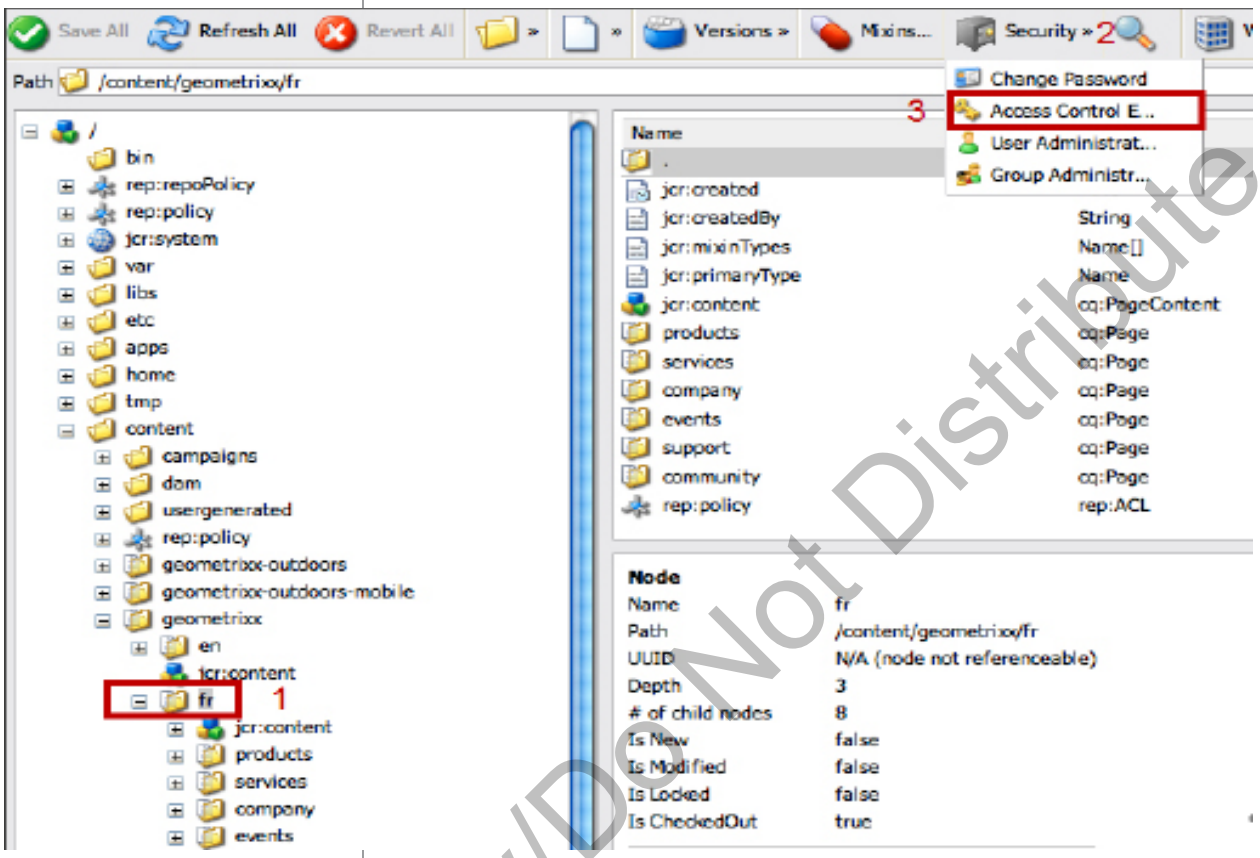
4. Add the user John Doe to the allow-access and deny-access group. For that you'll have to open each group, click on Members and Drag & Drop the user John Doe to the member list



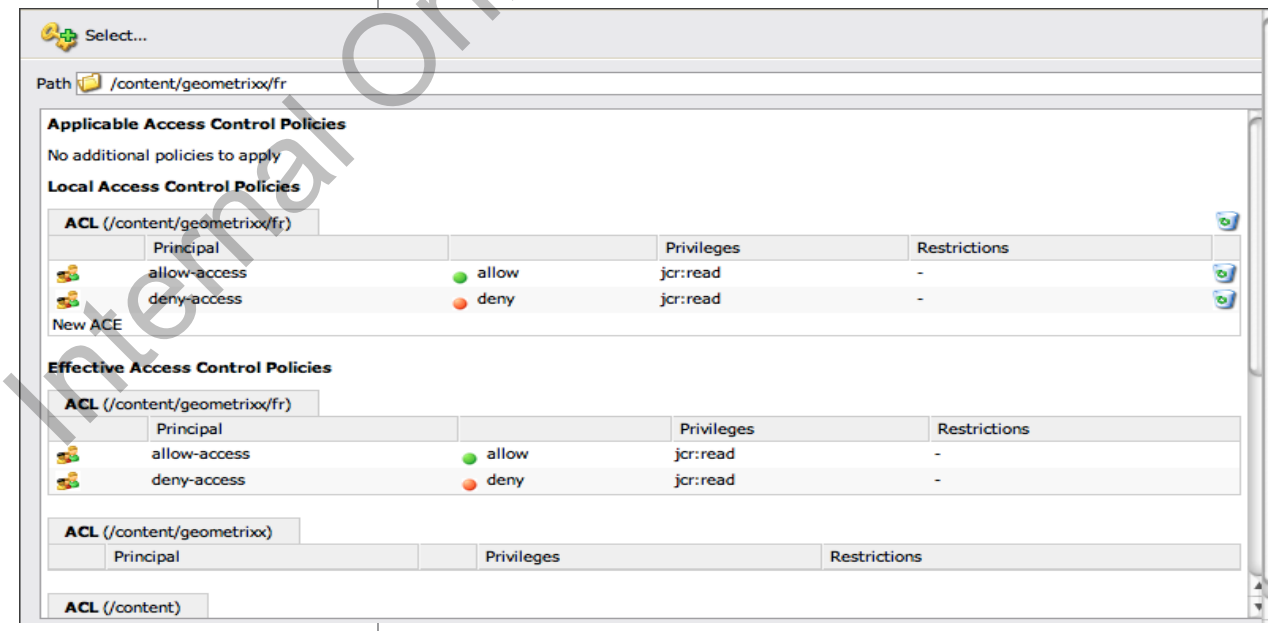
Don't forget to click on Save each time!

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:

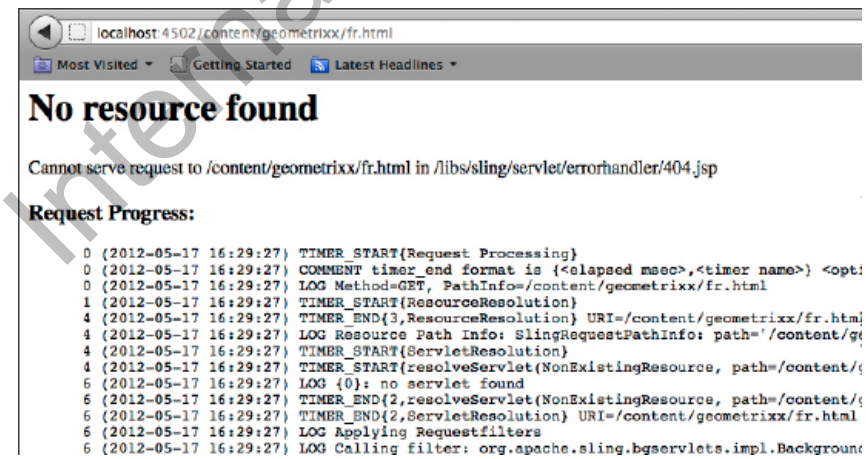


As you can see, the user John Doe doesn't have access to the French page as the deny rule is the one at the bottom of the ACL rules applied to the resource /content/geometrixx/fr and therefore takes precedence.

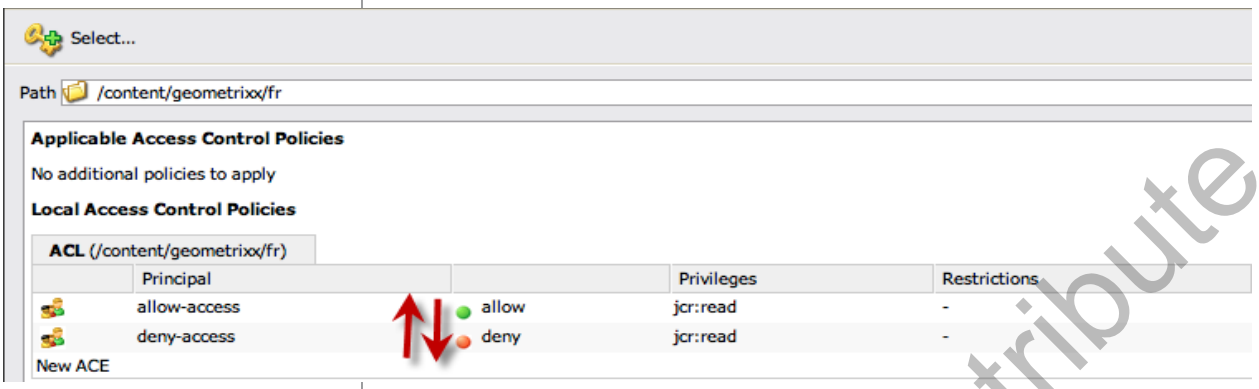
7. Open the page http://localhost:4502/content/geometrixx/fr.html. in your browser.

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

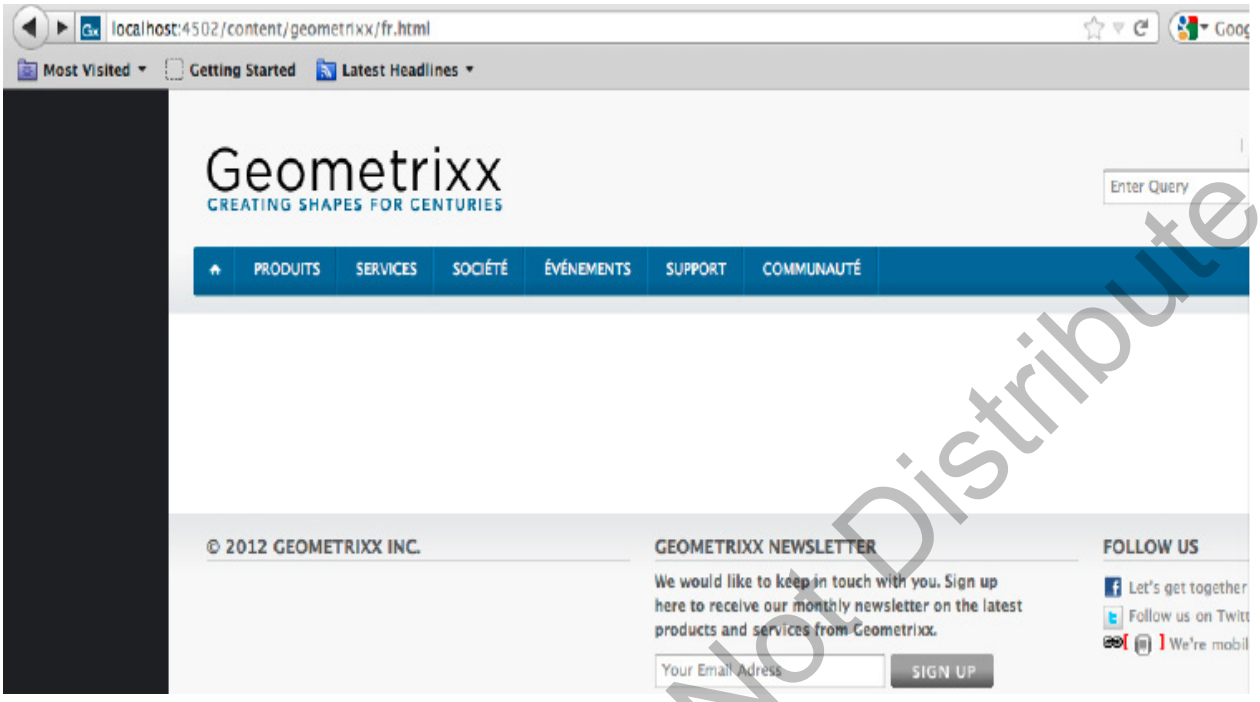


You should see the following

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. Again edit the ACLs so that the user John Doe doesn't have access to the French page.

11. Let's now write a service that grants the group 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 readily find this bundle in the Felix console. It is exported by the system bundle (bundle 0).

The service shall add the required permissions upon activation.

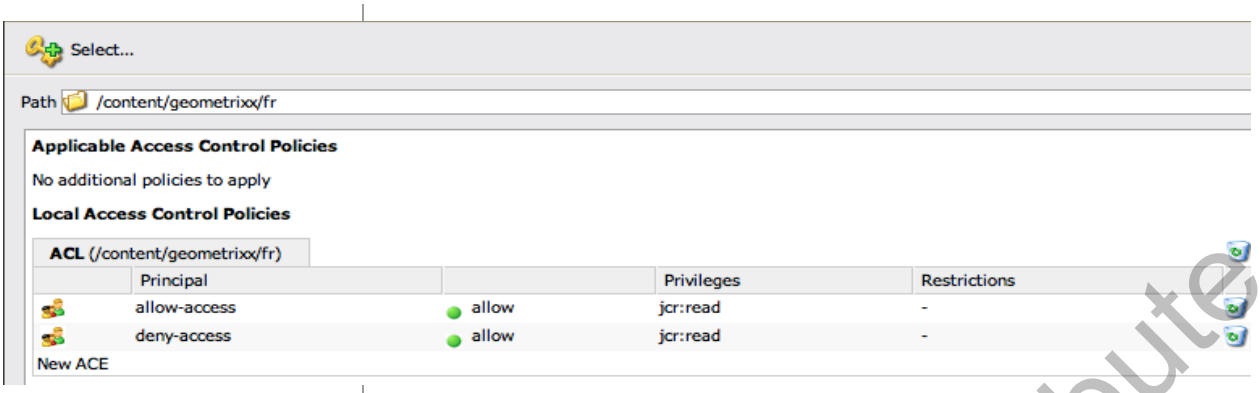
```
package com.adobe.training.core;

import java.util.NoSuchElementException;

import java.jcr.RepositoryException;
import java.jcr.Session
import java.jcr.security.AccessControlList;
import java.jcr.security.AccessControlManager;
import java.jcr.security.AccessControlPolicyIterator;
import java.jcr.securtity.Privilege;

import.org.apache.felix.scr.annotations.Activate;
import.org.apache.felix.scr.annotaions.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.apache.sling.jcr.api.SlingRepository;
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 = loggerFactory.getLogger(ModifyPermissions.class);

@Reference
private SlingRepository repository;

@Activate
protected void activate() {
     LOGGER.info(“Modify Permission activated”);
     modifyPermissions();
}
```

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

import java.util.NoSuchElementException;

import java.jcr.RepositoryException;
import java.jcr.Session
import java.jcr.security.AccessControlList;
private void modifyPermissions() {
      Session adminSession = null;
      try{
               adminSession= repo.loginAdministrative(null);

               UserManager userMgr= ((org.apache.jackrabbit.api.JackrabbitSession)adminSession).getUserManager();
               AccessControlManager accessControlManager = asminSession.getAccessControlManager();

               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)[]
               }
               Privilige[] 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();
        }
```

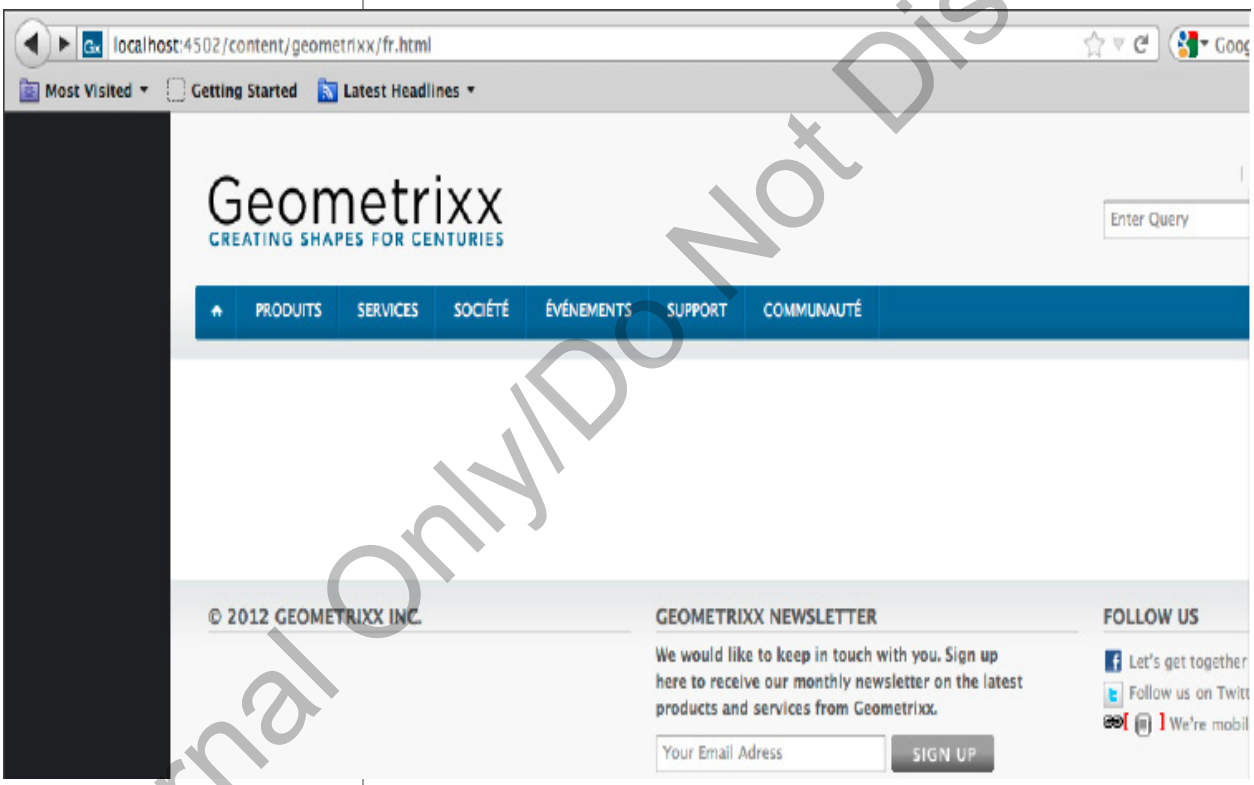
12. Deploy the component by executing:
    mvn clean install -P bundle

13. Reopen the ACL Editor and check the permissions for the page /content/geometrixx/fr

Now, you have access mode on both ACLs:



14. And of course with the user John Doe you are able again to open the French page



9 Testing (Sling and Maven)

In this chapter we are going to describe how unit tests can be conducted inside CQ.

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 Sling-based 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=MyTests Case,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) 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. If we do care about it, 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 ExampleTestclass:

```
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 API's 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 CQ 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 - Unit Tests using Junit and Maven

We will first create a very simple Testable class and then we will proceed to do some Unit tests using Junit and Maven

Steps

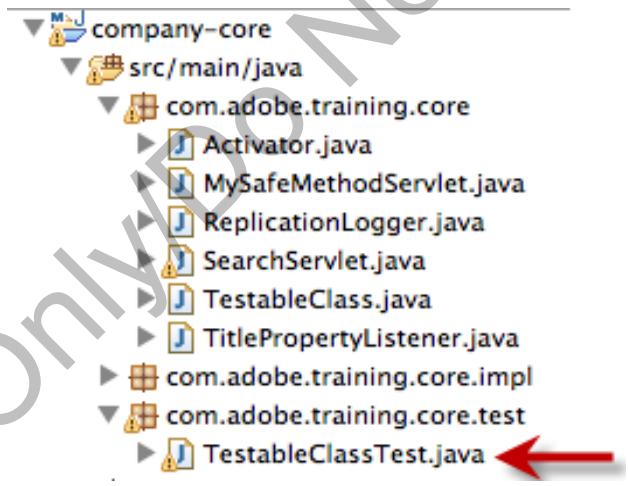
1. Create a testable class in com.adobe.training.core. It shall contain 2 methods: one that depends on JCR environment specifics and one that does not:

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

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

public class TestableClass {

    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 artefacts that we are going to use. The parent POM you received also contains those artefacts:

```
<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 TestableClass.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 like 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 TestableClassTest {

    @Test
    public void testStripNonLettersOrNumbers() {
        TestableClass tc = new TestableClass();
        assertEquals("abc1", tc.stripNonLettersOrNumbers("a_b!c.1"));
    }
```

5. Run "mvn clean test" to execute the test. You should see the following result:

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------

Running com.adobe.training.core.TestableClassTest
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(){
        TestableClass tc= new TestableClass();
        assertEquals("abbc1",tc.stripNonLettresOrNumbers("a_b!c.1"));
    }
```

7. Test again, now you should see an error

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.adobe.training.core.TestableClassTest
```

```
Tests run: 1, Failures: 1, Errors: 0, Skipped: 0, Time
elapsed: 0.058 sec <<< FAILURE!

Results :

Failed tests:
testStripNonLettersOrNumbers(com.adobe.training.core.
TestableClassTest): 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/CQ55AdvDev/
sampleproject/company-core/target/surefire-reports for the
individual test results.
[ERROR] -> [Help 1
```

Unit tests with Junit, EasyMock, PowerMock and Maven

We will now test the method TestableClass.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 use 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 TestableClass, we sill use Junit to test that our class method getContentPath(Session) retrieves the correct path ("/content")

```
assertEquals("/content", tc.getContentPath(SESSION _ MOCK));
```

We will verify those expectations with verifyAll()

1. Replace previous Testable Class test method. Let's create our method testGetContentPath()

NOTE: The call to tc.getContentPath at that line should trigger all those methods written in our expectations as well as return the expected values.

```
@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(TestableClass.CONTENT _ NODENAME)). andReturn(true);

      finalNode CONTENT _ NODE _ MOCK. = createMock(Node.class);
      expect(ROOT _ NODE _ MOCK.getNode(TestableClass.CONTENT _ NODENAME)). andReturn(CONTENT _ NODE _ MOCK);
      expect(CONTENT _ NODE _ MOCK.getPath()).andReturn("/content");
      replyAll();
      TestableClass tc = new TestableClass();
      asserEquals('/content", tc.getContentPAth(SESSION _ MOCK));
      //vewrify that all expected methods calls have been executed
      verfyAll();
```

2. Run "mvn clean test" again. You should see this time your two tests being executed successfully, and the project being built.

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------

Running com.adobe.training.core.TestableClassTest
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] ------------------------------------------------------------
----------------------------------------
```

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 CQ5 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)

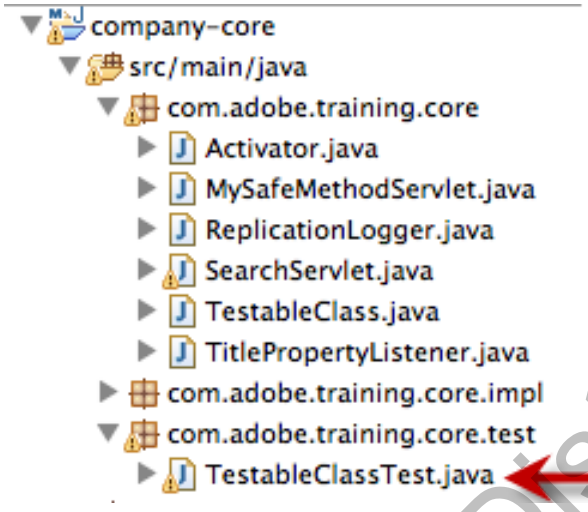
2. You should see your bundle's reference's in your parent's POM as well as in your core bundle's POM

```
        <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 in the src/main part of the project:




```
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.TestableClass;

@RunWith(SlingAnnotationsTestRunner.class)
public class TestableClassTest {
```

4. Add a method testGetcontentPath to test TestableClass.getContentPath (but running within the repository).

For that we first use the annotation @RunWith, when a class is annotated with it 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.

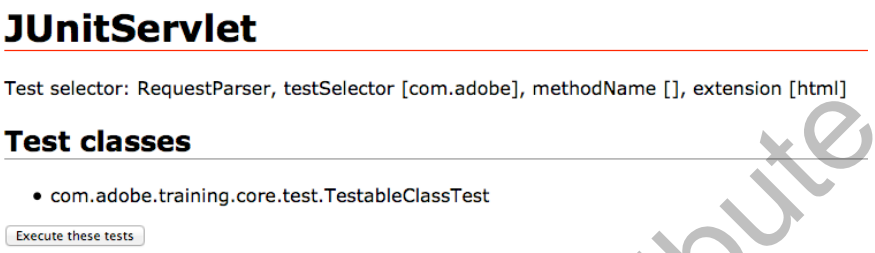
In order to have access to the SlingRepository (and be able to run our classes within it) we use @TestReference. @TestReference will inject the service of our SlingRepository in our code; it will be made available to our classes by the Service Component Runtime.

The SlingRepository extends the standard JCR repository interface with two methods: getDefaultWorkspace() and loginAdministrative(String). This method ease the use of a JCR 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 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.runner.RunWith;
import org.junit.Test;

import com.adobe.training.core.TestableClass;

@RunWith(SlingAnnotationsTestRunner.class)
public class TestableClassTest {

    @TestReference
    private SlingRepository repository;

    @Test
    public void testGetContentPath(){
        Session adminSession = null;
        try{
            adminSession =repository.loginAdministrative(null);
            TestableClass tc= new TestableClass();
            assertEquals("/content", tc.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.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 TestableClassTest {
    @TestReference
    private SlingRepository repository;

    public void testGetContentPath(){

    @Test
    public void testRepoName(){
        assertTrue(repository.getDescriptor(Repository.REP_NAME_DESC).equals("CRX"));
    }

}
```

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.html and execute the tests

JUnitServlet

Test selector: RequestParser, testSelector [com.adobe], methodName [], extension [html]

Test classes

- com.adobe.training.core.test.TestableClassTest

Execute these tests

You should see the following screen:

JUnitServlet

Running tests

com.adobe.training.core.test.TestableClassTest

Test finished: testGetContentPath(com.adobe.training.core.test.TestableClassTest)

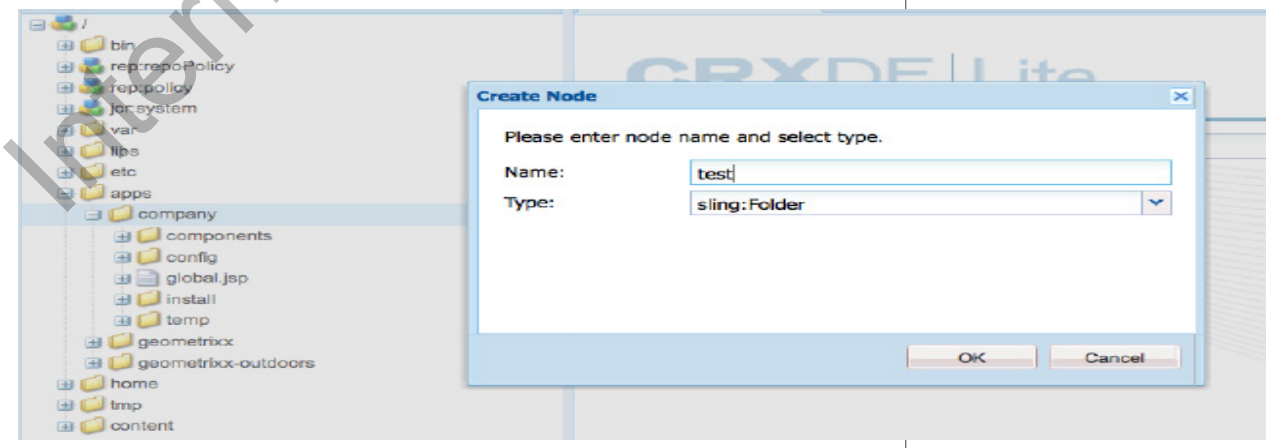
Test finished: testRepoName(com.adobe.training.core.test.TestableClassTest)

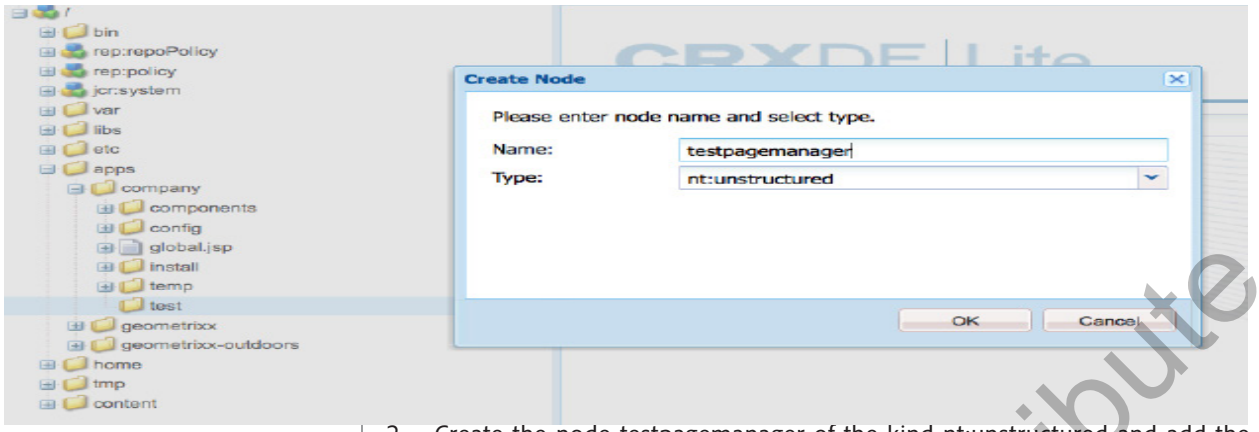
TEST RUN FINISHED: tests:2 , failures:0 , ignored:0

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. The only thing left to proceed to the tests is to ask for this resource in CQ

1. Open CRXDE Lite and add the node "tests" of the kind sling:Folder to /apps/company

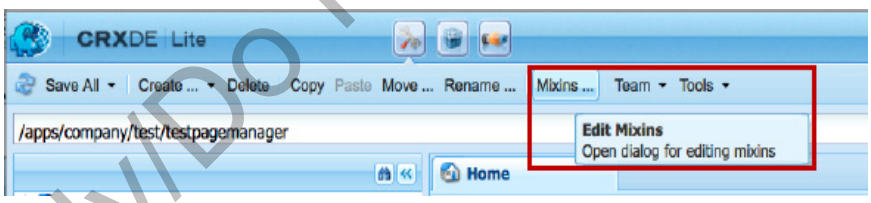


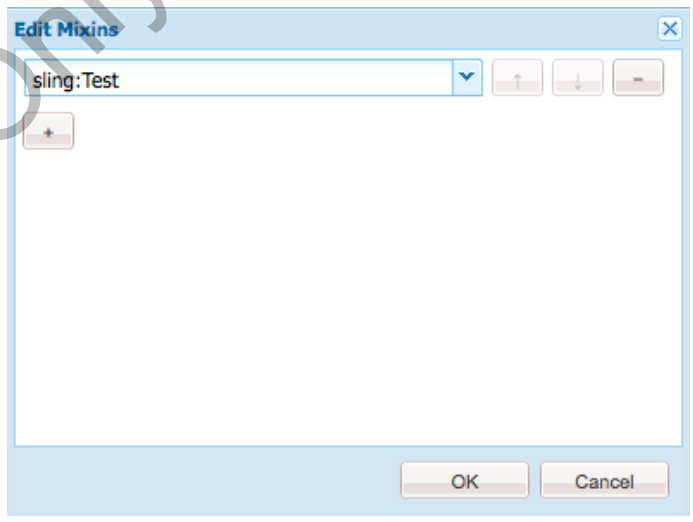


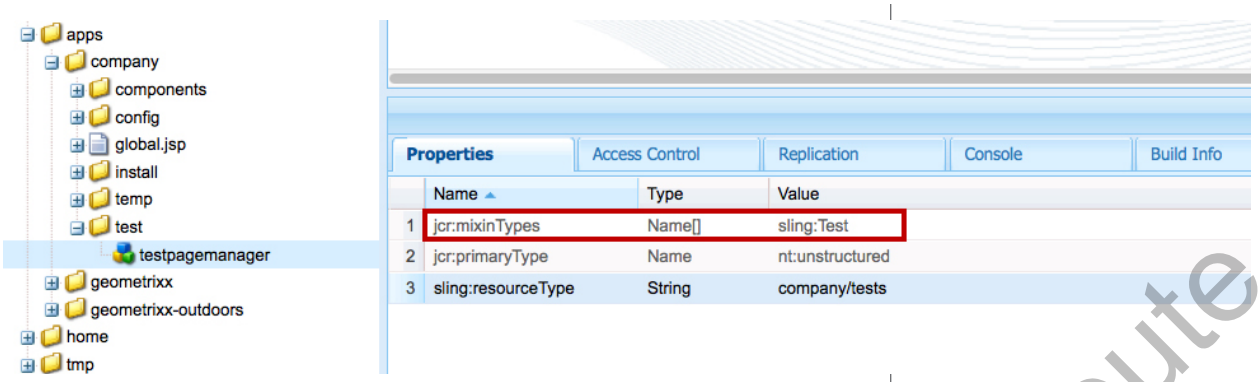
2. Create the node testpagemanager of the kind nt:unstructured and add the sling:resourceType company/tests

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

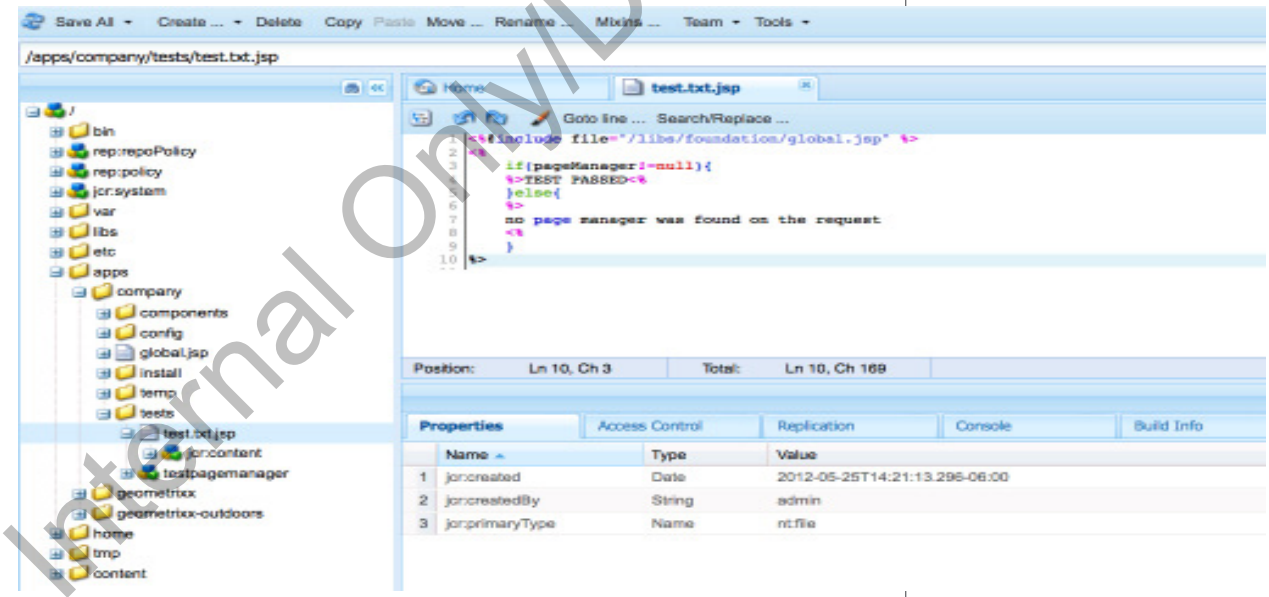




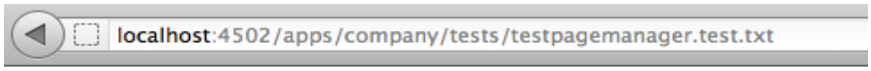


5. The test script shall test if /libs/foundation/global.jsp actually puts the PageManager object on the request.
   Create the a jsp called test.txt.jsp (notice the selector and the extension used) and add the following code:

```
<%@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:



TEST PASSED

10 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 CQ5 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.

	Name	Type	Value	Protected	Mandatory
1	jcr:created	Date	2012-05-25T11:01:47.205-06:00	true	false
2	jcr:createdBy	String	admin	true	false
3	jcr:primaryType	Name	sling:OsgiConfig	true	true
4	localMode	Boolean	true	false	false
5	timeoute	Long	20	false	false
6	url	String	http://sg1469p.corproot.net	false	false

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

- CQ 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 on the console:

```
java -Dsling.run.modes=author,foo,dev -Xmx256m -jar cq-quickstart-5.5.0.jar
```

or in the start script:

```
# default JVM options
CQ_JVM_OPTS='-Dsling.run.modes=author,foo,dev'
```

or by entries in the file crx-quickstart/conf/sling.properties

Add the line, for example:

```
Or as a Quickstart command line option:
java -jar cq-quickstart-5.5.0.jar -r author de
```

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.

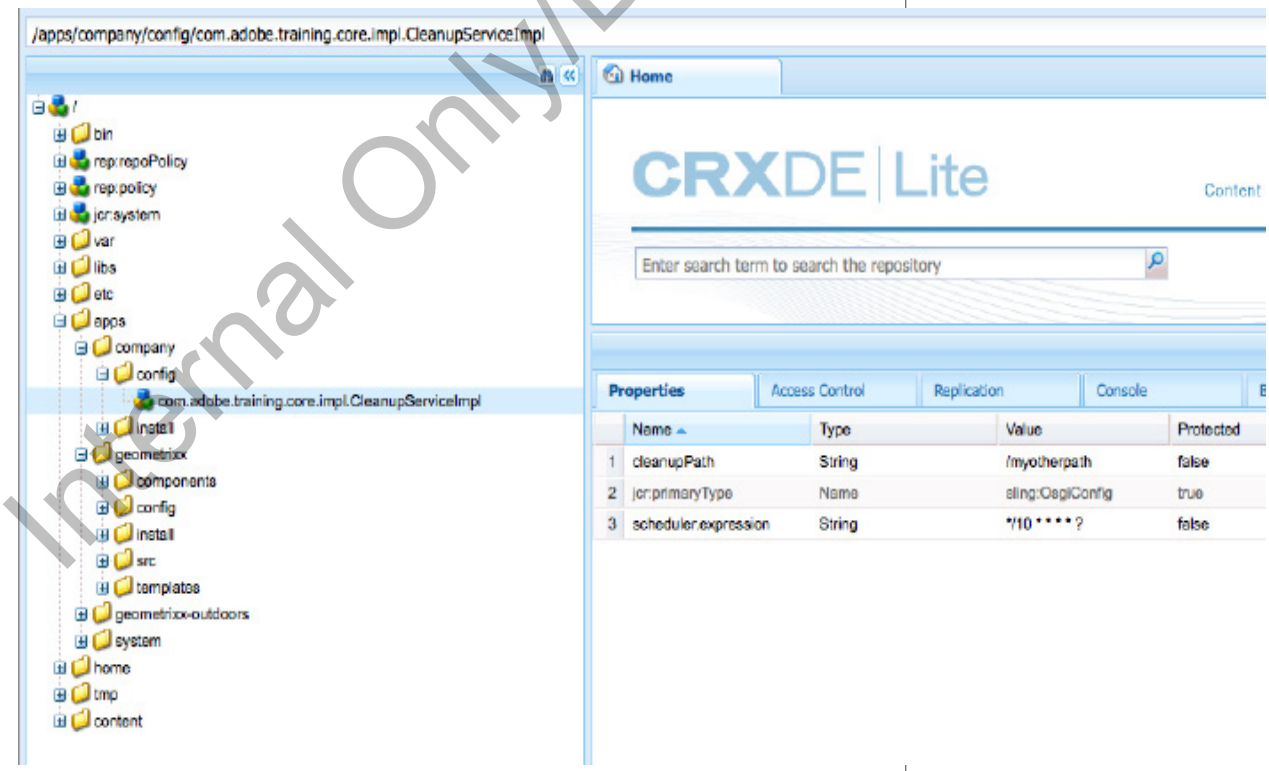
EXERCISE - Configuration Package

Goal

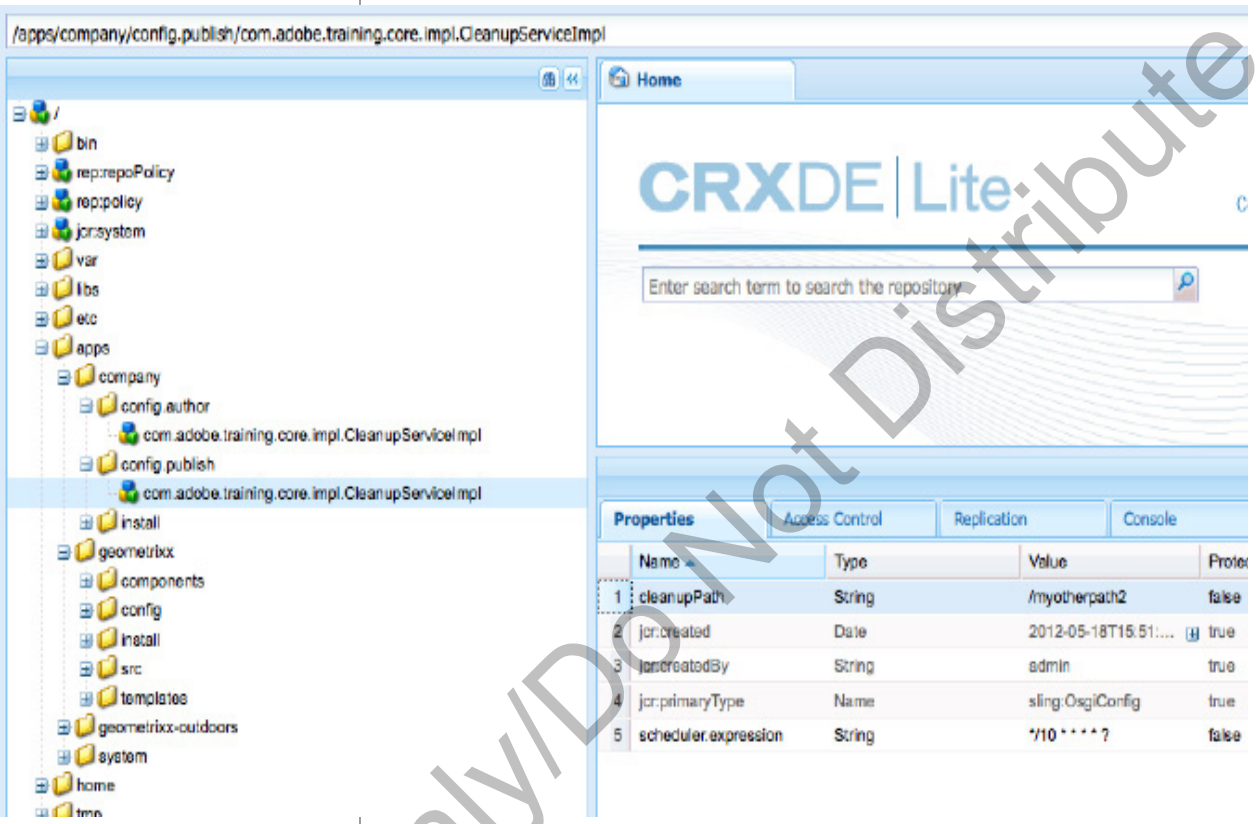
The aim of this exercise is to modify the custom configuration created earlier, and provide different settings on author and on publish instances. We will create a package, activate it, and check the results. If time allows you can also checkout the configuration into a new config project.

Steps

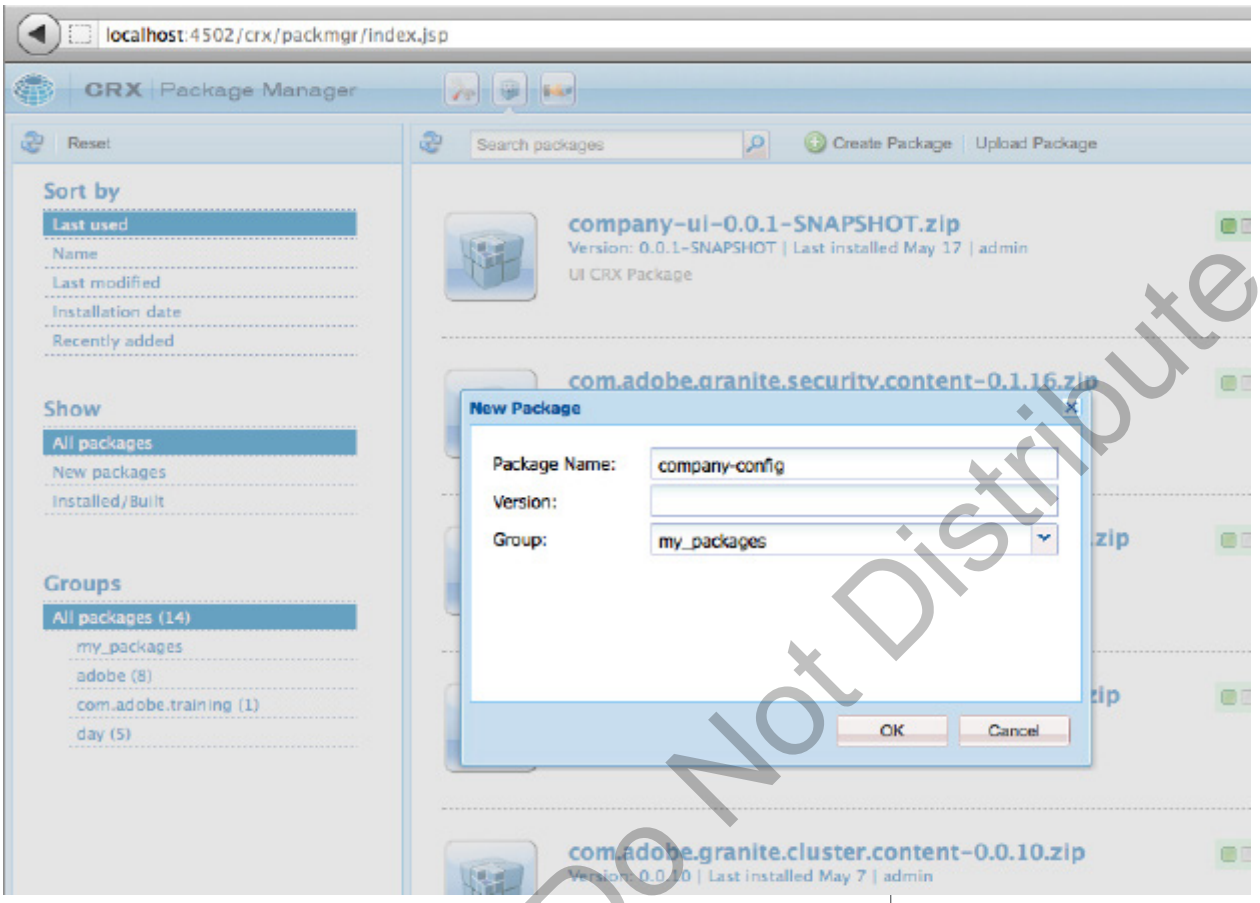
1. Start by reexamining the setting we created earlier under the node `config`:



2. Copy the `config` folder, and paste a new copy into the same location, then adjust the original and new folder names to `config.author` and `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 cleanuo Path property has changed:



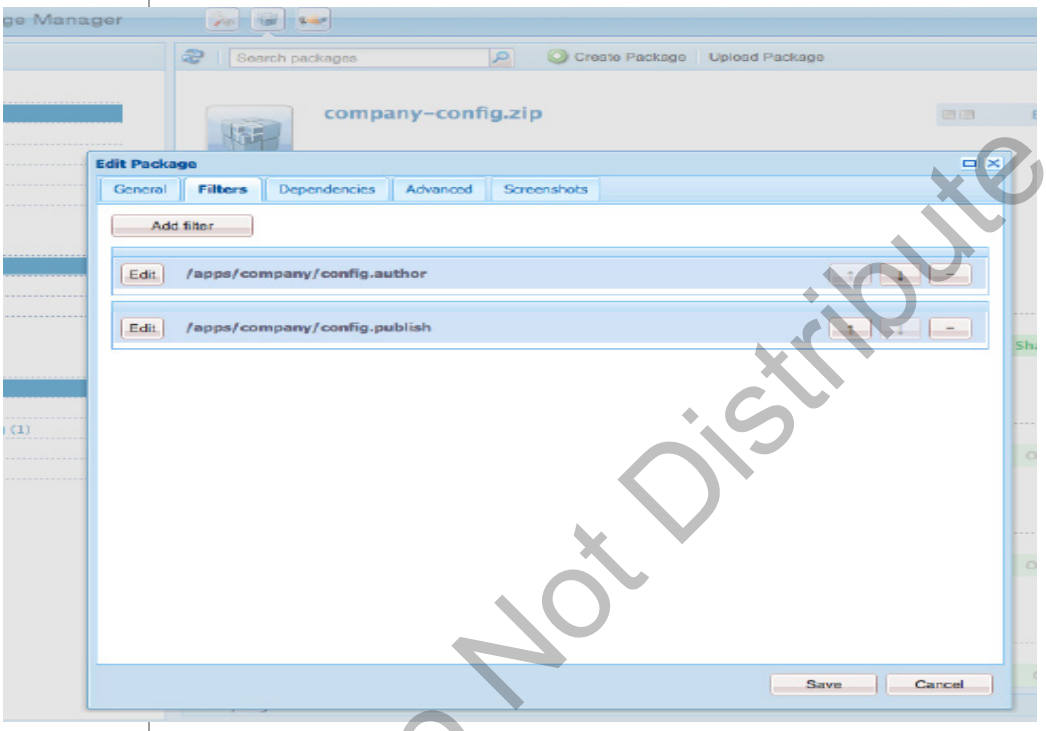
3. Create a package containing the `config.author` and `config.publish` folders. Go to http://localhost:4502/crx/packmgr/index.jsp and click on `create package`:



4. Add the package name company-config, and group name `my _ packages`, then click on OK.

5. Open the package, click on the Edit button, and add the two filters `/apps/company/config.author` and `apps/company/config.publish`:

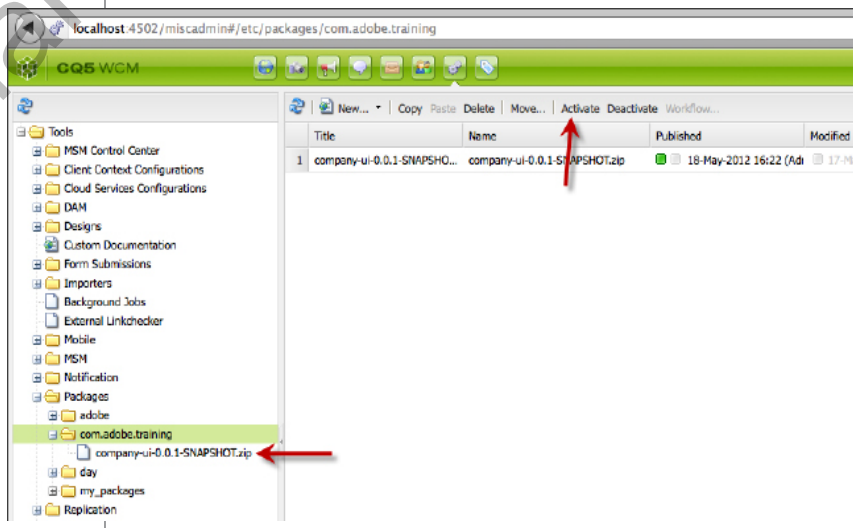
6. Build the package.



7. 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`

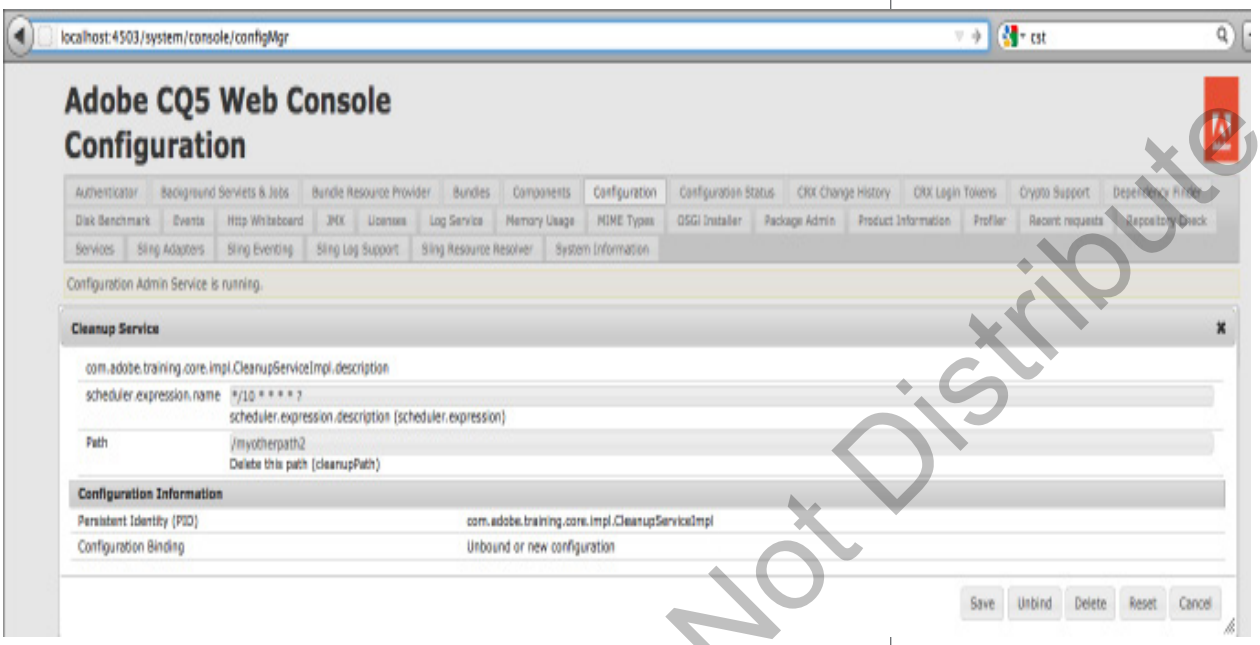
NOTE: You must activate the application package first and then the configuration package.

8. 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 CQ5 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.

11 Dispatcher, Reverse Replication

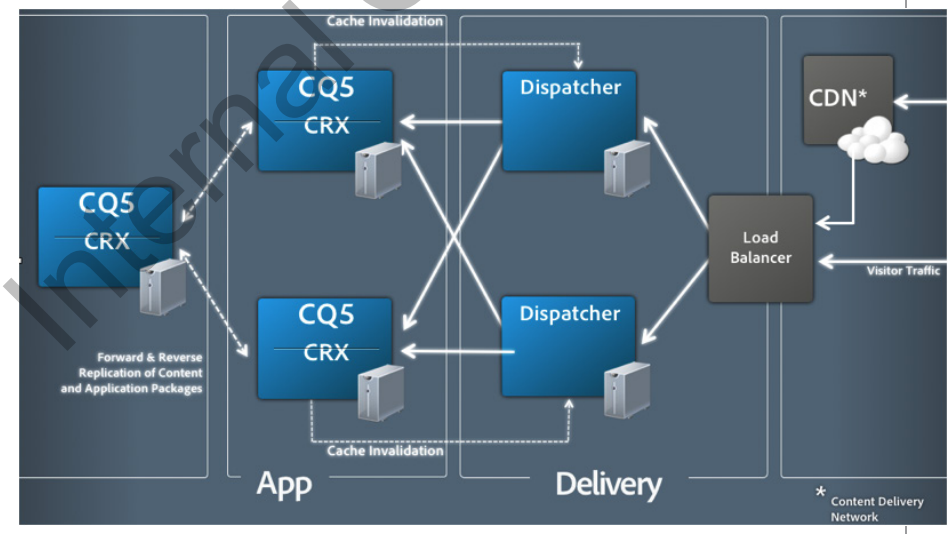
Dispatcher

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 CQ 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.



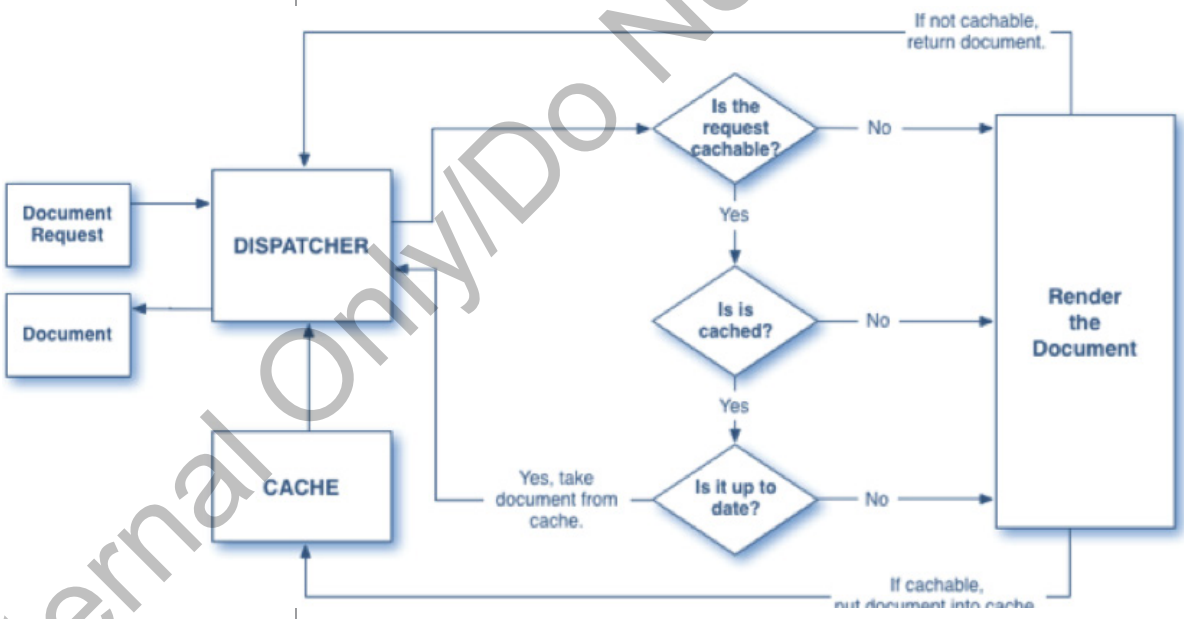
The Basics Revisited

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 CQ to update the cache when the content in CQ 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 CQ5 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 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 the German translation directly, you may encounter the following problem:

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 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 CQ (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

In order to find server-side execution bottlenecks in components, remember the **Foundation timing component** from the CQ Basic Developer training. The `<cq:include>` of the timing component:

```
<!--
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
-->
```

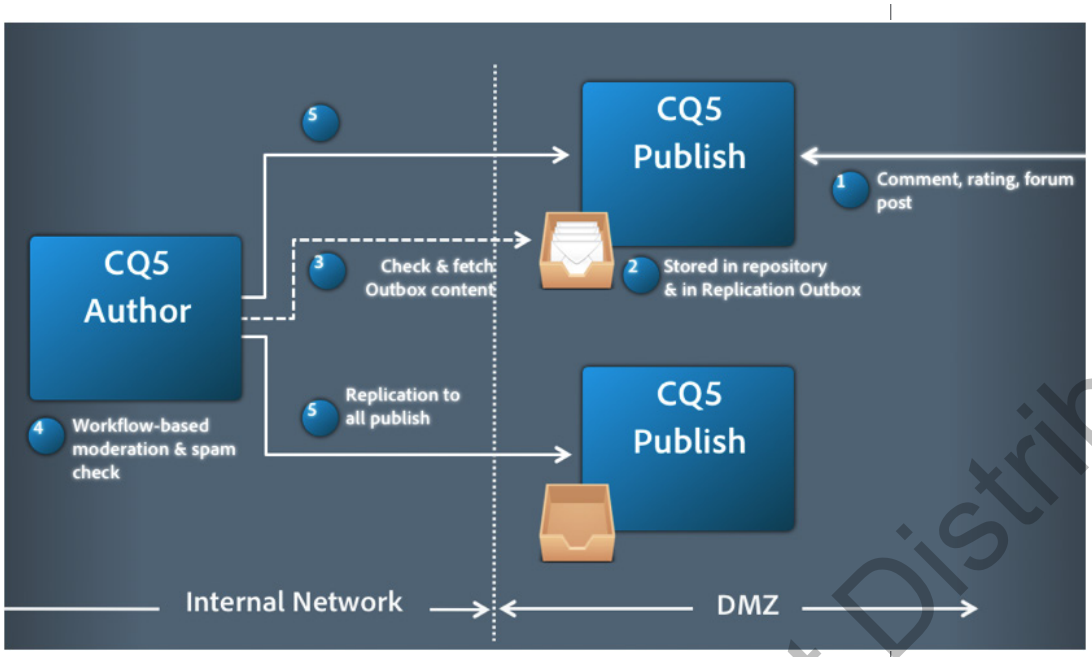
```
<cq:include path="timing" resourceType="foundation/components/timing"/>
```

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.

Programmatically Triggering Reverse Replication

Content created on publish is not automatically reverse-replicated. Out of the box, reverse replication only works for CQ'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

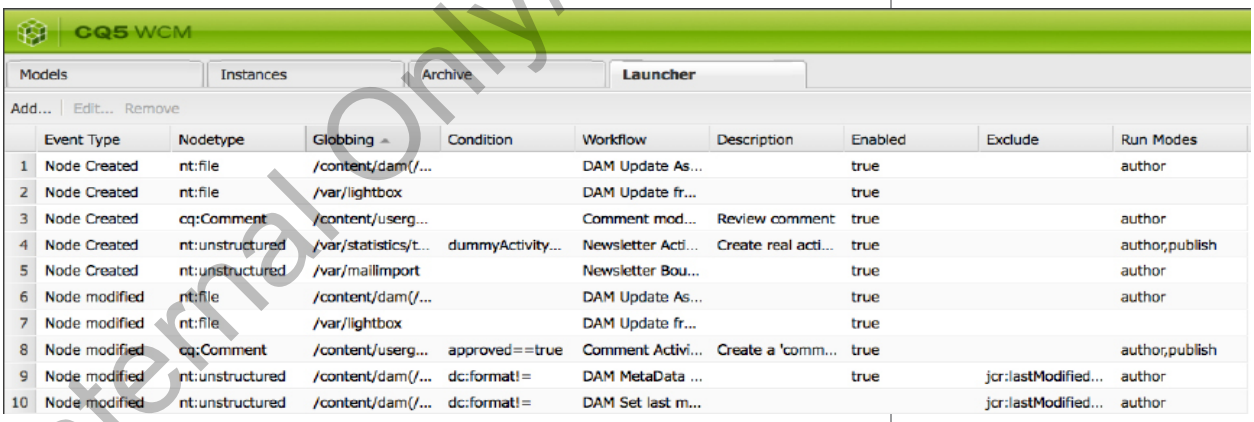
12 Content Automation, Periodic Importers

Event-based Workflow Launchers, what you learned already:

In the CQ 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 CQ. 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.

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 CQ5 WCM (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 - CQ5 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);
     }
```

NOTE: For the purposes of this exercise we have hard coded the beginning of the base URL. Typically, you would enter the full URL in the feed URL property or make this an OSGi config.

```
      catch (RepositoryException e) {
        LOGGER.error("RepositoryException", e);
      }

   }
```

4. The helper function to persist the last trade data:

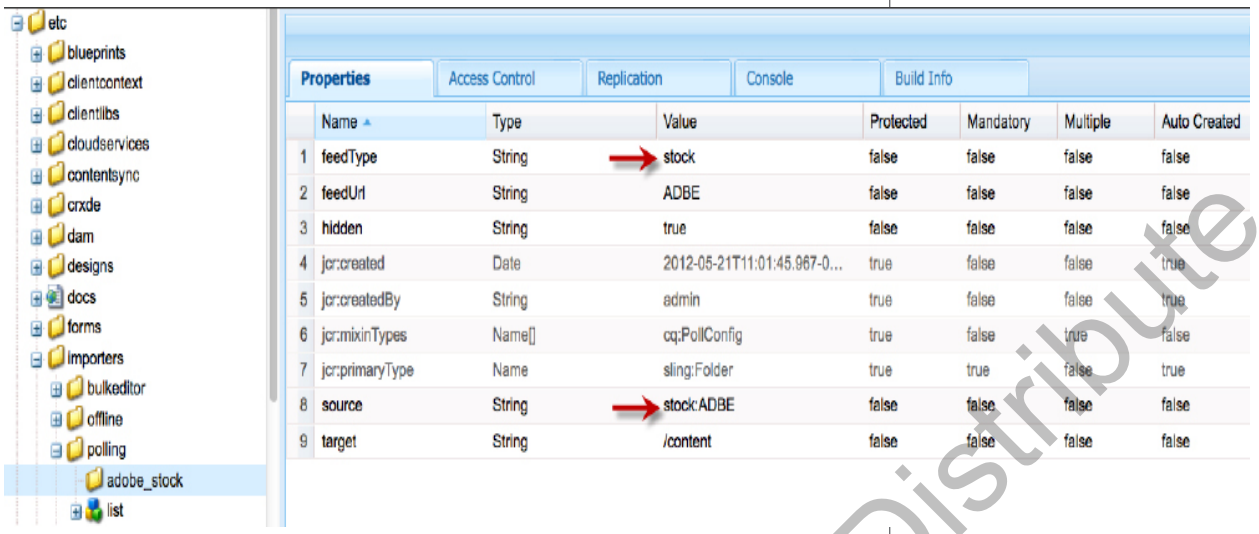
```
private void writeToRepository(final String stockSymbol,
       final String lastTrade, final Resource resource)
       throws RepositoryException {
     Node parent = resource.adaptTo(Node.class);
     Node stockPageNode = JcrUtil.createPath(parent.getPath()
       + "/" + stockSymbol, "cq:Page", 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.

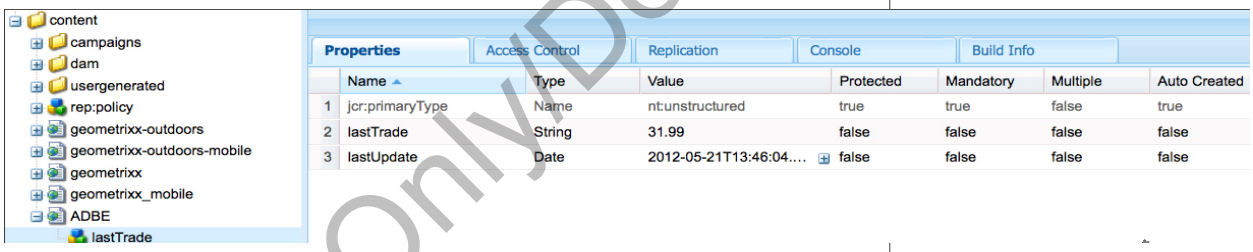
5. Deploy the core bundle and check that the component is installed in the Felix console.

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>
```

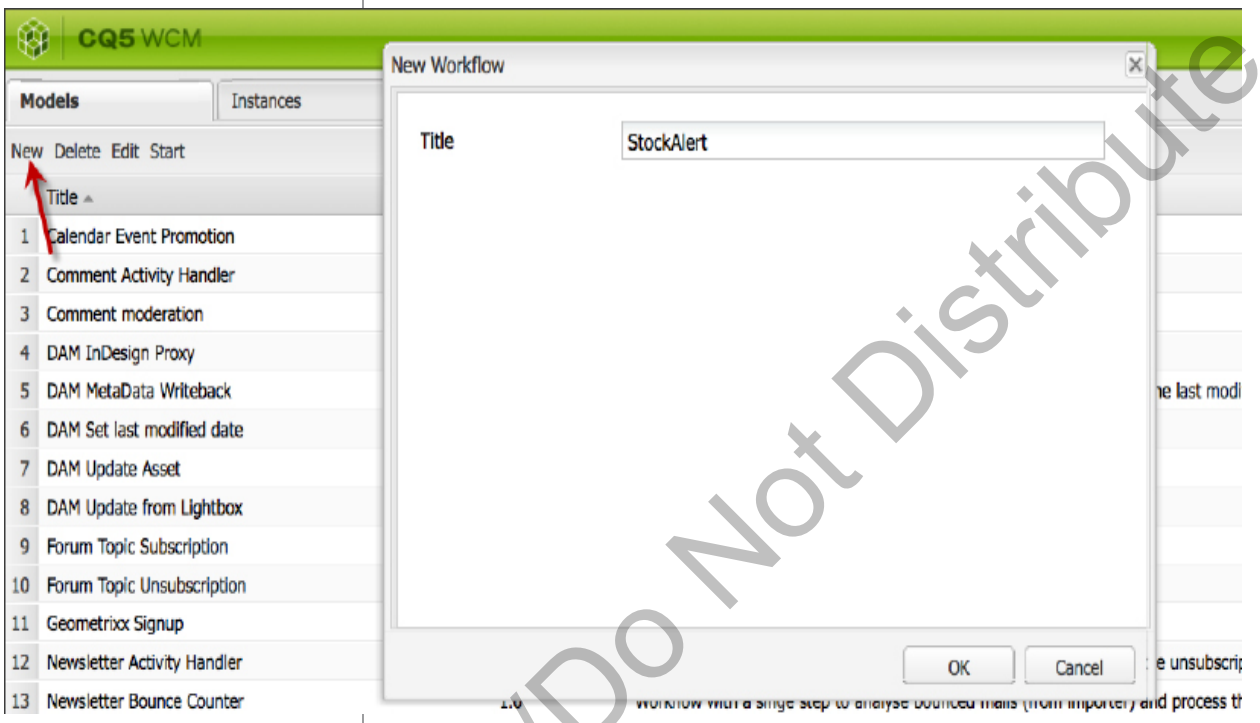
10. Create the automated workflow process, StockAlertProcess that checks a workflow item (assuming that is an imported stock data) against a list of alert thresholds and log if the threshold is exceeded:

```
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();
```

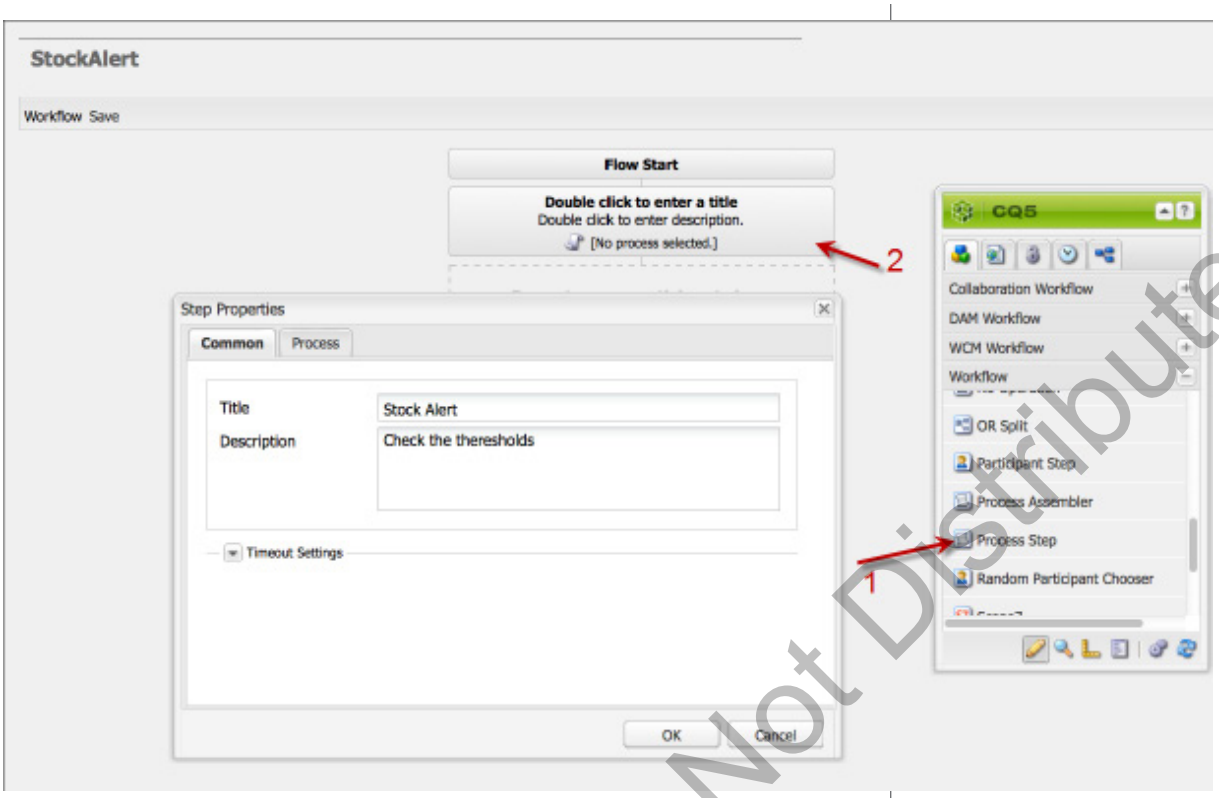
```
        Node node = null;
        String type = data.getPayloadType();
        if(type.equals(TYPE _ JCR _ PATH) &&
                    data.getPayload() != null) {
              String payloadData = (String) data.getPayload();
              if(session.itemExists(payloadData)) {
                    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!  {}  is
over {}",
                                symbol, currentLimit);
                    }
              }
        }
      }
      catch (RepositoryException e) {
        LOGGER.error("RepositoryException", e);
      }
   }
}
```

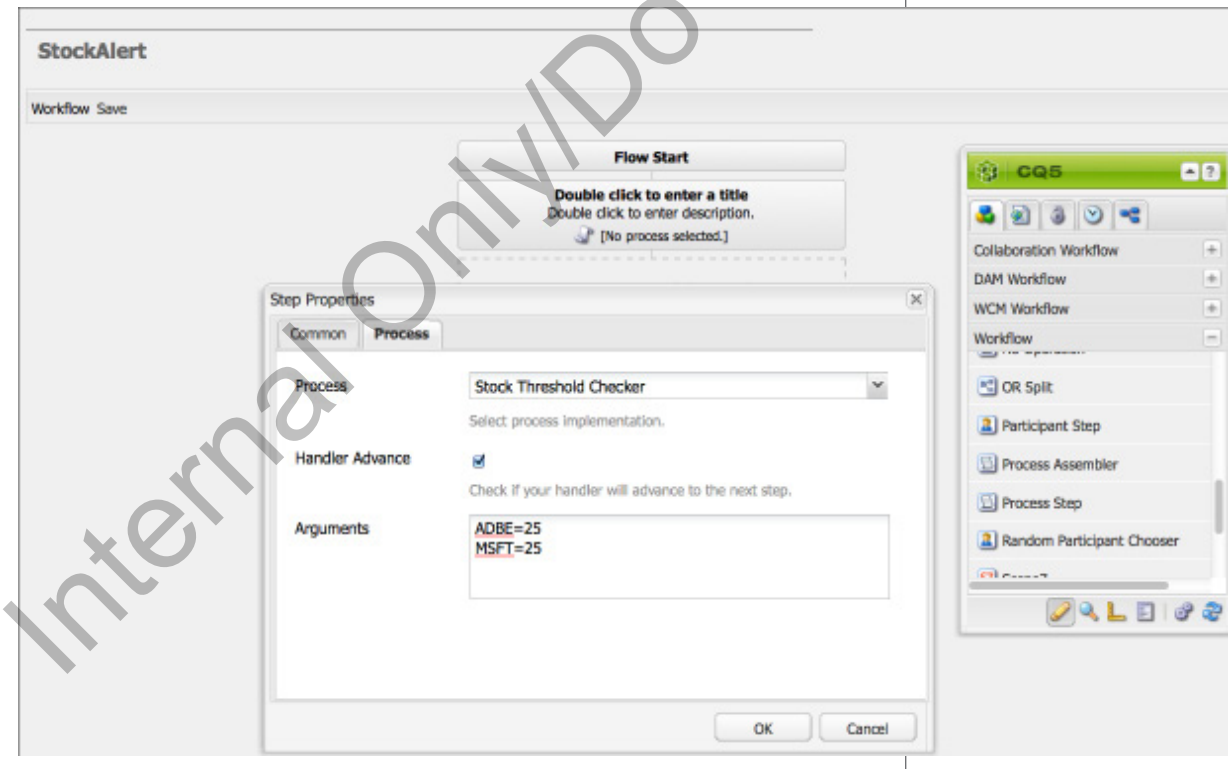
11. Deploy the workflow process via mvn.

12. In the workflow admin UI (linked from CQ5'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:

13 Client Libraries

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 Adobe CQ 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

Adobe CQ 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".

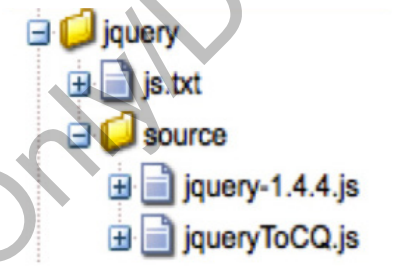
Resources, like images or fonts, that are used by JS or CSS files, are stored in the client library folder.

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

Adobe CQ 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).

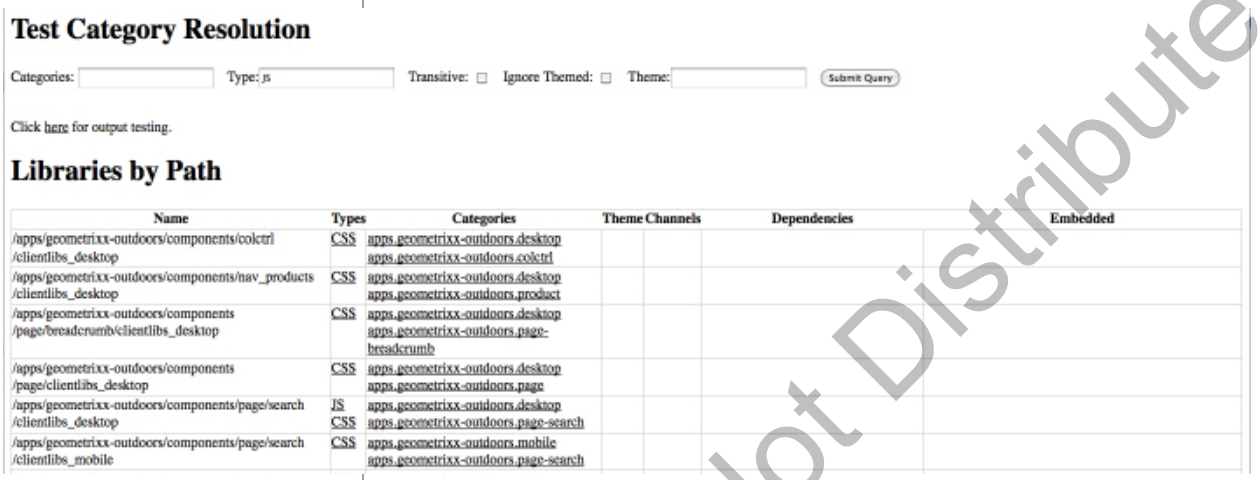
Examples of <cq:includeClientLib> Tag

```
<%-- 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

Adobe CQ 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 CQ 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 - Client Libraries (global)

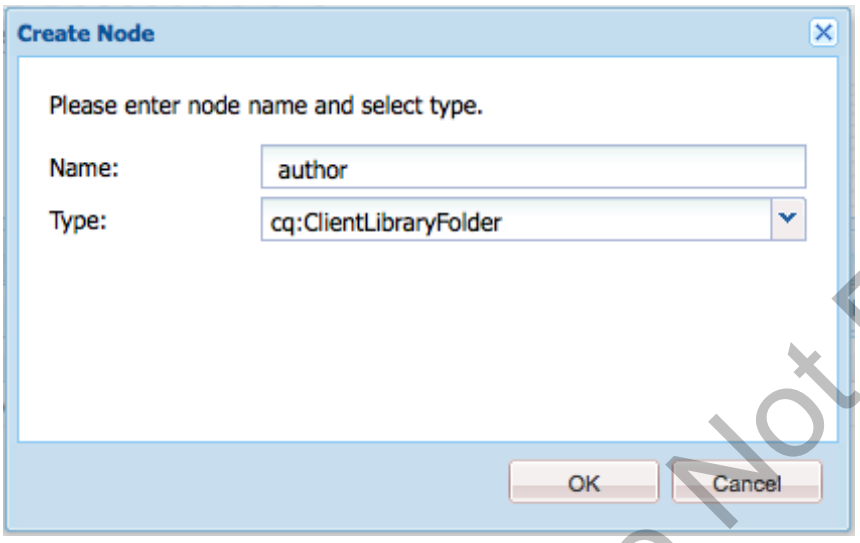
Goal

- Manage all client libraries from one place (/etc/clientlibs)
- Render categories individually for author and publish instances

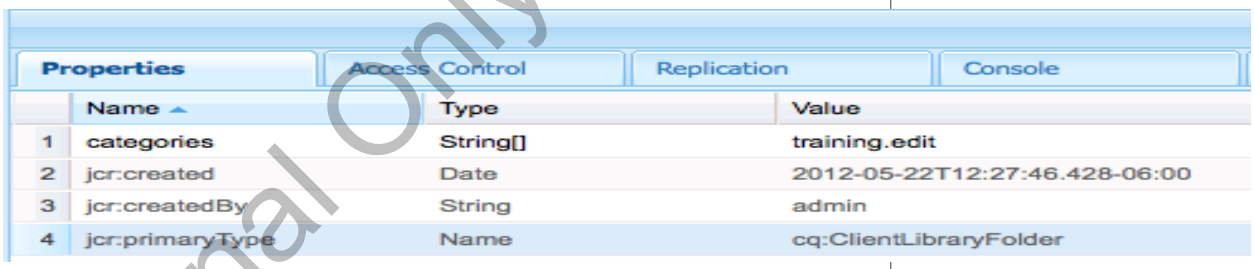
Steps

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".

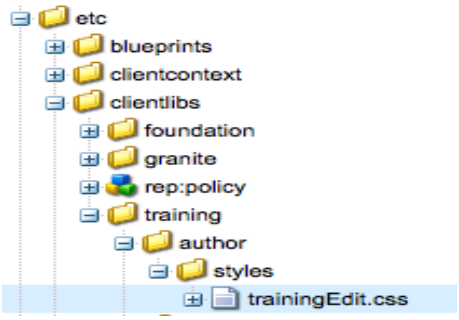


4. Repeat the steps 2 and 3 with a client library folder name "publish". Set the categories property for the publish folder to "training".

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.

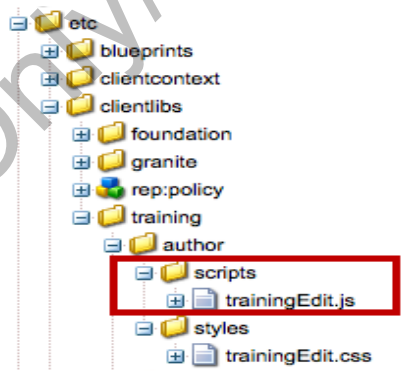
```
/* Training CSS File - Author */
#authorData { color: blue; }
```

9. Repeat for the publish client library. In /etc/clientlibs/training/pulbish 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 /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.

```
alert("This is an alert in Author mode");
```

13. Repeat for the publish client library. In /etc/clientlibs/training/pulbish create a new folder "scripts", add a new JavaScript file training.js and add a new snippet, e.g.

14. Since Adobe CQ 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.

```
/* Training JS File */
alert("This is an alert in Publish mode");
```

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.

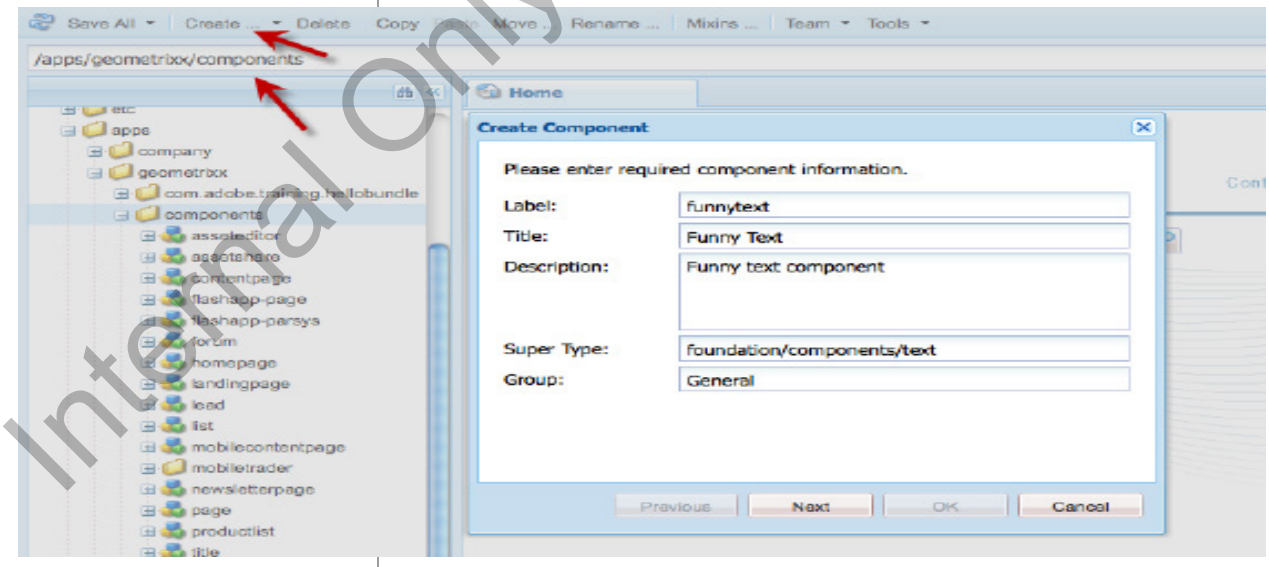
```
#base=scripts
trainingEdit.js
```

20. And repeat the same for the publish client library

```
#base=scripts
training.js
```

21. The client libraries are ready. To include them into an HTML page use the JSP Tag <cq:includeClientLib/>. You could use code. Similar to that shown below, in your page rendering component:

```
<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"/>
    </c:if>
  </head>
  <body>
    <p id="publishData">Publish-mode Data</p>
    <p id="authorData">Author-mode Data</p>
  </body>
</html>
```



EXERCISE - Client Libraries (below Components)

Goal

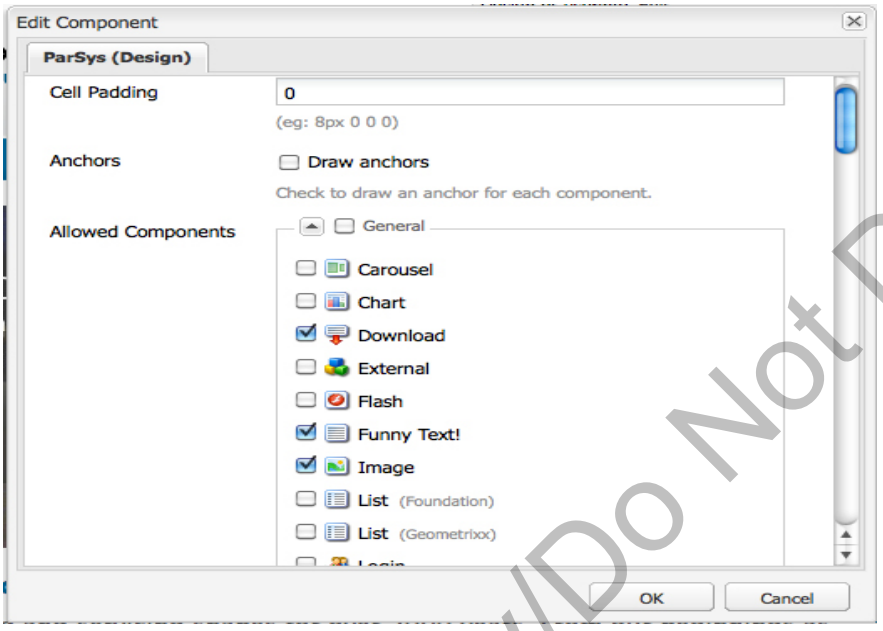
- Create a new Client Library in Geometrixx
- Make sure it gets embedded in rendered HTML pages

Steps

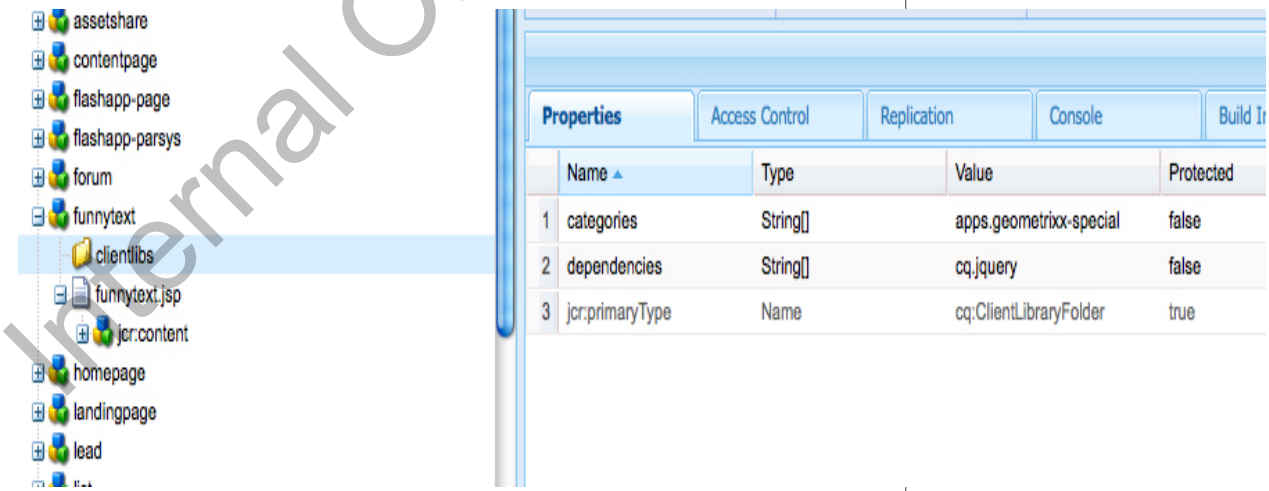
1. In CRXDE create a new component in the Geomterixx app that inherits from the foundation text component

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

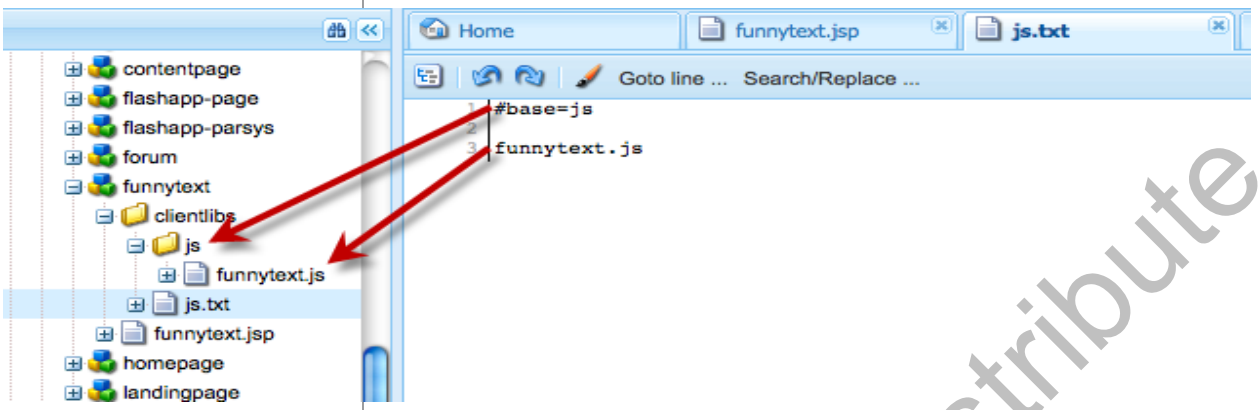
```
<%--

  Funny Text component.

  Funny text component

--%><%
%><%@include file="/libs/foundation/global.jsp"%><%
%><span class="funny"><cq:text property="text"/></span>
```



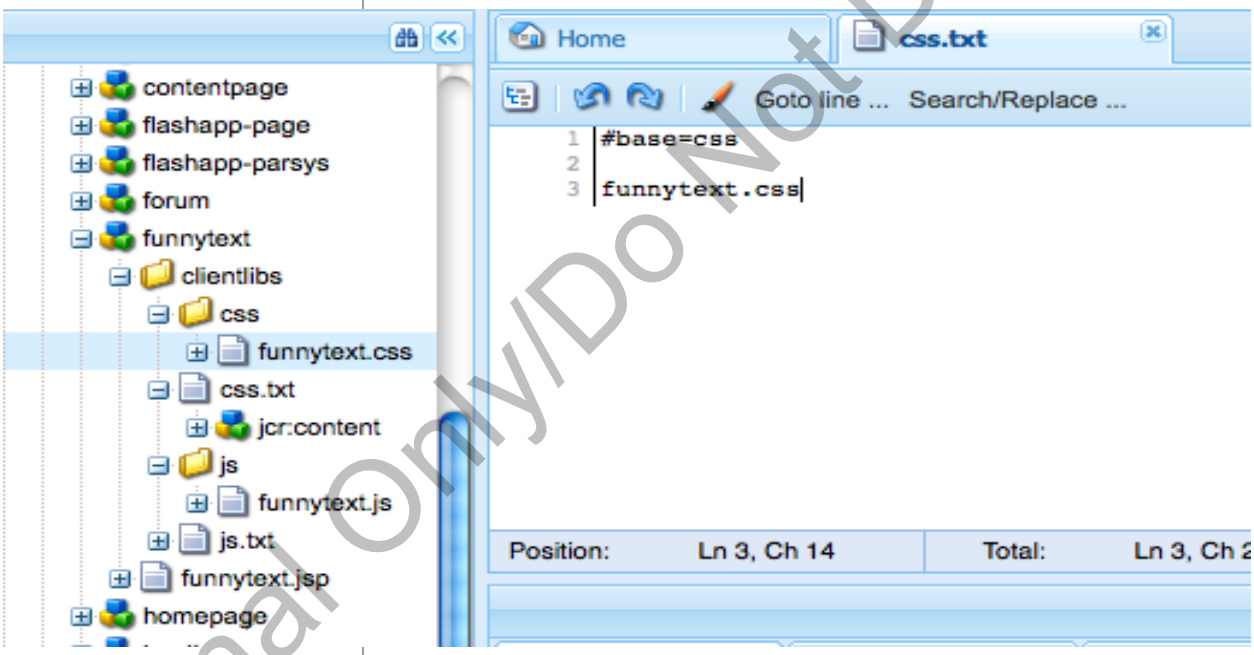
3. Create a clientlibs node (note the resource type!). Give it a new category and let it depend on the ootb jQuery:



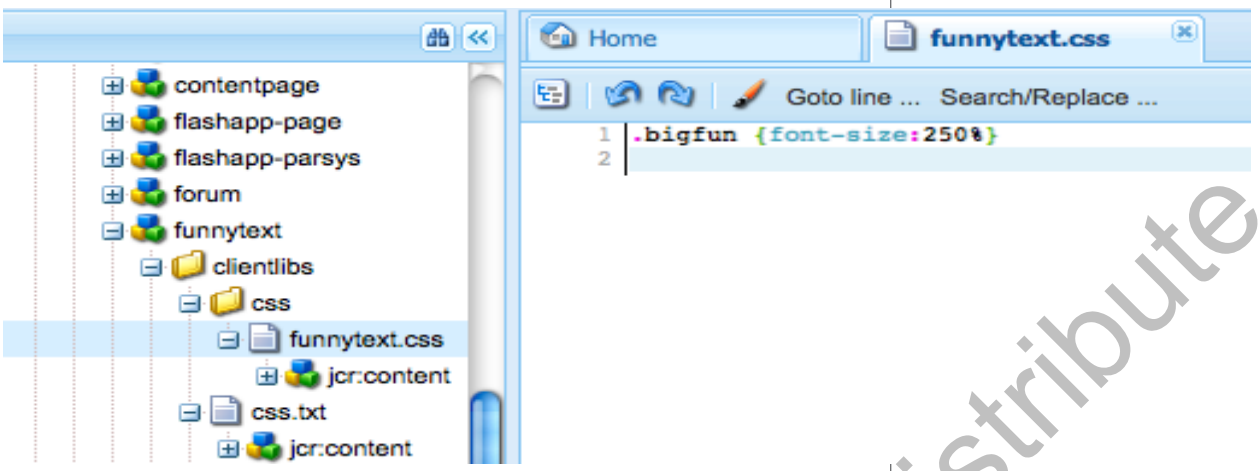
4. Create a js.txt file that references all relevant js files for this component and create the file funnytext.js to hold the actual js code



5. Analogous for css:



6. In funnytext.css add a css class:

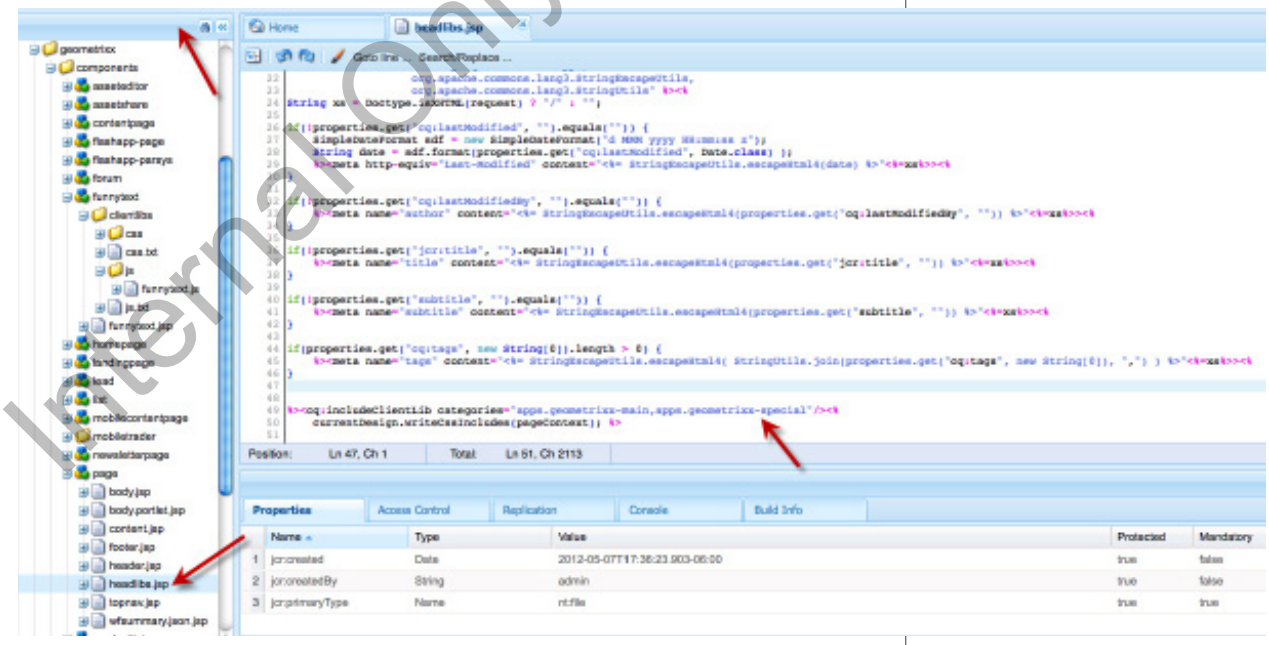


7. Edit funnytext.js and add a jQuery effect to the rendered text:

```
jQuery(function ($){
    $('.funny').hover(function() {
    console.log(this);
        $('.funny p').addClass("bigfun");
            }, function() {
        $('.funny p').removeClass("bigfun");
    });
});
```

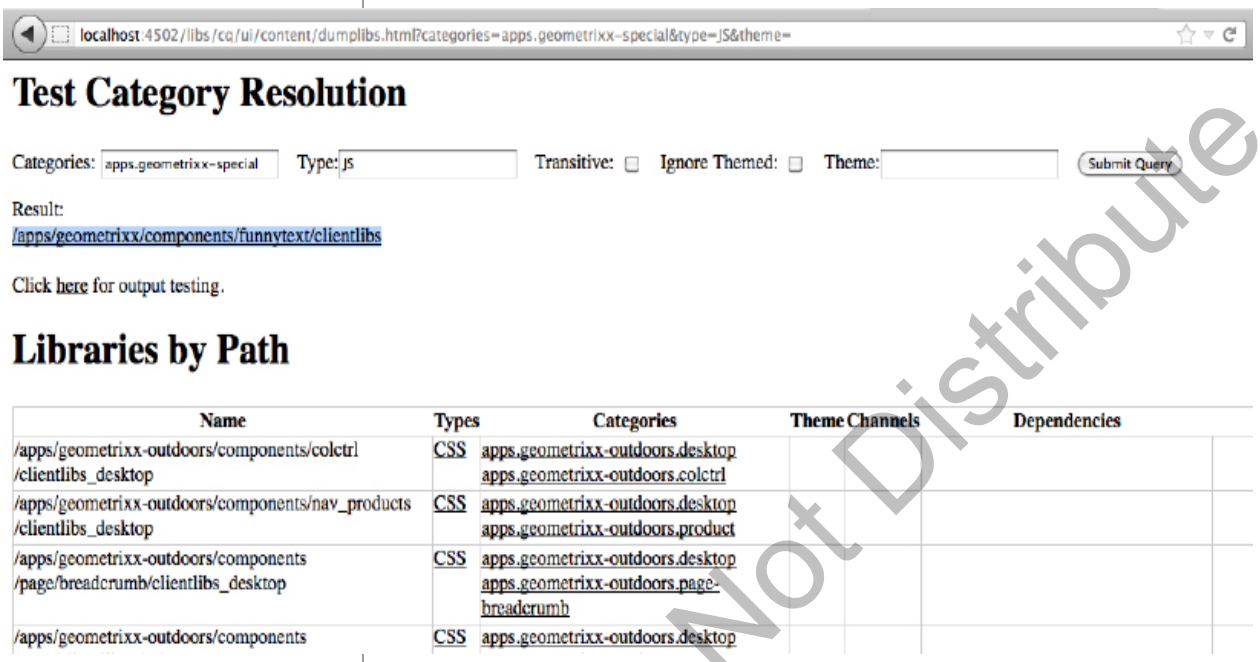
(remember that you added a dependency to jQuery so you can assume it to be defined)

8. Edit the Geometrixx jsp where the ClientLibs are included and add your new category:



9. Add the component to a Geometrixx page and verify the rollover effect.

10. Verify that your Clientlib appears in the clientlib console:



14 Coding CQ components

Separate Logic from Markup

One of the powerful concepts of Adobe CQ is that its modular concept allows authors to select from unlimited components to populate distinct areas of a Web page. Components are a combination of user interfaces (to add content), scripts (to process the content) and client libraries (JS/CSS to layout the content).

It's recommended practice to separate logic from markup. Such a separation not only allows teams to work on layout and functionality in parallel, but in a future stage a redesign can be made without changing the functionality or migrate content.

Another recommendation is that the JSP should only be used to render the HTML response. Functionality should be put in OSGi managed bundles or services.

A popular approach when creating functionality is to encapsulate the functionality in tag libraries or Java beans. Tags can be used in component scripts without having to worry about implementation details. A good example is the <cq:includeClientLib> tag. This tag writes the link tags for JS and CSS files. The attributes allow to include JS files only, CSS files only or all files.

Tag Libraries, JSTL and Beans

Adobe CQ provides OOTB tag libraries in addition to the JavaServer Pages Standard Tag Library.

JavaServer Pages Standard Tag Library (JSTL)

The JavaServer Pages Standard Tag Library (JSTL) encapsulates the core functionality common to many Web applications in simple tags. JSTL has support

for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags.
JSTL 1.2 is exposed as five custom tag libraries:

Description	Prefix	URI
Core Library	c	http://java.sun.com/jsp/jstl/core
I18N & Formatting	fmt	http://java.sun.com/jsp/jstl/fmt
Functions	fn	http://java.sun.com/jsp/jstl/functions

Description	Prefix	URI
XML Processing	xml	http://java.sun.com/jsp/jstl/xml
Database Access	sql	http://java.sun.com/jsp/jstl/sql

By default CQ declares the three common JSTL tag libraries in /libs/foundation/global.jsp:

```
%><%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %><%
%><%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %><%
%><%@taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %><%
```

Adobe CQ specific Tag Libraries

Adobe CQ has two specific tag libraries:

Description	Prefix	URI
CQ Tag Library	cq	http://www.day.com/taglibs/cq/1.0
Sling Tag Library	sling	http://sling.apache.org/taglibs/sling/1.0

The <cq:defineObjects> tag is a good example on how to encapsulate the model functionality from the view. The tag instantiates a couple of useful objects (resource, resourceResolver, currentPage, currentStyle, etc.) so that they are immediately available as Java objects or as Expression Language.

The file /libs/foundation/global.jsp is included in most every JSP script and declares the tag libraries plus instantiates commonly used Java objects with the <cq:defineObjects> tag.

```
--%><%@page session="false" import="javax.jcr.*,
        org.apache.sling.api.resource.Resource,
        org.apache.sling.api.resource.ValueMap,
        com.day.cq.commons.inherit.InheritanceValueMap,
        com.day.cq.wcm.commons.WCMUtils,
        com.day.cq.wcm.api.Page,
        com.day.cq.wcm.api.NameConstants,
        com.day.cq.wcm.api.PageManager,
        com.day.cq.wcm.api.designer.Designer,
        com.day.cq.wcm.api.designer.Design,
        com.day.cq.wcm.api.designer.Style,
        com.day.cq.wcm.api.components.ComponentContext,
        com.day.cq.wcm.api.components.EditContext"
%><%@taglib prefix="sling" uri="http://sling.apache.org/taglibs/sling/1.0" %><%
%><%@taglib prefix="cq" uri="http://www.day.com/taglibs/cq/1.0" %><%
%><%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %><%
%><%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %><%
%><%@taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %><%
%><cq:defineObjects /><%

    // add more initialization code here

%>
```

Custom Tag Libraries

In addition to the available JSTL and CQ/Sling tag libraries it's recommended to encapsulate functionality in custom tag libraries. One of the big advantages is that functionality changes happen on the tag library base, so the model, and not on the JSPs consuming the functionality.

An example: Imagine that in /etc/glossary there is a page/node for each glossary, consisting of a glossary title and name. In the "view" a glossary is assigned to one or more words by using an anchor tag with a ref attribute that triggers a tool tip with the glossary information. A good design practice is to use a custom tag, e.g. <util:glossary name="glossary_term">word</util:glossary>. The tag functionality then will search for the glossary, wrap the words within the anchor tag and prepare the tool tip. If e.g. the background color of the tool tip changes, that's done in the tag functionality, so in one place. The JSP pages remain untouched.

As for any other servlet engine tag libraries are created by defining the tag handler (the tag's Java class) and the tag library definition (the "tag library"). If the tag library is defined as OSGi bundle, changes can be deployed dynamically.

Creating Custom Tag Libraries in CRXDE

With CRXDE it's possible to create a custom tag library as an OSGi bundle.

Java Beans and Expression Language

For simple functionality that can be provided as "getter" and "setter" methods it's possible to also use Java Beans. Bean classes consist of methods starting with "get" (e.g. getMessage()), "set" or "is".

Once the bean is declared, the methods can be used with the Expression Language.

Declare the Java Bean in the JSP with the JSP tag <jsp:useBean>:

```
%><jsp:useBean id="pageHelper" class="com.adobe.training.PageHelper" /><%
```

Use the method getMessage() in an Expression Language:

```
<company:authorbox><c:out value="${pageHelper.message}" /></company:authorbox>
```

This line is a combination of using a bean in a custom JSP tag.

The following two exercises are very similar. The first is done completely in CRXDE, the second one is done in Eclipse, using Maven to build and install the package into the local CQ instance.

EXERCISE - Create a Component that Separates Markup and Functionality

Goal

- Create a component that lists child pages
- Markup and Functionality will be separated
- Use Tag Libraries, Java Beans, JSP Includes and JSTL

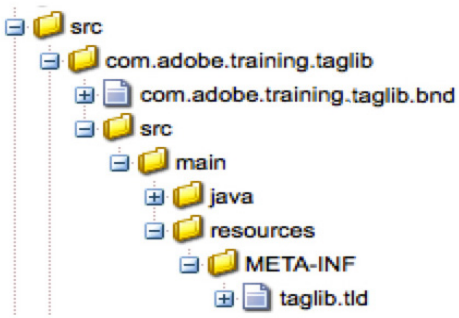
Steps

1. In /apps/company, add two folders, /src and /install.

2. In /apps/company/src, select " Create " then " Create Bundle " This will create the bundle folder. Include the .bnd file and prepare the bundle structure under /src/main
   Symbolic Name: com.adobe.training.taglib
   Name: Training Taglib
   Package: com.adobe.training

3. Create a new node, of node types sling:Folder, name META-INF under /src/main/java/resources.

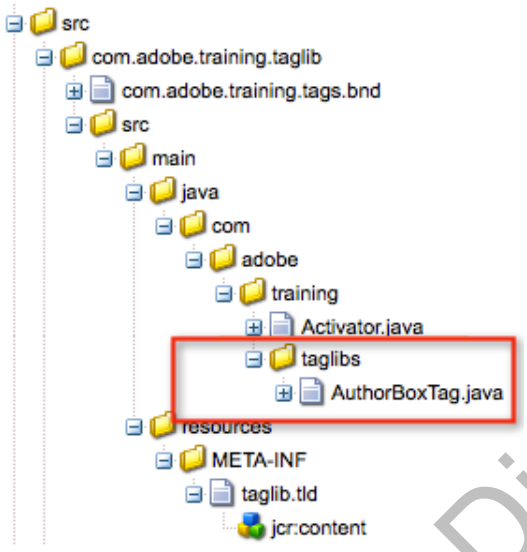
4. Add the tag library definition creating a file "taglib.tld"



5. Enter the tag library definition for an "author message box", which will display a message, but only in author-edit mode.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
    version="2.0">

    <description>Company Tag Library</description>
    <tlib-version>1.0</tlib-version>
    <short-name>company</short-name>
    <uri>http://www.example.com/taglibs/company/1.0</uri>

    <tag>
        <description>Infobox for authoring (author-mode only)</description>
        <name>authorbox</name>
        <tag-class>com.adobe.training.taglibs.AuthorBoxTag</tag-class>
    </tag>
</taglib>
```

6. Create a folder "taglibs" under src/main/java/com/adobe/training/ and then the tag library class "AuthorBoxTag.java" under the new folder.



7. Enter the code for the AuthorBoxTag.java class.

```
package com.adobe.training.taglibs;
```

```
package com.adobe.training.taglibs;

public class AuthorBoxTag extends BodyTagSupport {

    private static final long serialVersionUID = 1L;
    private static final Logger LOGGER = LoggerFactory.getLogger(AuthorBoxTag.class);
    private transient PageContext pc = null;

    public int doAfterBody() throws JspException {
        try {
            BodyContent bodyContent = super.getBodyContent();
            String bodyString = bodyContent.getString();
            if (WCMMode.fromRequest(pc.getRequest()) == WCMMode.EDIT) {
                JspWriter out = bodyContent.getEnclosingWriter();
                bodyString = "<div style=\"border-width:1px;border-style:solid;border-color:blue;\">"
                    + bodyString + "</div>";
                out.write(bodyString);
            }
        } catch (IOException e) {
            LOGGER.error("I/O error in authorbox tag: ", e);
        }
        return SKIP_BODY;

    }

    @Override
    public void setPageContext(PageContext p) {
        pc = p;
    }

    @Override
    public void release() {
        pc = null;
    }
}
```

8. In CRXDE Lite the packages will not be imported automatically. In that case add the following import statements to your code.

```
import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.BodyContent;
import javax.servlet.jsp.tagext.BodyTagSupport;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.day.cq.wcm.api.WCMMode;
```

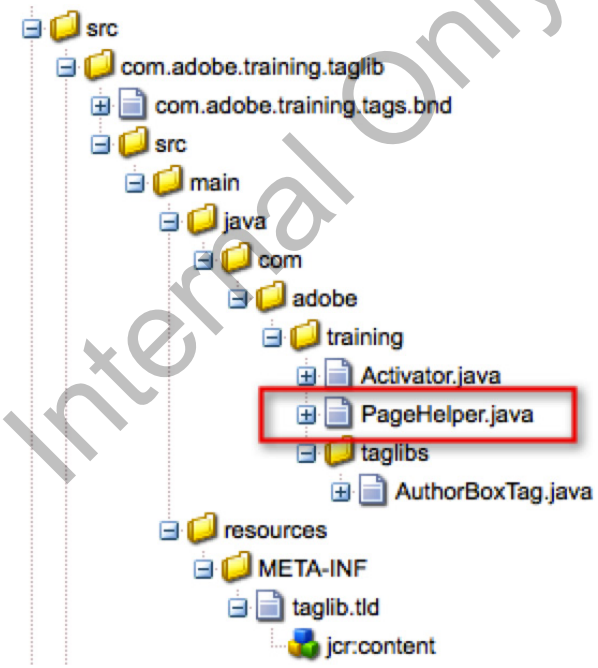
9. In your core pom.xml add the dependency (if not there already from a previous exercise)

```
<dependency>
      <groupId>com.day.cq.wcm</groupId>
      <artifactId>cq-wcm-api</artifactId>
</dependency>
```

10. Add a new exported package to your company-core/pom.xml (if not there already from a previous exercise:

```
<Export-Package>
    com.adobe.training.core.*, com.adobe.training.utils.*
</Export-Package>
```

11. Now create the PageHelper.java bean.



12. Open the PageHelper.java file and enter the code:

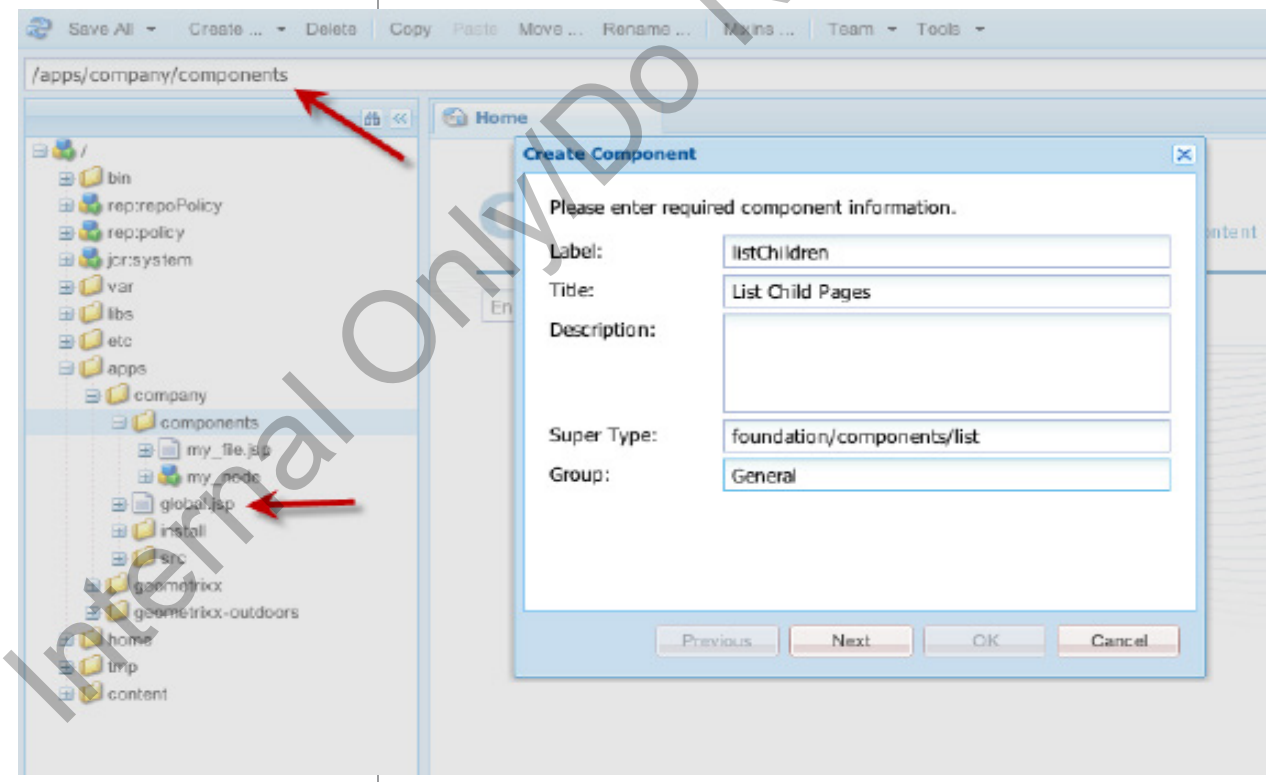
```
package com.adobe.training;

public class PageHelper {
    private static final String MESSAGE= "some helpful message";
    public String getMessage() {
        return MESSAGE;
    }
}
```

13. Before we create the component, we create a global.jsp file in /apps/company, with the purpose of defining the tag library. Make sure that you include it with the appropriate path in the JSP code in step 15 below. The content of the created file will be the code shown below:

```
<%@taglib prefix="company" uri="http://www.example.com/taglibs/company/1.0" %>
```

14. Build and Deploy.

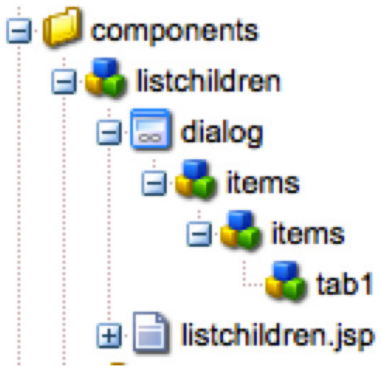
15. Now it's time to create the component to list the child pages.



16. Add the following code to the component's JSP file.

```
<%@include file="/libs/foundation/global.jsp"%><%
%><%@include file="/apps/company/global.jsp"%><%
%><jsp:useBean id="pageHelper" class="com.adobe.training.PageHelper" /><%
%><%@page session="false" %><%
%>
<c:forEach items="<%=currentPage.listChildren()%>" var="currentChild" varStatus="status">
    <p><c:out value="${status.count}" />: <c:out value="${currentChild.title}" /></p>
</c:forEach>
<company:authorbox><c:out value="${pageHelper.message}" /></company:authorbox>
```

17. You will need to add a dialog, so that the component is selectable from the paragraph list in Sidekick. Add just a few nodes, for the sake of having a dialog.



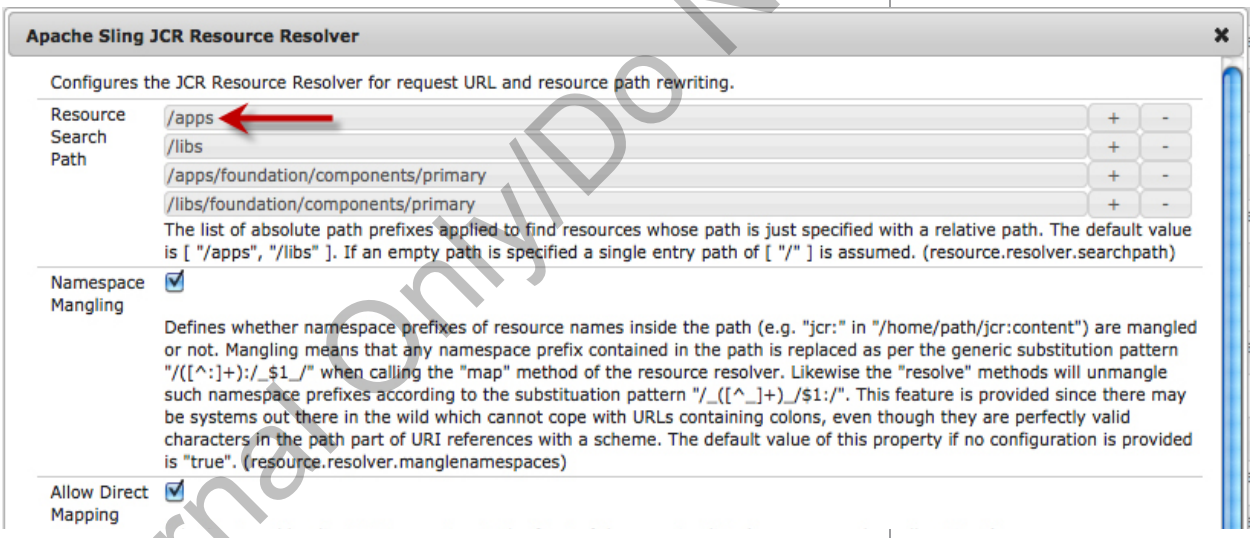
18. Choose a page with children and drag the listchildren component into the paragraph system and see the author box. Switch to "preview" mode and refresh the page. The author box will not be visible.

Note: The purpose of this exercise is to demonstrate how tag libraries, Java Beans (and EL), JSTL and JSP play together. In a real project you would extend this example to provide individual messages as needed.

15 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:resourceSuperType property of the node to which the ***sling:resourceType*** points

16 Content Migration/Import

It is a common requirement in many CQ5 projects to migrate content from an existing CMS into CQ. There are three basic possibilities to import content into CQ5:

- 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 CQ.
   - A variation on this method involves creating a content package (i.e. zip the exported content) which can then be handled by the CQ 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 cq: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 cq/jcr methods.
   - It is also useful for scripted fixes to the content structure.

EXERCISE - 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
   ```

   ...or on Windows:

   ```
   xcopy content\campaigns\geometrixx\scott-recommends content\capaigns\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
   ```

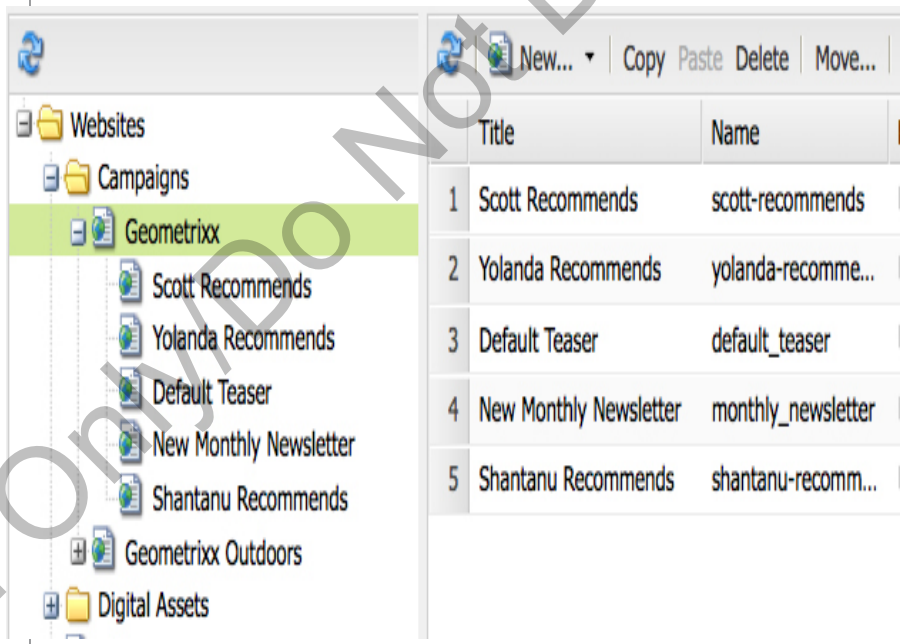
5. Edit `shantanu-recommends/.content.xml` and replace "Scott" with "Shantanu", e.g.:

```
<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.

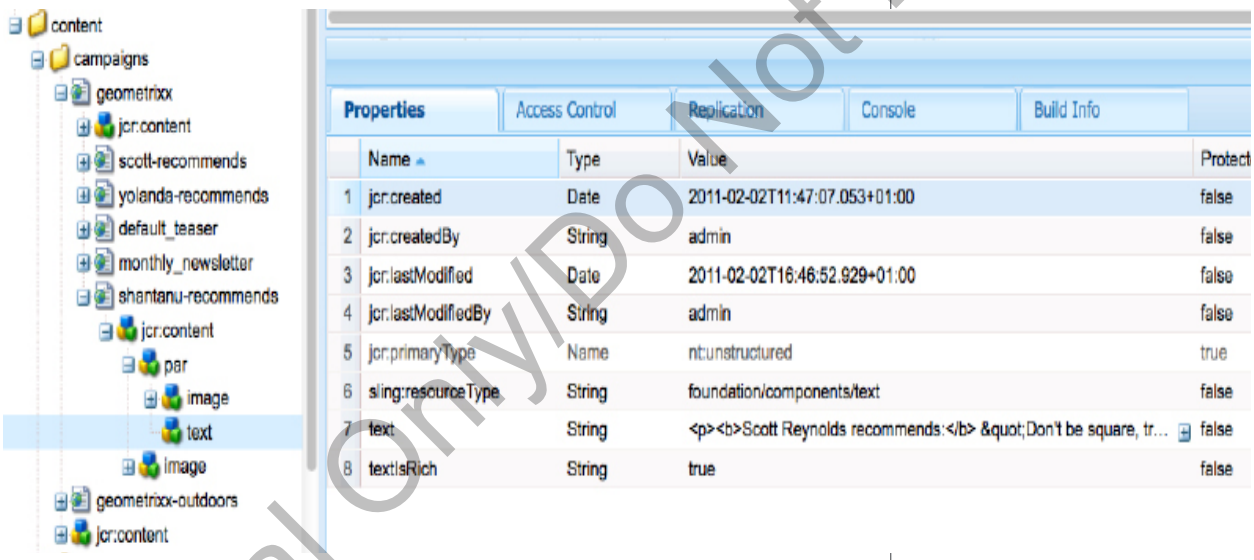
Installing Curl

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.

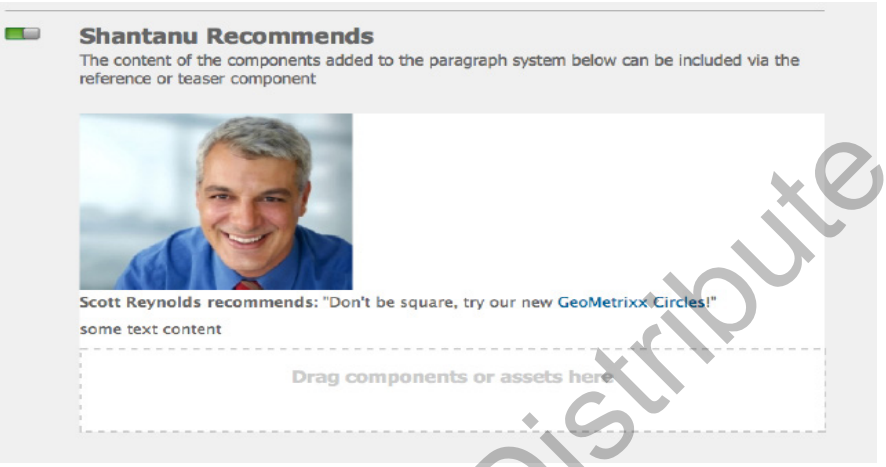


2. On the command line execute (in one line, below is split for better readability, and with no spaces in the URI):

```
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/shan-
tanu-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.

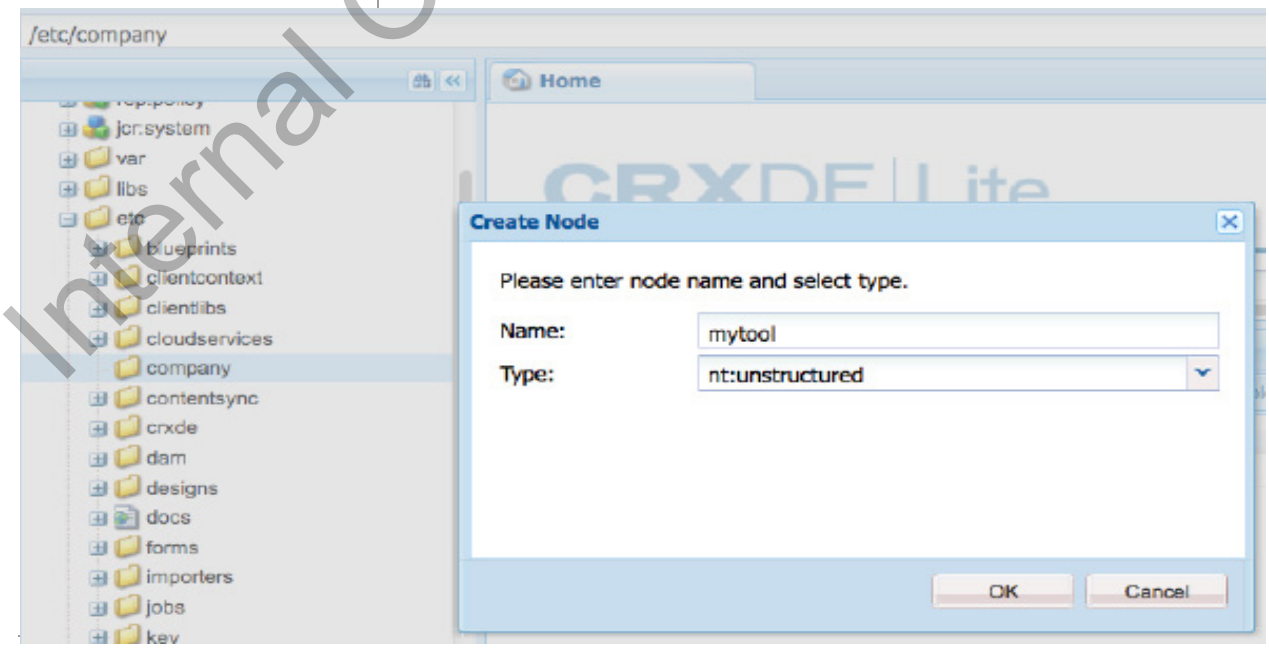
3. You should see when you open the campaign that there is a new text component:

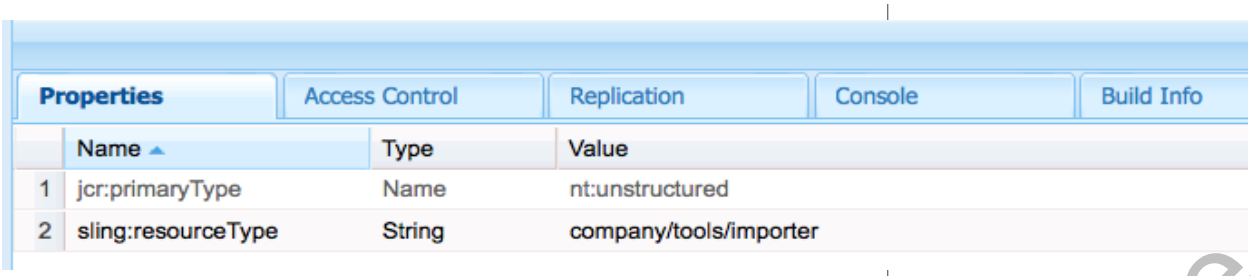


Using The JCR API

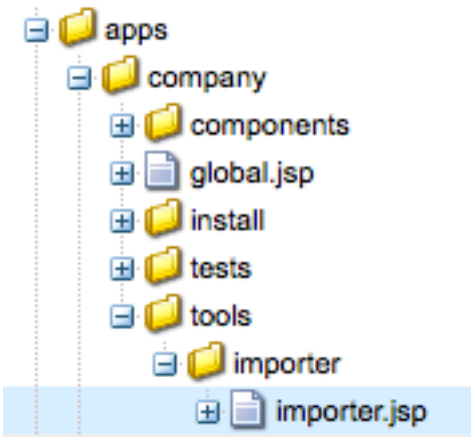
Finally, we will use a JSP script which makes use of the JCR API to search for specific content and add new content wherever it is found. This is a simple example, but the technique could be used to update content from a legacy system and add required nodes or properties to make the content useable in CQ.

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.

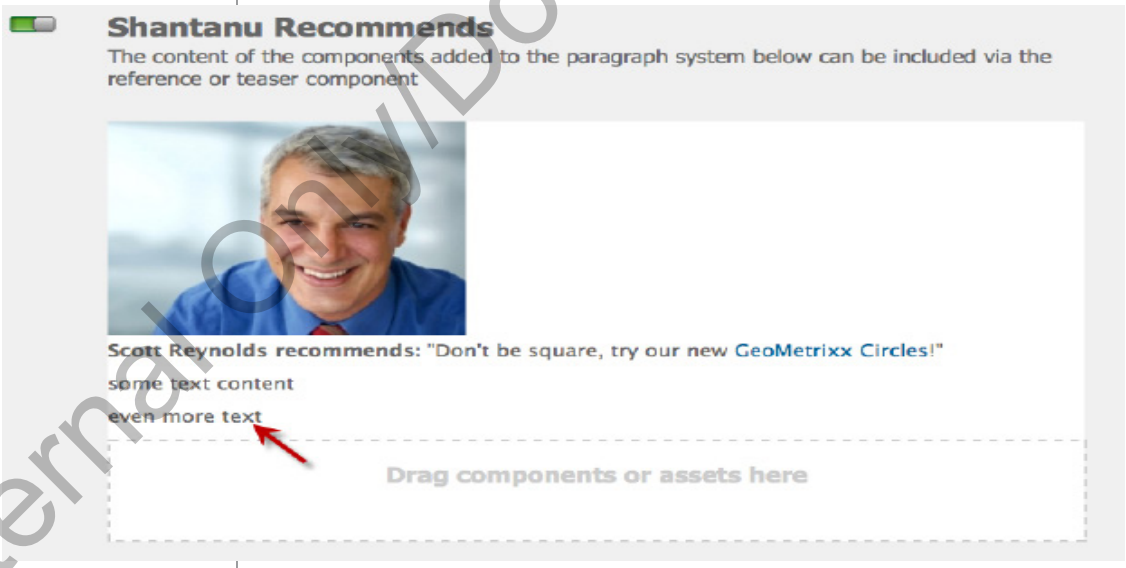


3. Now write the following jsp script, which will search for paragraph nodes in the Geometrixx campaigns, (by checking for `sling:resourceType='foundation/components/parsys'`) and add child nodes with text to all the nodes found:

```
<%@page import="javax.jcr.*,javax.jcr.query.*,java.util.*,
      com.day.cq.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:resourceType='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 CQ repository. These techniques can be adapted to solve many legacy migration issues that you may face.

17 Higher Level APIs

EXERCISE - Explore some higher level CQ APIs

When you are developing with CQ 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 CQ.. 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..

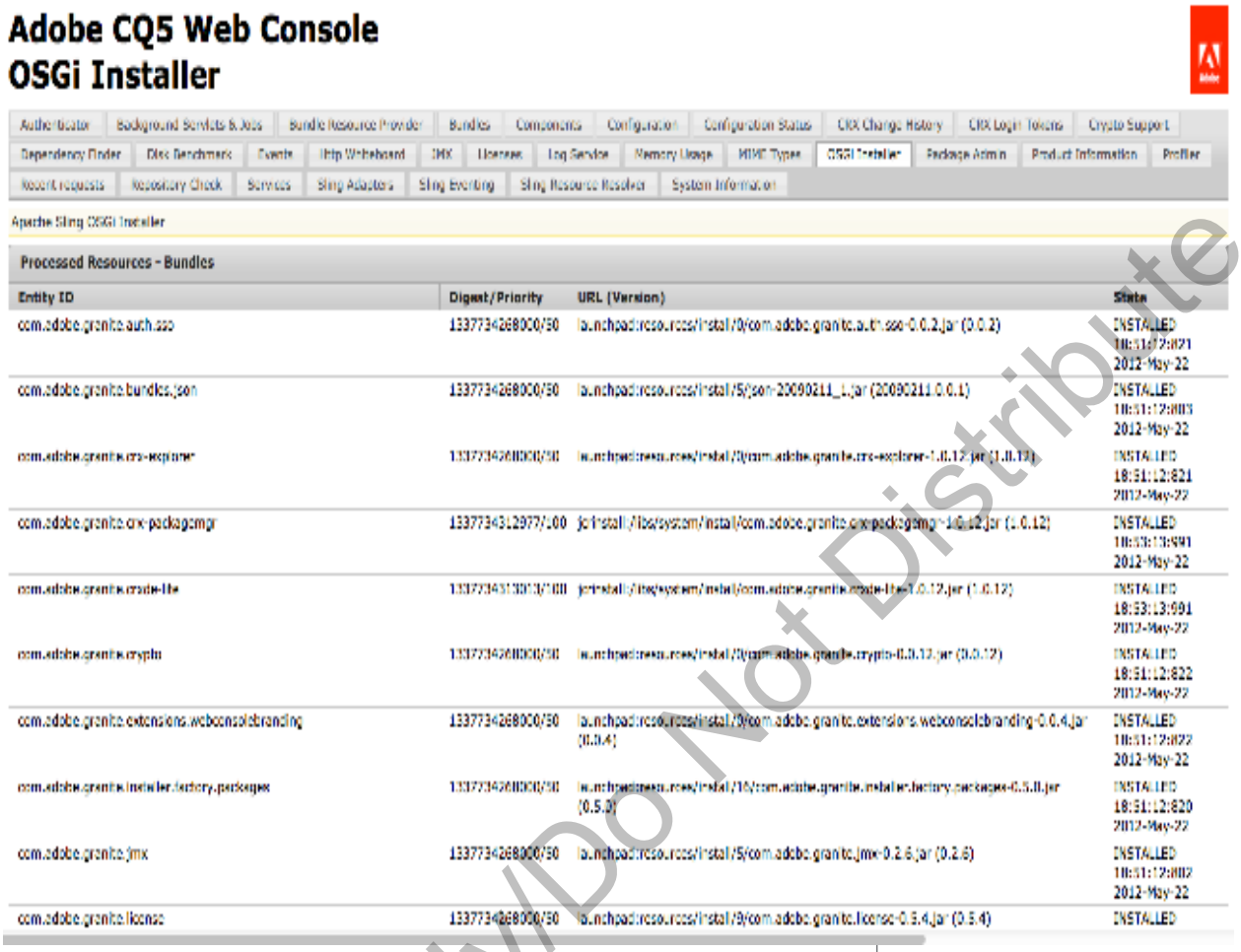
As stated during the developer training you should get familiar with all the classes and methods included in global.jsp. Those are:

- **ComponentContext** : The current component context object of the request (com.day.cq.wcm.api.components.ComponentContext interface).
- **Component** : The current CQ5 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 CQ5 WCM page object (com.day.cq.wcm.api.Page interface).
- **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 CQ5 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.Page interface).

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 is not just about the WCM API, some other high-level APIs are included with CQ, the full list of them can be seen in the Adobe CQ5 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.

Creating, tagging and activating a page using high-level APIs

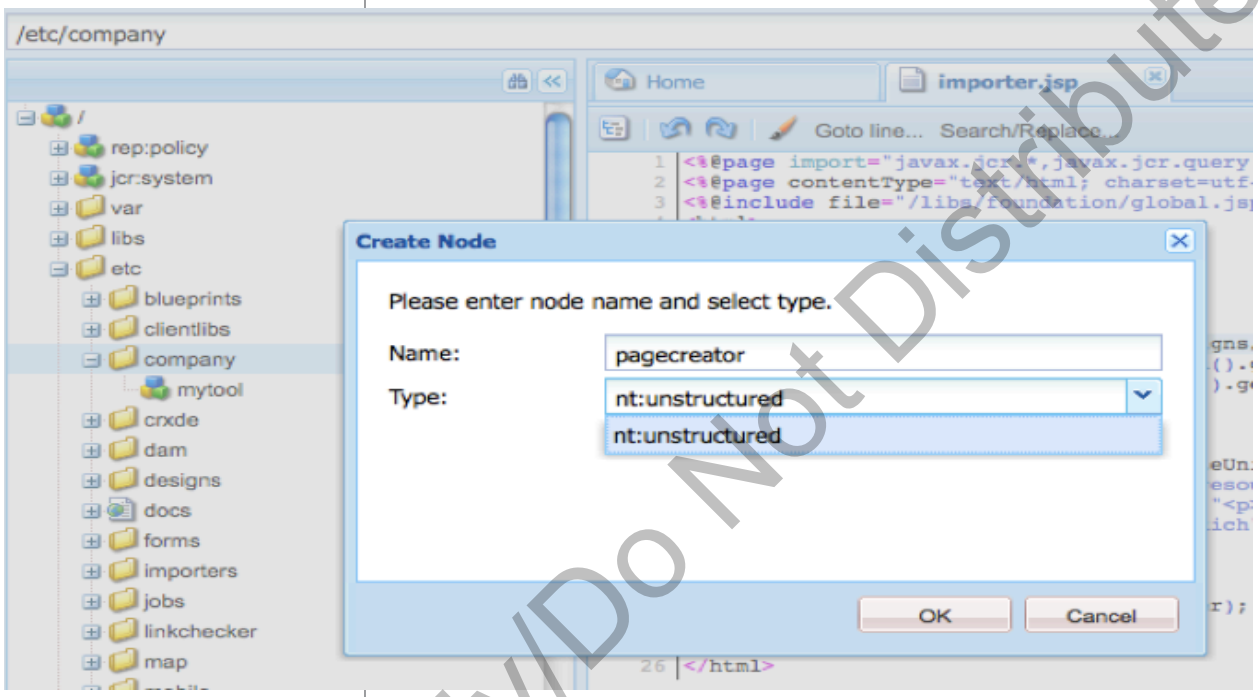
In our first exercise we will use the PageManager, TagManager and Replicator classes to create, tag and activate a page. We will do this by using a simple jsp page and with the help of our high level API. As you will see this can be made in a few lines of code.

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

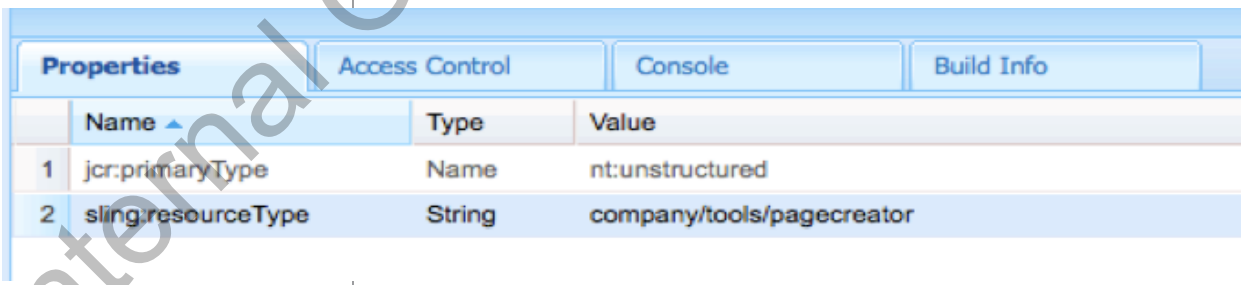
could be imported from external sources (something similar to the scaffolding tool), this is a very recurrent question during the developer training.

Steps

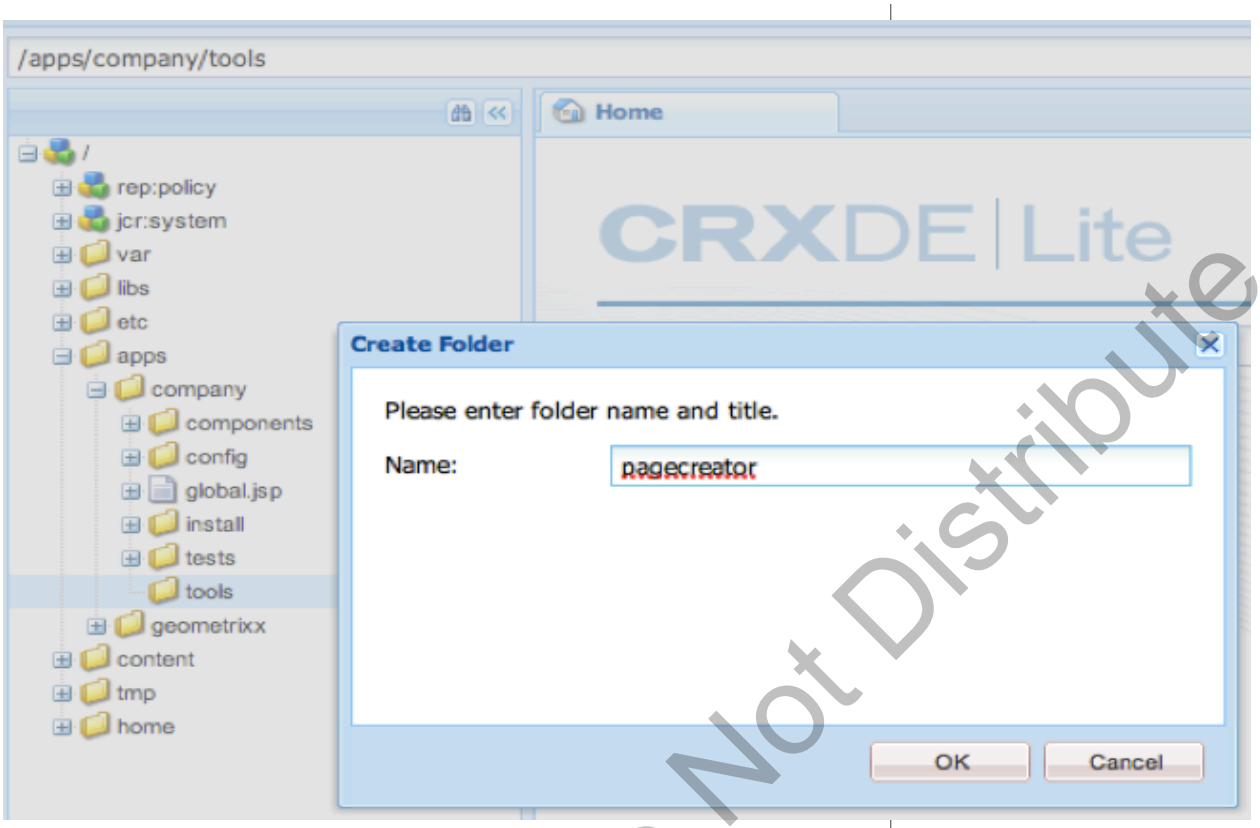
1. Open CRXDE and add an unstructured node "pagecreator" under the node "/etc/company" node created in the previous exercise.



The sling:resourceType of "pagecreator" shall be "company/tools/pagecreator"



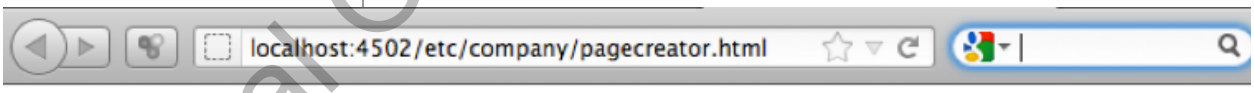
2. Create a new folder "pagecreator" under the folder /apps/company/tools created in the previous exercise.



3. Create a the file pagecreator.jsp under the newly created folder and enter the following code:

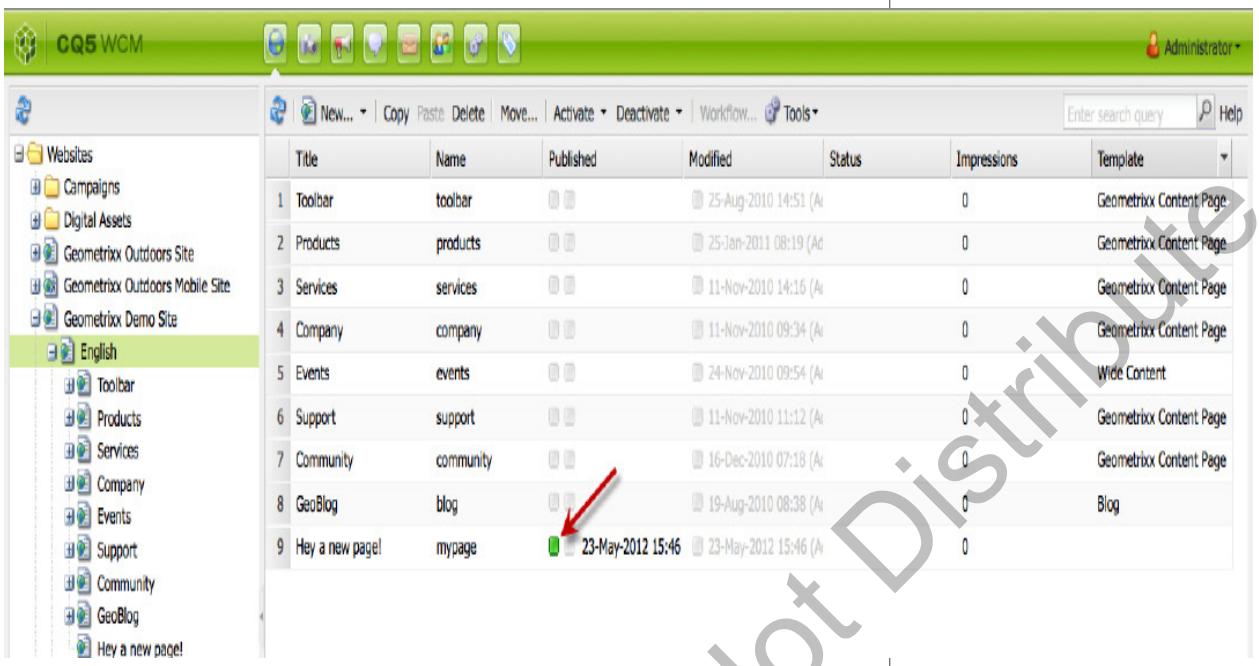
```
<%@page import="com.day.cq.tagging.*,
                com.day.cq.wcm.api.*,
                com.day.cq.replication.*"%>
<%@page contentType="text/html; charset=utf-8"%>
<%@include file="/libs/foundation/global.jsp"%>
<html>
<head>
<title>Page creator </title>
</head>
<body>
<%
   // pageManager is 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>
```

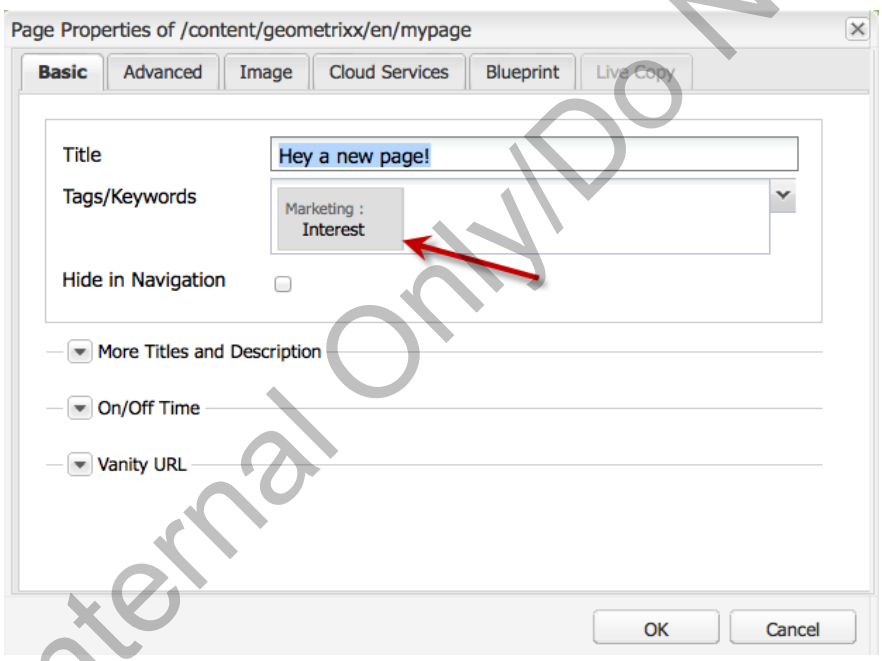
4. Execute the script at http://localhost:4502/etc/company/pagecreator.html



Page created, tagged and activated

5. Inspect the newly created page in the site admin.





6. Open the new page in the publish server.

Creating, tagging and activating an Asset using high-level APIs

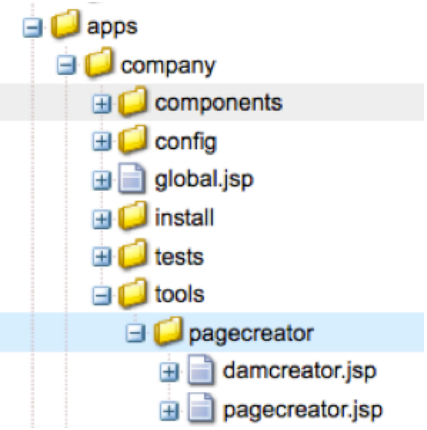
In this second exercise we will use the AssetManager, Asset, TagManager and Replicator classes to create, tag and activate an Asset. Again we will use our high level API to achieve this is a few lines of code.

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.

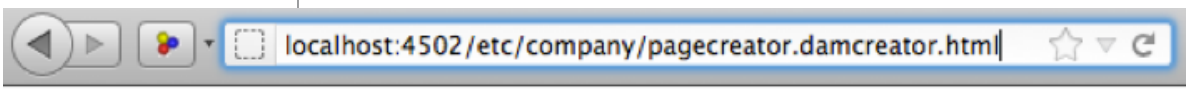
Taking advantage of the structure already created before we will create in CRXDE a new jsp called damcreator.jsp under /apps/company/tools/pagecreator.



2. Enter the following code:

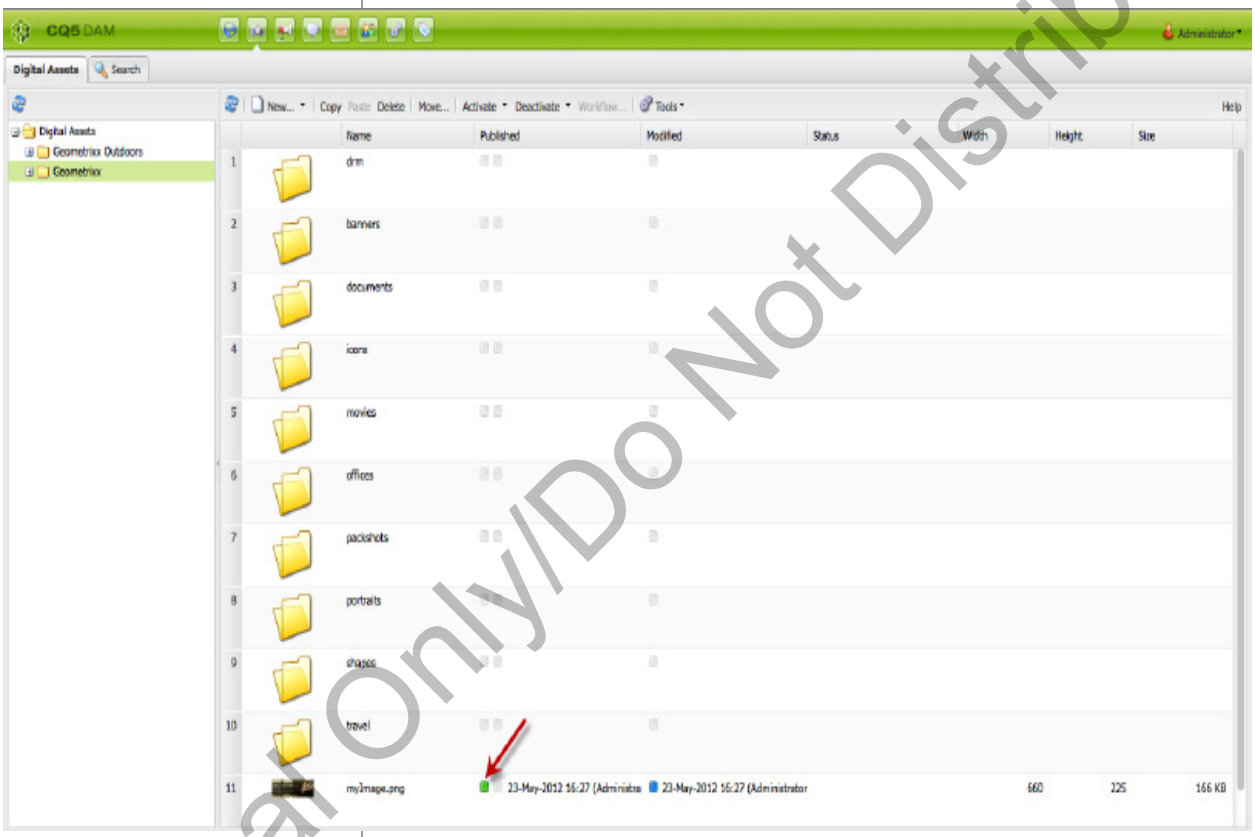
```
<%@page import="com.day.cq.tagging.*,
                com.day.cq.dam.api.*,
                com.day.cq.replication.*,
                java.net.URL"%>
<%@page contentType="text/html; charset=utf-8"%>
<%@include file="/libs/foundation/global.jsp"%>
<html>
<head>
<title>Asset creator </title>
</head>
<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.



Chapter Eighteen

18 Searching for Content

One of the powerful content repository features is the comprehensive search capability. The JCR 2.0 API offers XPATH and a SQL-type of query language (JCR-SQL2), but also a Query Object Model (JCR-QOM). In addition CQ 5 comes with a QueryBuilder, which consists of a Java API and a REST API.

Before we can explore the different search capabilities it is essential to understand the JCR concept and the hierarchical content model.

Understanding CRX and the Hierarchical Content Model

CRX is Adobe's implementation of the JSR 170 (aka JCR 1.0) and JSR 283 (aka JCR 2.0) Content Repository API for Java Technology. CRX 2.2 is the version installed with CQ 5.4 and is based on JSR 283. Therefore the focus will be only on JCR 2.0.

CRX Manages Content

CRX is an API, a Java API! CRX does not directly store content: CRX manages content. CRX delegates the storing of content to so called Persistence Managers. This is why CRX is aka a "Virtual Repository". CRX is an ideal player in the modern concept of content-centric application development, in which content elements are first retrieved then managed by the application. The API offers powerful, extendable services such as indexing, versioning, full-text search, observation and many more.

Understand the Hierarchical Content Model

Perhaps one of the most significant differences between CRX and a conventional RDBMS is the hierarchical content model. In an "Internet-driven" content-centric world a hierarchical model is a lot more powerful than the Entity Relationship Model. The goal of most Web initiatives is not to simply manage thousands of pages, but to reduce the "time-to-market" for new content to an absolute minimum.

A JCR repository is not necessarily a replacement for a RDBMS, but if offers many advantages when dealing with unstructured content, which is often the case in Web solutions. The "virtual" hierarchical content model (aka "tree structure") reflects the Web site structure. Content elements are directly related to distinct areas of the Web page and referenced by their "address". The main difference between a content-centric and an application-centric approach is that adding a content element to the Web page automatically extends the underlying content model while in an application-centric approach you first define the data model and then update the Web page.

Per JSR 170/283 definition the virtual hierarchy is defined by "Nodes" to split branches. Each content element is represented by a "Node". The logical tree structure however makes a distinction between "Nodes" and "Properties". Per definition a "Node" can have "Properties". Every "Node" has a unique address, which is a combination of slash-separated node-names, e.g.

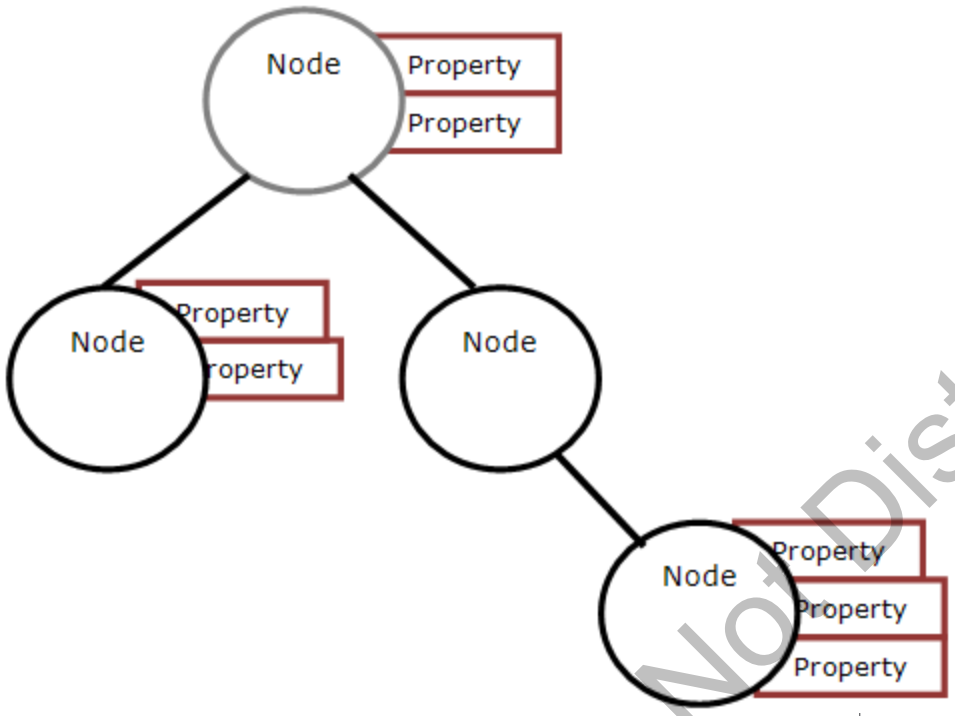
```
/parent/child/grandchild.
```

The hierarchical structure is achieved by defining a "virtual" dependency among nodes, e.g.

```
/parent/child/grandchild and /parent/child/anotherchild
```

have the same parent nodes and are in a virtual dependency.

The following graphic depicts the hierarchical content model.



The power of a JCR lays in the combination of the virtual hierarchical content structure and a powerful API to manage it. A developer's only concern is to get the address of a node. Once the address is available, the steps are always the same:

- Instantiate a Node object
- Get the Properties
- Display the Property values

A Note on Properties

A "property" is, similar to a "node", an address to access content. The technical differences to a node are encapsulated within the API and managed by "Persistence Managers". Each property represents a "content-element". This can be a text, an image, a video, anything. Adding a property is technically nothing else than adding a new address-name to a node. No structure required, no space allocation required. Just define an address and use the API to assign a content element to the property.

A Note on Persistence Managers

A Persistence Manager is the Java API to physically store and retrieve data. There are different PMs available, e.g. to store content as files or in RDBMS format (Oracle, Derby, etc.). Developers can also write their own PMs.

Searching Content

Finding a Node: Search and Query

It's getting obvious that searching for one or more nodes is an essential part of the content management process. In fact this exercise is all about searching nodes.

The purpose of any search or query is to find the nodes containing the information you were looking for. The result set is a collection of nodes and includes the properties containing the information you were searching for. The "full-text" search capability is a standard service provided by the JCR API.

Query Languages

Although the JCR API will look for nodes containing the search criteria, you still need to define what to search for. This is done by defining the search criteria using a special query syntax (or query language).

The JCR specification defines two query languages: xPath and SQL. The JCR specification group wanted to use common query languages and adapt them to the hierarchical content model.

The Abstract Query Model

The structure and evaluation semantics of a query are defined by an abstract query model (AQM) for which two concrete language bindings are specified: JCR-SQL2, which expresses a query as a string with syntax similar to SQL, and JCR-JQOM (JCR Java Query Object Model), which expresses a query as a tree of Java objects.

The languages are both direct mappings of the AQM and are therefore equally expressive; any query expressed in one can be machine-transformed to the other.

JCR-SQL2

The JSR 170 specification defines a language similar to the well-known RDBMS query language SQL. The syntax is very similar, with some slightly different meanings.

In JSR 283 the SQL syntax has been extended and introduced as JCR-SQL2. SQL2 is considered the standard search language for JCR 2.
With SQL2 it's possible to do a so called "full-text" search.

JCR implements the "Lucene Indexer", which indexes document content (e.g. from Word docs, Excel or PDF files). Having document content as part of the content index, it's possible to search content also within documents, all within one query statement.

JCR-JQOM

JCR-JQOM is a mapping of the AQM to a Java API. Each method and parameter name of the JCR-JQOM Java API corresponds to the type of the same name in the AQM grammar. The semantics of each JCR-JQOM method is described by reference to the semantics of the corresponding AQM production.

A JCR-JQOM query is built by assembling objects created using the factory methods of QueryObjectModelFactory.

XPath

XPath has been introduced in the JSR 170 specification. XPath is a common language to navigate through XML documents. Since XML docs also follow a hierarchical object model it makes sense to implement XPath as a query language.

In the JSR 283 specifications for JCR 2 the XPath query language has been deprecated. Since CRX 2.2 as part of CQ 5.4 is based on JSR 283, it's recommended not to implement XPath queries.

What the Query returns – The ResultSet

As perhaps expected, the result of a JCR query is a list of nodes where content matching the search term has been found. All the developer needs to do is to iterate through the list of node names, instantiate a node object and display whatever information is required (e.g. node title, content-snippet, etc.).

Useful Tools

The jcr-query-translator is a useful tool to translate queries from one query language to another. It allows to enter a query as SQL, XPath or QOM and translates to the other two languages. It can be found and downloaded at http://people.apache.org/~mreutegg/jcr-query-translator/translator.html.

The query debugger is included in all CQ instances and is useful for validating and testing dynamic queries. It can be accessed by e.g. from your local instance with http://localhost:4502/libs/cq/search/content/querydebug.html.

Unfortunately theses tools do not yet support JCR-SQL2.

EXERCISE - Display the Supported Query Languages

Goal

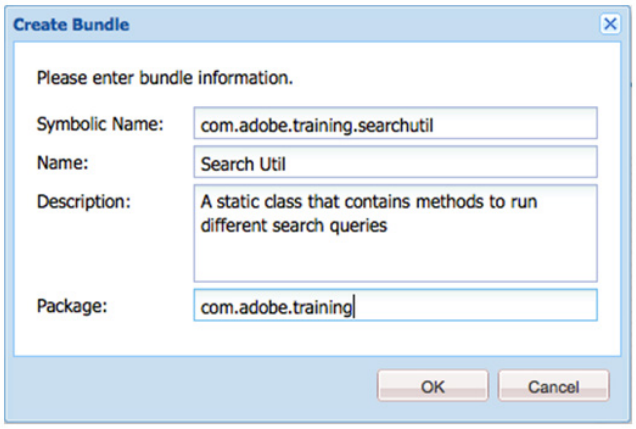
- Create a static class "SearchUtils.java" and add a method that returns a String array of the supported query languages. Additional methods will be added during this Search session.
- Create a component that uses the method to retrieve and display the list of supported query languages.

Steps

Assuming CRXDE is being used.

1. In your project folder (e.g. /apps/myproject) make sure you have two sub folders /src and /install. Create another sub folder under /src for this exercise, e.g. /src/searchexercise

2. Right-click on /src/searchexercise and select "Tools", then select "Create" and "Create Bundle ..." form the context menu.



3. Create the SearchUtil.java class and instantiate as "private".

```
public class SearchUtil {

      /**Reference to the logger class. */
      private static final Logger LOG = LoggerFactory.getLogger(SearchUtil.class;

      /**
        * Using a private constructor will prevent this class
        * from being instantiated.
        */
        private SearchUtil(){
               // Do nothing, just prevent this class from being instantiated
        }
```

4. At the bottom add a private method that will return the QueryManager object.

```
private static QueryManager getQueryManager(final Session session( {
      try {
        return session.getWorkspace().getQueryManager();
      } catch (RepositoryException e) {
        LOG.error(:Can not get the QueryManager object, returning null: Messages {}",e);
        return null;
      }
}
```

5. Add the method to return a String array of supported query languages.

```
public static String[] getSupportedQueryLanguages)final Session session) throws
RepositoryException {
      //Get QueryManager
      QueryMAnager queryManager = getQueryManager(session);

      //Get Supported queries languages
      return queryManager.getSupportedQueryLanguages();
}
```

6. Right-click on the ".bnd" file and select "Tools" and "Build Bundle" from the context menu.

7. Create a new component with the purpose to display the supported query languages.

8. Create a dialog with a "textfield" widget that will be used to enter a simple title (without a dialog the component cannot be selected from the paragraph list in Sidekick).

9. Write the component's JSP file.

```
<%--

  List of supported Query Languages component.

  Displays a list of query languages supported by CRX

--%><%
%><%@include file="/libs/foundation/global.jsp"%><%
%><%@page session="false" %><%
%><%@page import="com.xumak.training.SearchUtil"%><%

/**
 * First we check if the author has entered
 * a title. If not, we use "Supported Query Languages" as default title.
 */
final String DEFAULT_TITLE = "Supported Query Languages";
String sqlTitle = properties.get("sqlTitle", DEFAULT_TITLE);
%>
<c:set var="languages"
    value="<%= SearchUtil.getSupportedQueryLanguages(resourceResolver.adaptTo(Session.class)) %>"/>
  <div>
    <h1><%= sqlTitle %></h1>

    <c:forEach items="${languages}" var="result">
      <li> ${result}  </li>
    </c:forEach>

    <% out.println("<P> Resolver class name: "+resourceResolver.getClass()+"</P>"); %>
  </div>
```

10. Create a new Web page and enjoy the result!

JCR-SQL2

The Structured Query Language (SQL) is a language designed for querying and modifying relational databases. It has been the standard query language for RDBMS since the 80s.

In JSR 170 a SQL-like language has been introduced for searching the repository. It defines queries using strings that are similar to SQL.

The motivation was that many developers are familiar with the SQL syntax. This first version has a limited feature-set but was sufficient to search the content repository.

The 6. Query section of the JSR 283 specification introduced JCR 2.0 and the JCR-SQL2 query language. JCR-SQL from JCR 1.0 has been deprecated and replaced by the more powerful JCR-SQL2 language, by providing e.g. Joins and Selectors.

Comparison of SQL semantics

JCR-SQL

```
SELECT * FROM nt:base WHERE jcr:path LIKE '/content/%'
```

JCR-SQL2

```
SELECT * FROM [nt:base] WHERE ISDESCENDANTNODE([/content])
```

A JCR-SQL2 query has the following structure:

```
'SELECT' columns
          'FROM' Source
          ['WHERE' Constraint]
           ['ORDER BY' orderings]
```

The `columns` statement defines properties or nodes to be retrieve which are included in a source or selector.

The `source` statement defines the node types to be included in the search. One more source may be defined.

The `constraint` statement defines predicates to limit the source set. Constraints may include node paths, descendant nodes, Text Searches, etc.

The `order by` statement indicates how to order the results.

Some JCR-SQL2 Examples

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]
```

JCR-SQL2 JOINS

In JCR 2.0 a powerful search feature was introduced: JOINS. They work similar as in regular standard SQL language, where joins are used to query data from two or more tables. JCR-SQL2 has extended the concept to query data from two or more sources were a source represents a set of nodes of the same type. JCR-SQL2 joins provide a way to select nodes and filter the selection with conditions which depend on other nodes. In the JCR 2.0 specification the support for joins is optional.

CRX has support for "LEFT OUTER JOIN", "RIGHT OUTER JOIN" and "INNER JOIN" joins as defined by the JCR 2.0 specification. The level of join support can be verified by querying the repository descriptor table with the key "Repository. QUERY_JOINS". At the time of this writing joins suffer from performance issues and results are only accessible through the row view, so we do not recommend using joins unless necessary. Joins are necessary when querying properties or applying constraints on nodes that are descendants or parents of a set of nodes also included in the query.

JCR-JQOM

JCR-JQOM (Java Query Object Model) has been introduced in JSR 283. Each method and parameter name of the JCR-JQOM Java API corresponds to the type

of the same name in the AQM grammar. The semantics of each JCR-JQOM method is described by reference to the semantics of the corresponding AQM production.

The API allows for queries to be constructed by combining expressions. Fairly complex queries can be constructed this way. The structure of QOM is equivalent to JCR-SQL2.

JCR-JQOM query is built by assembling objects created using the factory methods of `QueryObjectModelFactory`.

A Query consists of:

- A **Source**. When the query is evaluated, the Source evaluates its selectors and the joins between them to produce a (possibly empty) set of node-tuples. This is a set of 1-tuples if the query has one selector (and therefore no joins), a set of 2-tuples if the query has two selectors (and therefore one join), a set of 3-tuples if the query has three selectors (two joins), and so forth.

- An (optional) **Constraint**. When the query is evaluated, the constraint filters the set of node-tuples.
- A list of zero or more **Orderings**. The orderings specify the order in which the node-tuples appear in the query results. The relative order of two node-tuples is determined by evaluating the specified orderings, in list order, until encountering an ordering for which one node-tuple precedes the other. If no orderings are specified, or if there is no ordering specified in which one node-tuple precedes the other, then the relative order of the node-tuples is implementation determined (and may be arbitrary).

- A list of zero or more **Columns** to include in the tabular view of the query results. If no columns are specified, the columns available in the tabular view are implementation determined, but minimally include, for each selector, a column for each single-valued non-residual property of the selector's node type.

A query is represented by a QueryObjectModel object, created with:

```
QueryObjectModel QueryObjectModelFactory. createQuery(Source
source, Constraint constraint, Ordering[] orderings,
Column[] columns)
```

`QueryObjectModel` extends `javax.jcr.query.Query` and declares:

```
Source QueryObjectModel.getSource()
Constraint QueryObjectModel.getConstraint()
Ordering[] QueryObjectModel.getOrderings()
Column[] QueryObjectModel.getColumns()
```

Some JCR-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);
```

EXERCISE - Search

Goal

- write a servlet that implements a Sling selector. The servlet shall perform a full text search below the requested node. (the results shall only be pages, but we will select for nt:unstructured, as the page content in jcr:content is of that type)

- The query results are returned as JSON

Steps

1. Create a servlet that responds to resource type geometrixx/components/homepage and selector "search"

```
@SlingServlet(resourceTypes = "geometrixx/components/homepage". selectors = "search")
public class SearchServlet extends SlingSafeMethodsServlet {
```

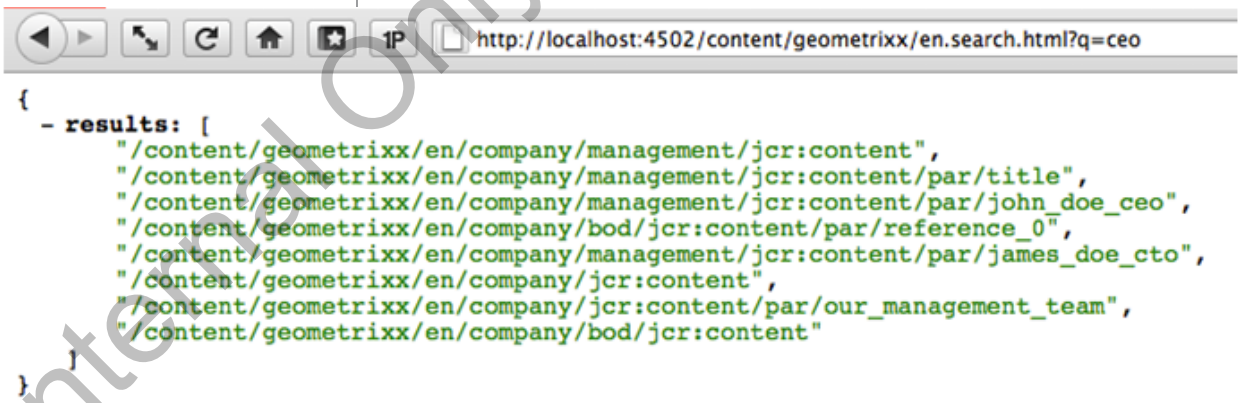
   a. Implement GET method that writes search results into JSONArray

```
@Overide
public final void doGet(final SlingHttpServletRequest request, final SlingHttpServletResponse
response)
      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 it is the
jcr:content node
           Node currentNode = request.getResource().adaptTo(Node.class);
           PageMAnager pageManager = request.getResource().getResourceResolver().
adaptTo(PageManager.class);
           // node that is the cq:page containing the requested node
           Node queryRoot = pageManager.getContainingPage(currentNode.getPath()).adaptTo(Node.
class);

           String queryTerm = request.getPatameter("q");
           if (queryTerm != null) {
                NodeIterator searchResults = performSearch(queryRoot, queryTerm);
                while 9searchResults.hasNext()) resultArray.put(searchResults.nextNode().
      getPath());
           }
      }
      catch (Exception e) {
           e.printStackTrace();
      }
      response.getWriter().print(jsonObject.toString());
}
```

2. Implement actual search method NodeIterator performSearch(Node queryRoot, String queryTerm)

```
private NodeIterator performSearch(Node queryRoot, String queryTerm) throws RepositoryException {
    // JQOM infrastructure
    QueryObjectModelFactory qf = queryRoot.getSession().getWorkspace().getQueryManager().getQOMFactory();

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

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

3. Results are available at e.g. http://localhost:4502/content/geometrixx/en.search.html?q=ceo



4. 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();
}
```

19 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

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.)
- *Interface 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 from RangeIterator.

- *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 CQ

- *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 oriented interfaces

- *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 thestandardJCRrepositoryinterfacewithtwomethods: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.