



Extend and Customize Adobe Experience Manager Student Guide



©2017 Adobe Systems Incorporated. All rights reserved.

Extend and Customize Adobe Experience Manager

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, Acrobat, the Creative Cloud logo, and the Adobe Marketing Cloud logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

All other trademarks are the property of their respective owners.

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

Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

May 12, 2017

Table of Contents

1	ARCHITECTURE OVERVIEW	9
2	INSTALLATION	14
	LAB A – START ADOBE EXPERIENCE MANAGER	20
	Exercise 1 – Start an AEM Author Instance	21
	Exercise 2 - Start an AEM Publish Instance	23
	Exercise 3 - Start AEM using the command line [optional]	26
3	DEVELOPER TOOLS.....	28
	LAB B - WORK WITH PACKAGES	36
	Exercise 1 – Install a package in AEM	37
	Exercise 2 - Create, Build, and Download a Package	40
4	CONFIGURING THE DEVELOPMENT ENVIRONMENT	43
	Working with Maven	43
	Installing and Configuring Eclipse.....	48
	LAB C – CONFIGURE THE DEVELOPMENT ENVIRONMENT	55
	Exercise 1 – Install and configure Eclipse (Optional)	56
	Exercise 2 – Install and Configure Eclipse AEM PLUGIN (optional).....	60
	Exercise 3 – Create an AEM project using Maven Archetypes	66
	Exercise 4 – Build and Deploy to Adobe Experience Manager	74
	Exercise 5 – Configure the AEM Server in Eclipse.....	80

5	CONFIGURING RUNMODES AND CONFIG NODES.....	88
Using Custom Run Modes	89	
LAB D – WORK WITH CUSTOM RUN MODE	98	
Exercise 1 – Start Adobe Experience Manager in Custom Run Mode	99	
Exercise 2 – Create a Configuration Node.....	101	
6	CONFIGURING CUSTOM LOG FILES.....	106
Understanding the Logging System	107	
Loggers and Writers.....	109	
LAB E - CONFIGURE CUSTOM LOGGERS.....	113	
Exercise 1 – Observe Project Logger Config Node	114	
Exercise 2 – Create Temporary Loggers	118	
7	DEEP DIVE IN TO OSGI.....	121
OSGi - An Overview	122	
OSGi Architecture	123	
Bundles	124	
Dependency Management Resolution	125	
Components and Annotations in OSGi	135	
LAB F – IMPLEMENT OSGI CONFIGURATION.....	141	
Exercise 1- Implement a Bundle Activator	142	
Exercise 2- Create and Use a Custom Service.....	146	
Exercise 3- Code OSGi Configurations.....	152	
Exercise 4 – Handle OSGi Events	158	

8 DEEP DIVE INTO SLING	163
The Sling Architecture.....	164
RESTful Architecture.....	164
Creating System Users	165
SlingRepository	170
Working with Sling Schedules	179
Scheduling Jobs	179
Working with Sling Models	180
LAB G – EXPLORE DIFFERENT AREAS OF SLING	186
Exercise 1 – Work with Servlets	187
Exercise 2 – Create a sling model	191
Exercise 3 – Create a System User	198
Exercise 4 – Create a Job Consumer for a Topic	202
Exercise 5 – Clean up Nodes with a Sling Scheduler	206
9 JCR DEEP DIVE	212
JCR Model	213
Event Modelling	217
Query Index.....	220
Indexing Tools	225
LAB H – IMPLEMENT THE JCR API	231
Exercise 1 – Create an Observation Listener	232
Exercise 2 – Write Queries for a Search Servlet	237
10 DEEP DIVE INTO AEM APIs.....	241
Polling Importers	242
Content Components.....	242

AEM Workflows.....	243
Workflow Launchers.....	248
Creating AEM Pages and Assets Programmatically	250
LAB I – IMPLEMENT AEM APIs	251
Exercise 1 –Create a Polling Importer	252
Exercise 2 – Connect External Data to an AEM Component.....	258
Exercise 3 – Execute a Workflow	264
Exercise 4 – Create a Custom Workflow Step.....	266
Exercise 5 –Use the Workflow Launcher to Monitor Polling events.....	270
Exercise 6 – Programmatically Create AEM Pages.....	279
Exercise 7 – Programmatically create a Website.....	288
11 WRITING TESTS	297
Understanding Testing Frameworks.....	298
Performing Unit Tests	299
LAB J – WRITING TESTS	303
Exercise 1 – Create Unit Tests using Mockito	304
Exercise 2 – Create Server-side Sling JUnit Test	308
Exercise 3 – Create Scriptable Server-Side Tests	317
Exercise 4 – Create an Automated test with Hobbes.....	322
12 CONTENT INGESTION.....	329
Understanding Data Migration	330
LAB K - USE DIFFERENT METHODS TO INGEST CONTENT	334
Exercise 1 – Create a Page using a Content Package	335
Exercise 2 – Add Content with the Sling POST Servlet.....	338
Exercise 3 – Add Content with the JCR API.....	340

Extend and Customize Adobe Experience Manager Student Guide

13	USERS, GROUPS AND PERMISSIONS	344
	Users and Groups	345
	Permissions and ACLs	347
	LAB L- WORK WITH ACLS.....	350
	Exercise 1 – Create a New User	351
	Exercise 2 – Create a New Group.....	353
	Exercise 3 – Add an User to a Group.....	356
	Exercise 4 – Create and Participate in a Custom AEM Project.....	359
	Part 1 – Create a Custom Workflow Process	359
	Part 2 – Increase Permissions on the Workflow Process USER	362
	Part 3 – Create a Custom Workflow	363
	Part 4: Create a Project Template.....	367
	Part 5: Testing the Business Process with Dynamic Permissions.....	369
	APPENDIX: VLT COMMAND LINE INSTALLATION	379
	Using FileVault	379
	Commonly Used FileVault Commands.....	379
	Installing and Configuring FileVault.....	379
	Using FileVault to Synchronize Content with Server	379
	Collaborating with Teams	380
	APPENDIX LAB - GENERATE PROJECT FILES FOR ECLIPSE.....	381
	APPENDIX LAB: INSTALL AND CONFIGURE VLT ON YOUR SYSTEM	383
	Exercise - Use VLT to perform content synchronization	384

APPENDIX: OPERATIONS DASHBOARD.....	389
Exercise 1: Diagnosing an issue using the Operations Dashboard.....	392
Exercise 2: Add a New Task to the Maintenance Schedule.....	394
Exercise 3 Modify the Maintenance Schedule.....	396
Exercise 4 Configure the Workflow Purge Task.	397
Extra credit exercise: Successfully run the Audit log Maintenance Task.....	401
Exercise 6 Working with Security Checks	404

1 ARCHITECTURE OVERVIEW

What is Adobe Experience Manager (AEM)?

Adobe Experience Manager is a content management system for building websites, content services, and forms. It is a web-based client-server system made up of a number of infrastructure-level and application-level functions. You can use these infrastructure and application-level building blocks to create customized solutions by building the required applications.



Infrastructure-Level Functions

At the infrastructure-level, Adobe Experience Manager has:

Web Application Server: You can deploy Adobe Experience Manager in standalone mode or as a web application within a third-party application server, such as WebLogic and WebSphere. The standalone mode includes an integrated Jetty web server.

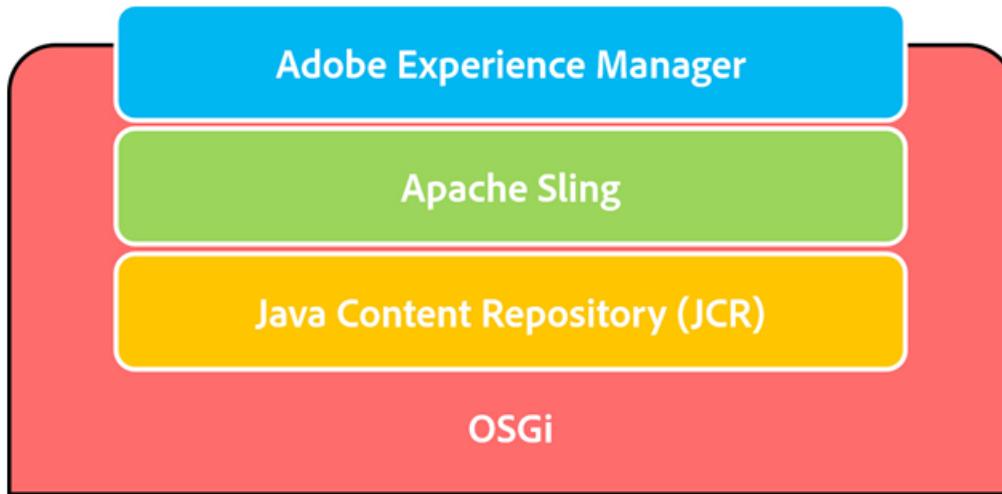
Web Application Framework: Adobe Experience Manager has a Sling web application framework that helps simplify the writing of REST, content-oriented web applications.

Content Repository: Adobe Experience Manager includes a Java Content Repository (JCR), a type of hierarchical database designed specifically for unstructured and semi-structured data.

Basics of the Architecture Stack

Adobe Experience Manager is a Java web application, and is based on technologies such as Open Service Gateway Initiative (OSGi), Java Content Repository (JCR), and Apache Sling.

The following diagram is a high-level view of the architecture stack.



Introduction to the Granite Platform

Granite is a general purpose platform for building robust scalable applications; it supports an “open architecture,” which is based on both “open standards” (JCR and OSGi) and “open source” projects (Apache Sling and Apache Jackrabbit).

Granite is Adobe's open web stack and Adobe Experience Manager is built on the Granite platform. Granite is open development, but **not open source**.

Technically, at the core, Granite provides:

An application launcher: Quickstart for standalone Java Application or a Web Application Archive for deployment in existing Servlet Containers or Application Servers.

An OSGi Framework into which everything is deployed.

A number of OSGi Compendium Services to support building applications: Log Service, Http Service, Event Admin Service, Configuration Admin Service, Declarative Services, and Metatype Service.

A comprehensive Logging Framework providing various logging APIs: SLF4J, Log4F, Apache Commons Logging, and OSGi Log Service.

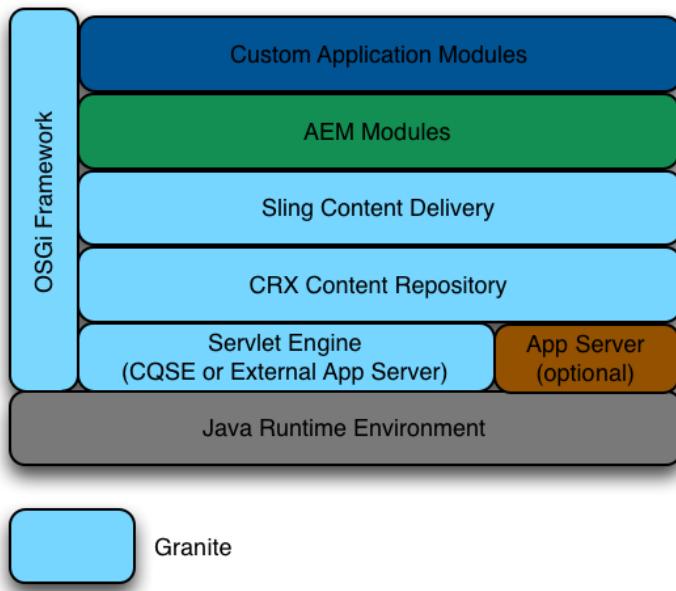
The JCR API Specification, based on Apache Jackrabbit and OSGified for Granite.

The Apache Sling Web Framework.

OSGi Framework

OSGi enables a collaborative and modular environment, where each application may be built and implemented as a small bundle. Each of these bundles is a collection of tightly coupled, dynamically loadable classes, JAR files, and configuration files that explicitly declare their external dependencies.

All content is stored in the content repository, which means backup is done at the repository level. OSGi runtime hosts Java applications that can access the repository using the JCR API. As part of the application runtime, you get Apache Sling, a RESTful web application framework that exposes the full repository content using HTTP and other protocols.



Apache Felix is an open source implementation of the OSGi for the Adobe Experience Manager framework. Apache Felix provides a dynamic runtime environment, where code and content bundles can be loaded, unloaded, and reconfigured at runtime.

Introduction to the Java Content Repository (JCR)

The JCR, specifically JSR-283 (Java Specification Request-283), is a database that supports structured and unstructured content, versioning, and observation. All data pertaining to Adobe Experience Manager such as HTML, CSS, JavaScript/Java, images, and videos are stored in the JCR object database. It is built with Apache Jackrabbit Oak, an open-source project. The Adobe implementation of JSR-283 used to be known as the Content Repository eXtreme (CRX). Hence, the naming of "CRX" you will see in some tools and interfaces in AEM. However, the CRX as a feature of AEM is being phased out. In its place is the Granite platform, and the use of Apache Jackrabbit Oak.

Adobe Experience Manager also works with other JCR repositories, such as Apache Jackrabbit 2.x, and with a number of non-JCR data stores through connectors. Because Adobe Experience Manager is built on top of this standard, it is capable of pulling content not just from its built-in Oak repository, but from any JCR-compliant source, such as a third-party repository (for example, Jackrabbit 2.x) or a connector that exposes legacy storage through JCR.

Advantages of using JCR:

- JCR provides a generic application data store for structured and unstructured content. While file systems provide excellent storage for unstructured, hierarchical content, and databases provide storage for structured data, JCR provides the best of both data storage architectures.
- JCR supports 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 (:). For example: jcr:title

Introduction to Apache Sling

Apache Sling is a web application framework for content-centric applications, and uses a Java Content Repository (such as Apache Jackrabbit Oak) to store and manage content. Apache Sling is based on Representational State Transfer (REST) principles, and helps build applications as a series of OSGi bundles.

Advantages of Apache Sling:

Apache Open Source

Sling is based on REST principles

applications are built as a series of OSGi bundles

is resource-oriented (every resource has a URI), and maps to JCR nodes

The embedded Apache Felix OSGi framework and console provides a dynamic runtime environment, where code and content bundles can be loaded, unloaded, and reconfigured at runtime. Sling makes it easy to implement simple applications, while providing an enterprise-level framework for more complex applications.

Sling assumes "Everything is a Resource". That is, Sling is resource-oriented, where the resources have URLs, and each resource is mapped into a JCR node. A request URL is first resolved to a resource, and then based on the resource, it selects the Servlet or script to handle that request. Servlets and scripts are handled uniformly in that they are represented as resources themselves and are accessible by a resource path. This means every script, Servlet, filter, error handler, and so on is available from the ResourceResolver just like normal content—providing data to be rendered on request.

The Functional Building Blocks of AEM

The Adobe Experience Manager application modules such as Sites, Assets, Communities, Forms, and Apps, sit on top of the Adobe Experience Manager shared framework. This shared framework is known as Granite, and includes all the application layer functionality such as mobile functionality, multisite manager, taxonomy management, and workflow. These functionalities are shared among all the application modules. In addition to these functionalities, these Adobe Experience Manager applications share the same infrastructure and UI framework, and are very tightly integrated with each other. All of this is encapsulated within the OSGi container.

Application-Level Functions

At the application-level, Adobe Experience Manager has the following functions to manage:

- Websites
- Content Services
- Forms
- Digital Assets
- Communities
- Online Commerce

2 INSTALLATION

Adobe Experience Manager Instances

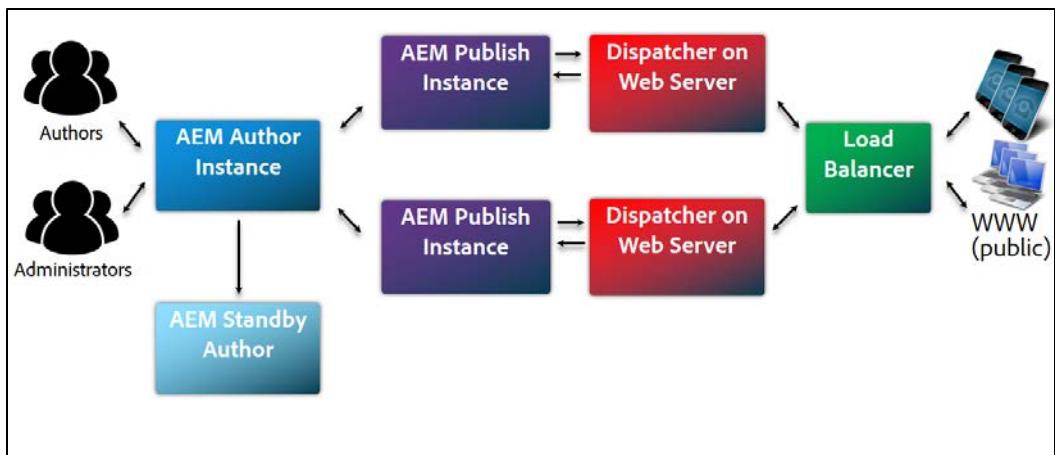
Adobe Experience Manager runs on most operating systems that support the Java platform. All client interactions with Adobe Experience Manager are done through a web browser.

In Adobe Experience Manager terminology, an “instance” is a copy of Adobe Experience Manager running on a server. Adobe Experience Manager installations usually involve at least two instances running on separate computers and a dispatcher:

- Author: An Adobe Experience Manager instance used to create, upload, and edit content, and administer the website. After content is ready to go live, it is replicated to the Publish Instance.
- Publish: An Adobe Experience Manager instance that serves the published content to the public.
- Dispatcher: A static web server (Apache httpd, Microsoft IIS, and so on) augmented with the Adobe Experience Manager Dispatcher module. It caches webpages produced by the Publish instance to improve performance.



NOTE: The author and publish instances are the same software stack but two different run modes.



Installation Prerequisites

To install Adobe Experience Manager, you need:

- Adobe Experience Manager installation and startup JAR file
- A valid Adobe Experience Manager license key properties file
- JDK version 1.8
- Approximately 4 GB of free space per instance
- Approximately 4 GB of RAM (at the **very minimum!**)

The Adobe Experience Manager installation and startup JAR file is also known as the "quickstart" file. You use the file to install Adobe Experience Manager. Once installed, the file is referred to as the Adobe Experience Manager startup file. During installation, you will notice the JAR file creates a root folder called **crx-quickstart**.

You also need to set environment variables as a part of your JDK 1.8 setup.



NOTE: You can download the latest JDK version from the following link:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

Installing Adobe Experience Manager on Your System

In general, when you want to install Adobe Experience Manager on your system, you would follow this procedure:

1. Create two specific folder structures for your Adobe Experience Manager instances.

➤ Author	instance
For	Windows: C:/adobe/AEM/author
For Mac OS or *x:	/opt/adobe/AEM/author OR /Applications/AEM/author
➤ Publish	instance
For	Windows: C:/adobe/AEM/publish
For Mac OS or *x:	/opt/adobe/AEM/publish OR /Applications/AEM/publish

2. Add the **AEM Quickstart JAR** file along with the license.properties file to each folder which you created earlier.
3. Rename the jar file to include the run mode as well as the port number. That is, rename the file to the format:
aem-<run mode>-<port number>.jar.

4.

For example,

Author instance: aem-author-4502

Publish instance: aem-publish-4503

You can therefore control the way Adobe Experience Manager is installed by defining properties via file name.

The first time you double-click the jar file, Adobe Experience Manager will install on your system, creating a root folder called crx-quickstart, which serves as your repository.

A sample folder structure for an Author instance is shown below.

Name	Date modified	Type	Size
crx-quickstart	3/6/2017 1:40 PM	File folder	
aem-author-4502.jar	3/6/2017 11:41 AM	Executable Jar File	525,452 KB
license.properties	1/12/2017 3:13 PM	PROPERTIES File	1 KB



NOTE: The Adobe Experience Manager quickstart file is renamed for installation purposes. When running for the first time, the quickstart file will notice that it has to install Adobe Experience Manager. By renaming the file, you use a convention of passing the instance name (Webpathcontext) and port number through the file name so no user interaction is needed during the installation process. If no port number is provided in the file name, Adobe Experience Manager will select the first available port from the following list in this specific order: 1) 4502, 2) 8080, 3) 8081, 4) 8082, 5) 8083, 6) 8084, or a random port.



NOTE: If you have multiple author and multiple publish instances, a best practice to consider is using an even/odd numbering paradigm for port numbers. So, your author instances would be 4502, 4504, 4506, and so on. Your publish instances would be 4503, 4505, 4507, and so on.

Starting an Adobe Experience Manager Instance

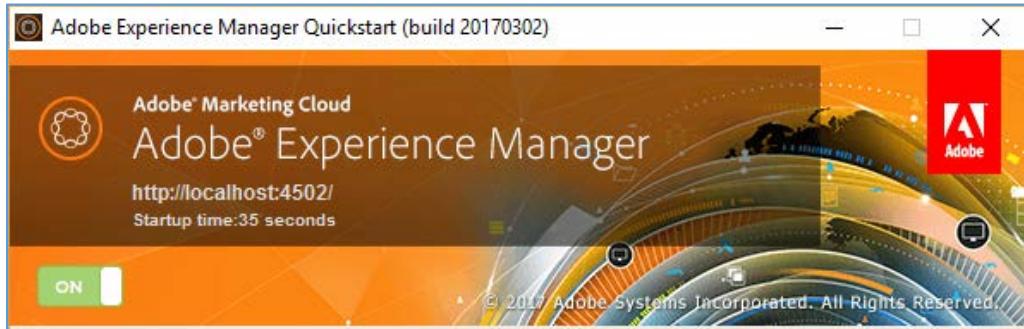
There are many ways of starting an Adobe Experience Manager instance, two of which are—graphical and by command line. The latter is more powerful because you have the possibility of providing additional performance-tuning parameters to the Java Virtual Machine (JVM).

Using the *.jar file to Start an Adobe Experience Manager Instance

In a Windows or Mac OS environment, you can double-click the aem-author-4502.jar file to start an Author instance (or the aem-publish-4503.jar file for a Publish instance).

Installation will take approximately 5-7 minutes, depending on your system's capabilities.

- A dialog window will pop up similar to the following (this is known as the GUI):



After Adobe Experience Manager starts, your default browser will open automatically, pointing to Adobe Experience Manager's start URL (where the port number is the one you defined on installation).

Using the Command Line to Start an Adobe Experience Manager Author Instance

Prior to the installation, you may want to know which parameters are available to configure quickstart. Enter the following command to display a complete list of optional parameters:

```
java -jar aem-author-4502.jar -h
```

The Adobe Experience Manager quickstart installer will show all available command-line options without starting the server. In addition, you need to tune the JVM used for running Adobe Experience Manager. Tuning the JVM is an important and delicate task and requires a more realistic environment in terms of resources (hardware, operating system, and so on) and workload (content, requests, and so on). For now, it will be enough to know that you can start your instance (Author or Publish) using the following parameters:

-Xms --> assigns the initial heap size

Default value	64 MB for a JVM running on 32-bit machines, or 83 MB for 64-bit
---------------	---

	machines
Recommended	Specific to physical memory available and expected traffic
Syntax	-Xms512m (sets the initial heap size to 512 MB)

-Xmx --> assigns the maximum size the heap can grow

Default value	64 MB for a JVM running on 32-bit machines, or 83 MB for 64-bit machines
Recommended	Specific to physical memory available and expected traffic, but should be equal or greater than the initial size. To run Adobe Experience Manager, it is recommended to allocate at least 1024 MB of heap size.
Syntax	-Xmx1024m (sets the maximum size for the heap. In the example, we are letting it grow to 1024 MB; however, in production, this should be higher because Adobe Experience Manager consumes a lot of resources).

You can now install and start Adobe Experience Manager from the command line together with increasing the Java heap size, which will improve performance.

Using the Command Line to Start Adobe Experience Manager Publish Instance

If you wanted to start AEM using a command prompt, navigate to the directory containing your quickstart jar file (such as **/adobe/AEM/publish**), and enter the following command to install the publish instance:

```
java -jar aem-publish-4503.jar
```

LAB A – START ADOBE EXPERIENCE MANAGER

Overview

In this lab, you will start an AEM author instance and publish instance.

Objectives

- Start an AEM author instance
- Start an AEM publish instance
- Start AEM using the Command line

Pre-requisites

You need the following to complete the tasks in this module:

- Adobe Experience Manager quickstart JAR file
- license.properties file

Directions

Complete the exercises that follow.

Exercise 1 – Start an AEM Author Instance

NOTE: If you are attending a VILT class using ReadyTech, steps 1 through 3 were completed for you.

Steps:

1. Create a folder structure on your file system where you will store, install, and start your Adobe Experience Manager author instance. For example:

Windows: C:/adobe/AEM/author

MacOS X: /Applications/adobe/AEM/author or *x: /opt/adobe/AEM/author

2. Copy the aem-quickstart-6.3.0.jar and license.properties files from the Exercise_Files zip to your newly created directory.

3. Rename the aem-quickstart-6.3.0.jar file to aem-author-4502.jar:

aem = Application

author = Web Content Management (WCM) mode it will run in (in this case, Author)

4502 = Port it will run in

Name	Date modified	Type	Size
aem-author-4502.jar	3/6/2017 11:41 AM	Executable Jar File	525,452 KB
license.properties	1/12/2017 3:13 PM	PROPERTIES File	1 KB

4. In a Windows or MacOS X environment, double-click the aem-author-4502.jar file. Installation will take approximately 5–7 minutes depending on your system's capabilities.
5. After Adobe Experience Manager Author instance has started successfully, the start-up screen (the GUI) will change to something similar to the following:



6. In addition, after Adobe Experience Manager starts, your default browser will automatically open to Adobe Experience Manager's start URL (where the port number is the one you defined on installation); for example: <http://localhost:4502>. A Sign In screen will be displayed:



NOTE: A crx-quickstart directory is also created on your machine:

Name	Date modified	Type	Size
crx-quickstart	3/6/2017 1:40 PM	File folder	
aem-author-4502.jar	3/6/2017 11:41 AM	Executable Jar File	525,452 KB
license.properties	1/12/2017 3:13 PM	PROPERTIES File	1 KB

Exercise 2 - Start an AEM Publish Instance

NOTE: If you are attending a VILT class using ReadyTech, steps 1 through 3 were completed for you.

Steps:

1. Create a folder structure on your file system where you will store, install, and start your Adobe Experience Manager publish instance. For example:
 - Windows: C:/adobe/AEM/publish
 - MacOS X: /Applications/adobe/AEM/author or *x: /opt/adobe/AEM/publish
2. Copy the aem-quickstart-6.3.0.jar and license.properties files from the Exercise_Files zip to your newly created directory.
3. Rename the aem-quickstart-6.3.0.jar file to aem-publish-4503.jar:
aem = Application
publish = Web Content Management (WCM) mode it will run in (in this case, Author)
4503 = Port it will run in

Name	Date modified	Type	Size
aem-publish-4503.jar	3/6/2017 11:41 AM	Executable Jar File	525,452 KB
license.properties	1/12/2017 3:13 PM	PROPERTIES FILE	1 KB

4. In a Windows or MacOS X environment, double-click the aem-publish-4503.jar file. Installation will take approximately 5–7 minutes depending on your system's capabilities.
5. After Adobe Experience Manager Publish instance has started successfully, the start-up screen will change to something similar to the following:



6. In addition, the Adobe Experience Manager login page opens from your default browser (where the port number is the one you defined on installation); for example, <http://localhost:4503>.
7. The following screen appears once the Publish instance is up and running:



NOTE: There is no need to sign in. The publish instance loads the We.Retail reference site immediately.

NOTE: We.Retail is a reference implementation that illustrates the recommended way of setting up an online presence with Adobe Experience Manager. While We.Retail illustrates a retail vertical, the way the site is set up can be applied to any vertical. Only the product catalog and cart features are retail-specific.

You have now successfully installed and started Adobe Experience Manager Author and Publish instances on localhost.

To stop an Adobe Experience Manager instance, click the "on / off" toggle button in the GUI window:



To start Adobe Experience Manager in the future, double-click the renamed `aem-quickstart-6.3.0.jar` file; for example, `aem-author-4502.jar`.

Exercise 3 - Start AEM using the command line [optional]

Overview:

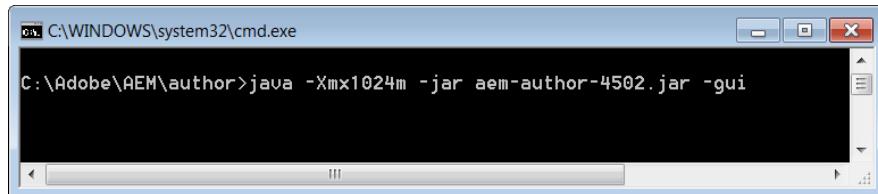
You already have an author instance and a publish instance running. Perform this task only when necessary or as an add-on exercise to try out beyond this class.

This is a powerful method because you can provide additional performance-tuning parameters to the Java Virtual Machine (JVM). On Windows, MacOS X, or *x, you can install or start Adobe Experience Manager from the command line, while increasing the Java heap size, which improves performance.

A typical command line to start AEM by setting the Java heap size will have the following:

```
java -Xmx1024m -jar aem-author-4502.jar -v
```

This example command below starts AEM author runmode with a specific memory allocation to the JVM and the GUI window "on":



Steps:

1. Stop your author instance by clicking the **On** button in the GUI window (see screenshot at the top of this page).
2. In your command prompt, navigate to the `Adobe\AEM\author` directory (or the directory where your author *.jar file is), and use the following command to start

Adobe Experience Manager the very first time without installing the We.Retail reference site:

```
java -jar aem-author-4502.jar -r author, nosamplecontent -gui
```

NOTE: You would run with "nosamplecontent" if you are performing a production installation, in which case the sample content is not needed. Also, note the "nosamplecontent" option is only available upon first starting the instance.

TIP: To open a directory in Windows Explorer in the command-line, select the directory, hold down the Shift key, and right-click. Then, you will see an option to open that directory in a command-line window.

Summary

You should now be able to:

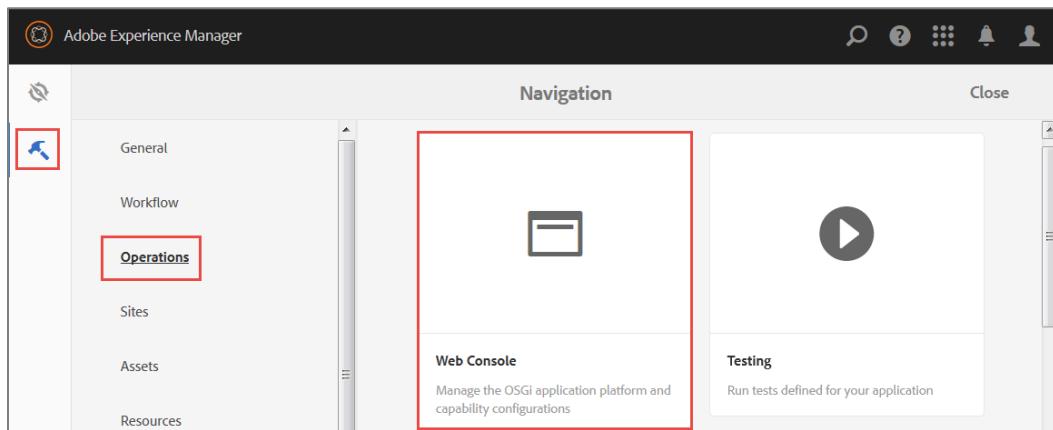
- Install and run the Adobe Experience Manager Author instance
- Install and run the Adobe Experience Manager Publish instance

3 DEVELOPER TOOLS

The Web Console

The Web Console in Adobe Experience Manager is based on the Apache Felix Web Management Console. It is used to manage various bundles and configurations. Any changes made through this console are automatically applied to the running system, without the need to restart the instance.

You can access the console at: <http://localhost:4502/system/console> or by navigating to Tools > Operations > Web Console, as shown:



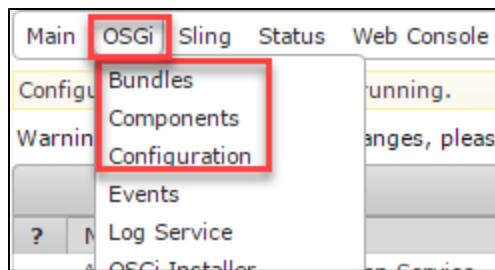
If you use the navigation in AEM, the Web Console opens in a new tab in your browser.

The most important menu selections under the OSGi tab are:

Bundles—used for installing and managing bundles.

Components—used for managing and controlling the status of components required for Adobe Experience Manager.

Configuration—used for configuring OSGi bundles, and is the underlying mechanism for configuring Adobe Experience Manager parameters.

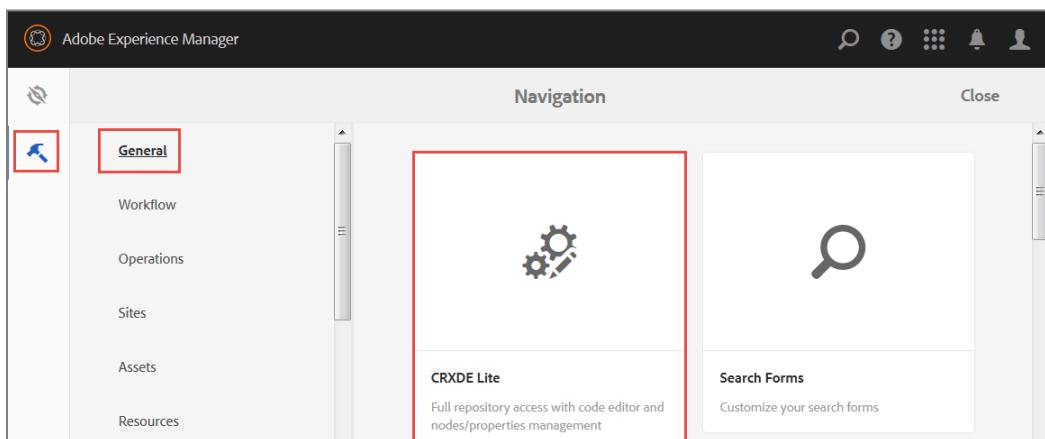


CRXDE Lite

The CRXDE Lite console is embedded into Adobe Experience Manager and allows you to perform common development and administration tasks in the browser. Because it is embedded in the server and is always available, CRXDE Lite is often the preferred tool for administrators and developers for working with nodes and properties in the JCR. It gives quick and direct access to the repository for monitoring, configuration, and development.

NOTE: You will use this interface frequently in this training.

You can access this console through <http://localhost:4502/crx/de/index.jsp> or by navigating to **Tools > General > CRXDE Lite** from the UI, as shown:



With CRXDE Lite, you can create a project, create and edit files (like .jsp and .java), folders, templates, components, dialogs, nodes, properties, and bundles. CRXDE Lite is recommended for most repository-level administration tasks as well as many light weight development tasks.

The CRXDE Lite console contains:

Top switcher bar: Enables you to quickly switch between CRXDE Lite, Package Manager, and Package Share.

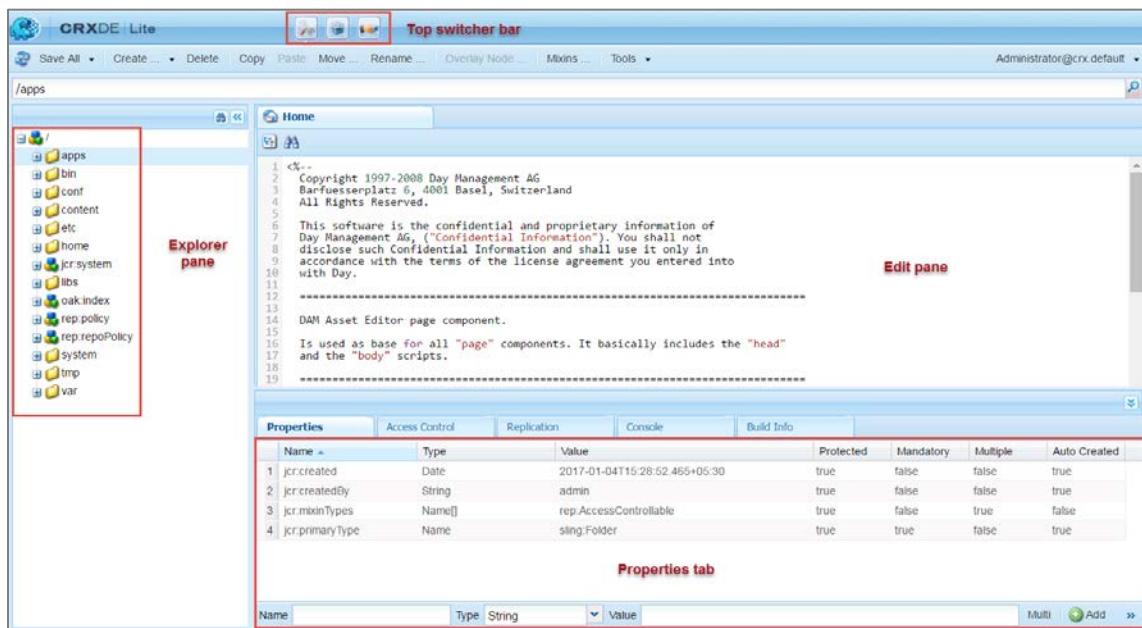
Explorer pane: Displays a tree of all the nodes in the repository.

You can perform the following actions on a node in the tree:

- Select the node and view its properties in the **Properties** tab. Examine all the JCR properties of different nodes.
- Right-click the node and perform an action on it, such as renaming the node, creating a new node, creating a folder, creating a file.

Edit pane: Double-click a file in the Explorer pane to display its content. For example, a .jsp or a .html file. You can then modify the code and save the changes.

Properties tab: Displays the properties of the node that you selected. You can add new properties or delete existing ones.



Working with Packages

A package is a zip file that holds repository content in the form of a file-system serialization called "vault" serialization. Packages provide an easy-to-use-and-edit representation of files such as pages, assets, and folders.

Throughout this training, you will be working with packages; either by creating your own or installing an available package. These packages enable you to import and export repository content. A package may contain:

Page-related content

Project-related content

Assets-related content

Vault meta information such as filter definitions and import configuration information

Other package information such as package settings, package filters, package screenshots, and package icons

Packages enable you to import and export repository content.

You commonly use packages for any of the following:

Install new functionality

Transfer content between instances

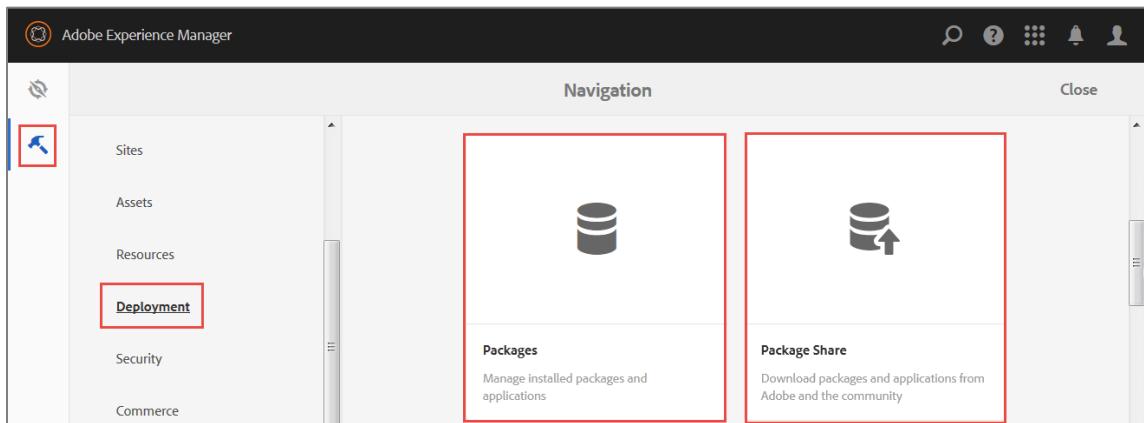
Back-up repository content

Export content to the local file system

You can work with packages in two ways:

1. **Package Manager**—can be used to import or export content, transfer content between instances, and back-up repository content. Using filters, you can create a package to contain page content, or project-related content. You can access this console through the following link: <http://localhost:4502/crx/packmgr/index.jsp>. With the Package Manager, you can perform the following common tasks:
 - Create, build, and download content packages
 - Upload and install packages
 - Modify existing packages
 - View package information
2. **Package Share**—a centralized server that contains both public and private packages available across all instances. Public packages may include hotfixes, new functionality, documentation, and so on. You can access this console through the following link: <http://localhost:4502/crx/packageshare/index.html>.

To access Package Manager and Package Share from the UI, navigate to **Tools > Deployment > Packages/ Package Share** as shown:



Creating and Building New Packages

When creating packages using Package Manager, you can apply rules and filters to determine the content package should extract from the repository. After you define the content, you can build it. A package is created in a .zip file and you can download it to your local file system. You can test the contents of the package before building it.

There are many options in Package Manager to work with packages, such as:

Rebuild: Helps rebuild the package if there is a change in the repository content.

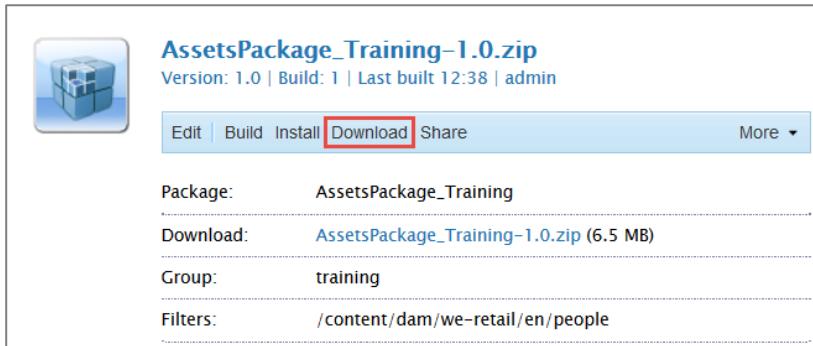
Edit: Helps edit filters or rules applied to the package.

Test: Helps perform a dry run of the installation.

Rewrap: Helps recreate the package with additional information such as thumbnails and icons.

Downloading packages to your file system

You can download a package by clicking the download link. This link displays when the package details are expanded. After downloading the package, you can unzip the contents



The screenshot shows a package details page for 'AssetsPackage_Training-1.0.zip'. The page includes a thumbnail icon, the package name, version, build, last built time, and administrator. A navigation bar at the top has links for Edit, Build, Install, Download (which is highlighted with a red box), Share, and More. Below the navigation, there are four data rows: Package (AssetsPackage_Training), Download (AssetsPackage_Training-1.0.zip (6.5 MB)), Group (training), and Filters (/content/dam/we-retail/en/people).

of the package onto your local system.

Typically, a content package contains the following folders:

- **jcr_root:** Represents the root node of the repository and contains the actual content of the package.
- **META-INF:** Contains metadata regarding node definitions and the filter.xml file that gives directions to FileVault about the paths to include.

Using the Package Share

Package Share is a centralized server where public and private packages are made available. These packages may be hotfixes, feature sets, updates, or Adobe Experience Manager content generated by other users. You can search, download, and install any package either to your instance or to your local file system.

Within the Package Share, you have access to the following:

- Adobe packages provided by Adobe
- Shared packages provided by others companies and made public by Adobe

Your company packages that are private

You can access the Package Share with this link:

<http://localhost:4502/crx/packageshare/login.html>

- You can directly access Package Share (without using CRXDE Lite) through the following link: <https://www.adobeacmcloud.com/content/packageshare.html>
-



NOTE: You must have an Adobe ID to use the Package Share. For more information, refer to the online documentation about Package Share.

LAB B - WORK WITH PACKAGES

Scenario

You are assigned to a project that requires your project artifacts like code, components, and so on, with several members of a team. To do this, you need to configure new development tools apart from the in-built Adobe Experience Manager capabilities.

Challenge

Identify the code editor for easy use, catering to developers comfortable with simple HTML editors. Identify the developer tool available in Adobe Experience Manager that enables packaging of developer artifacts for distribution.

Objectives

- Install a Package in Adobe Experience Manager

Prerequisites

- AEM installed on your machine:
 - Running Adobe Experience Manager Author instance
 - An Eclipse project from the AEM Archetype

Directions

Complete the exercises that follow.

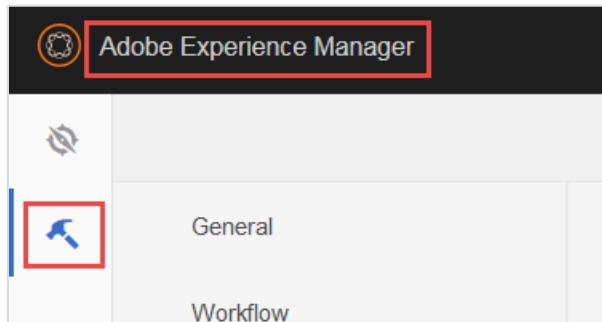
Exercise 1 – Install a package in AEM

Overview:

In this exercise, you will install a sample package in Adobe Experience Manager.

Steps:

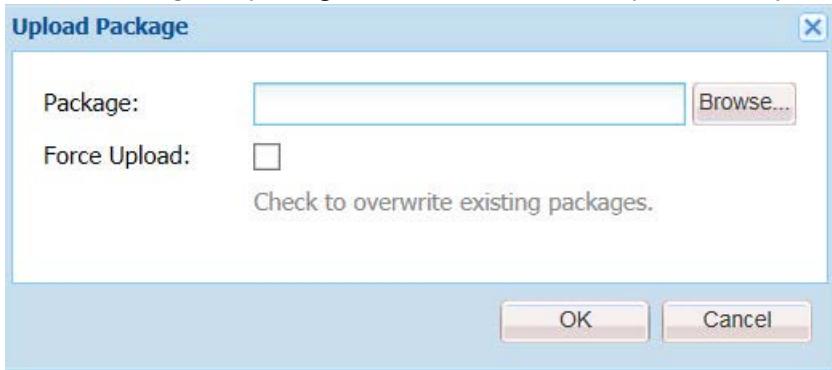
1. Navigate to the Navigation console, or click the following link: <http://localhost:4502>
2. Click **Adobe Experience Manager** in the upper-left, then click the **Tools** icon.



3. Under the **Deployment** section, select **Packages**. This opens the Package Manager of CRXDE Lite. As an alternative, click the following link to open the page directly: <http://localhost:4502/crx/packmgr/index.jsp>
4. Click **Upload Package**.



5. In the **Upload Package** dialog box, click **Browse**, and select the **SamplePackage.zip** package from the Exercise_Files provided to you. Click **OK**.

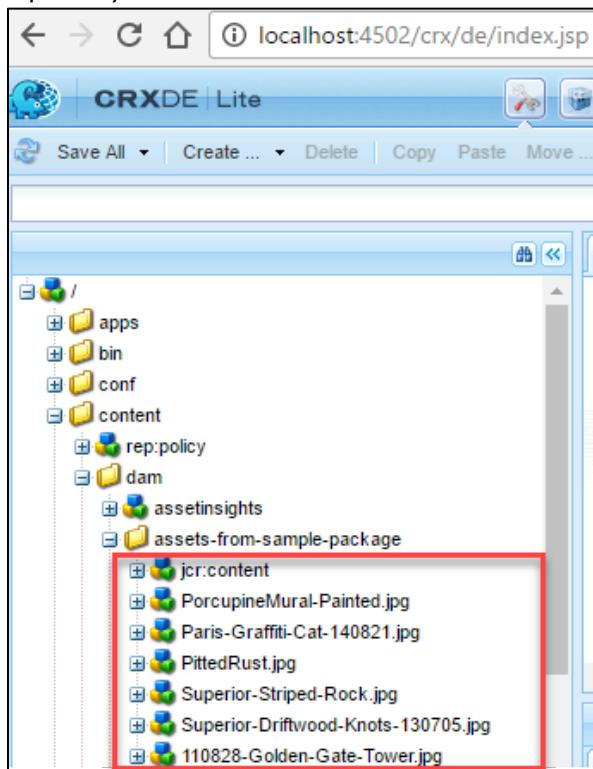


6. After the package is uploaded, click **Install**.



7. In the **Install Package** dialog box, ignore the Advanced Settings area and click **Install**.
8. Check the **Activity Log**. You can see the content was added from the package.
9. Check the **/content/dam/assets-from-sample-package** folder in CRXDE Lite to see the changes reflected there. The contents of the package are added to the

repository.



Exercise 2 - Create, Build, and Download a Package

Overview:

In this exercise, you will create, build and download the sample package in Adobe Experience Manager.

Steps:

1. Click the Package Manager button in CRXDE Lite. This takes you back to the Package Manager.

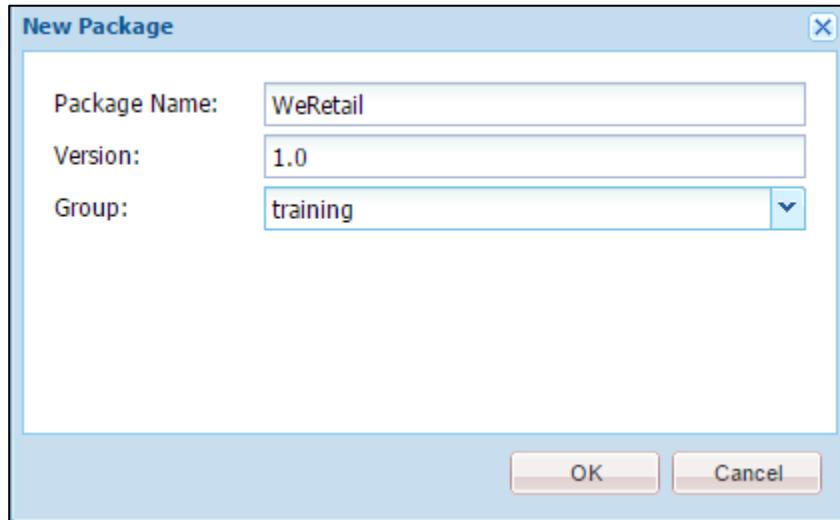


2. As an alternative, click the following link to open the page directly:
<http://localhost:4502/crx/packmgr/index.jsp>
3. Click Create Package.

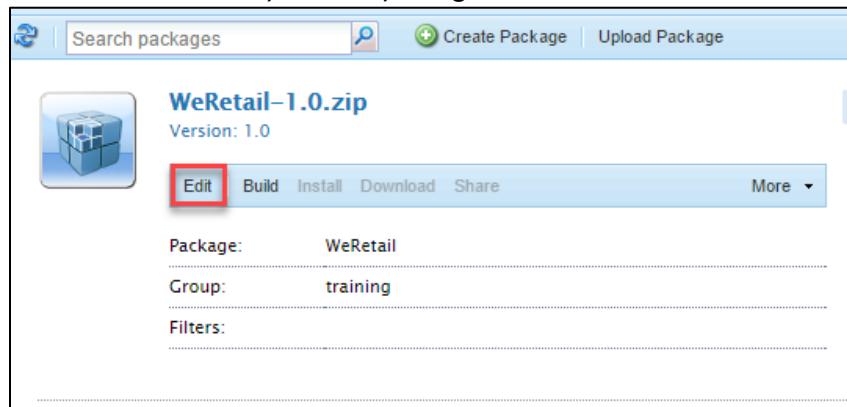


In the **New Package** dialog box, enter the following details:

Property	Value
Package Name	WeRetail
Version	1.0
Group	training

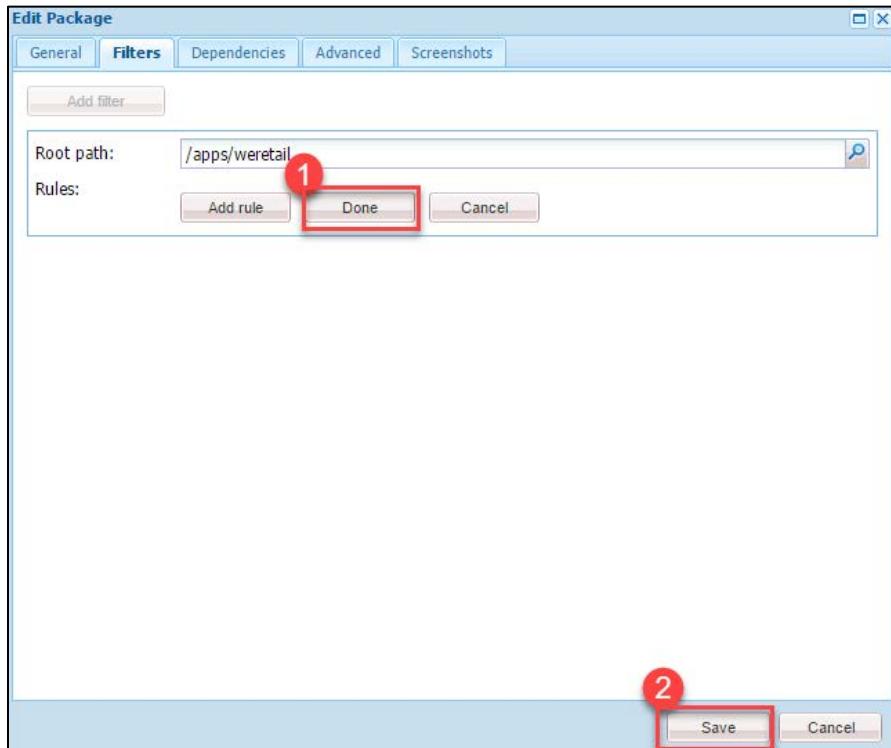


4. Click **OK**.
5. Click **Edit** on the newly-created package.



6. To add filters to the package, click the **Filters** tab, and click **Add Filter**.
7. For the **Root** path, browse and select the **WeRetail** folder under **apps**, and click **OK**.

8. Click **Done**, and then click **Save**.



9. Click **Build** to build the package, and then click **Build** again in the confirmation dialog box.



10. The package is now ready for download. Click the **Download** link to download a copy of the package to your computer.

4 CONFIGURING THE DEVELOPMENT ENVIRONMENT

For this portion of the course, you will use the following development tools:

- Maven: This tool builds and deploys bundles to the JCR.
- Eclipse: This tool lets you edit code, build OSGi bundles, and perform Unit tests.

The following sections guide you through the installation and configuration of Maven and Eclipse and have your development environment ready to start working on your projects.

Working with Maven

Maven is a tool you can use to build and manage Java-based projects. It specifies how software is built, as well as its dependencies. Maven has a central repository from which it can dynamically download Java libraries and plugins, and store them in a local cache. This cache can also be updated with artifacts created by local projects. Public repositories can also be updated.

Maven projects are configured using a Project Object Model (POM) stored in a pom.xml file.

Advantages of Using Maven

The main objective of Maven is to allow a developer to comprehend the complete state of a development effort in a minimal amount of time. With Maven, you have the following advantages:

- Makes the build process easier
- Provides a uniform build system
- Provides quality project information
- Provides guidelines for best practices development
- Allows transparent migration to new features

Contents of a POM File

POM files are XML files that contain the identity and structure of the project, build configuration, and other dependencies. Being in the root folder of your project, it identifies the project as a Maven project. A typical POM file includes:

- General project information: This includes the project name, website URL, and organization. It can also include a list of developers and contributors along with the license for a project.
- Build settings: This section includes the directory settings of source and tests, plugins, plugin goals, and site-generation parameters.
- Build environment: This consists of profiles that can be activated for use in different environments. The build environment customizes the build settings for a specific environment and is often supplemented by a custom `settings.xml` file in the Maven repository.

A Maven repository is a directory that stores all project files, including JAR files, plugins, artifacts, and other dependencies.

- POM relationships or dependencies: There may be interdependencies between projects, with their POM settings being inherited from parent POMs. These details are described in the POM relationships section and include:
 - groupId
 - artifactId
 - Version
 - Dependencies

Maven project POMs extend a POM called the Super POM, which is in \${M2_HOME}/lib/maven-xxx-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 has an ID value of central. By default, all Maven clients are configured to read from this repository, and it is also the repository of the default plugins. You can override these by configuring a custom settings.xml file.

- Snapshot versions: Snapshot versions are indicated by the string '-SNAPSHOT'. These are used for projects under active development. If a 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 the repository when you run a build.
- Property references: These refer to properties from another part of the POM file, Java system properties, or implicit environment variables. It supports three environment variables:
 - env
 - project
 - settings

- Plugins: Plugins provide functionalities such as compiling source, packaging a WAR file, or running JUnit tests. These plugins are retrieved from the Maven repository. If new functionality is added to a plugin, you can make use of it by updating the version number of the plugin in a single POM configuration file.

The UberJar

The UberJar is a special JAR file provided by Adobe. The main goal of using the UberJar is to reduce the number of dependencies. Earlier, for every API being used, a separate and individual dependency had to be added to the project. This is avoided by using the UberJar.

The UberJar file contains:

- all public Java APIs exposed by Adobe Experience Manager.
- limited external libraries—all public APIs available in Adobe Experience Manager, which comes from Apache Sling, Apache Jackrabbit, Apache Lucene, Google Guava, and two libraries used for image processing.
- interfaces and classes exported by an OSGi bundle in Adobe Experience Manager.
- MANIFEST.MF file with the correct package export versions for all exported packages.

Using the UberJar

To use the UberJar, you must add the following elements to your pom.xml file.

- Dependency element: to add the actual dependency to your project

```
1. <dependency>
2.   <groupId>com.adobe.aem</groupId>
3.   <artifactId>uber-jar</artifactId>
4.   <version>6.2.0</version>
5.   <classifier>obfuscated-apis</classifier>
6.   <scope>provided</scope>
7. </dependency>
```

- Repository element

```
1. <repositories>
2.   <repository>
3.     <id>adobe-public-releases</id>
4.     <name>Adobe Public Repository</name>
5.     <url>https://repo.adobe.com/nexus/content/groups/public/</url>
6.     <layout>default</layout>
7.   </repository>
8. </repositories>
9. <pluginRepositories>
10.  <pluginRepository>
11.    <id>adobe-public-releases</id>
12.    <name>Adobe Public Repository</name>
13.    <url>https://repo.adobe.com/nexus/content/groups/public/</url>
14.    <layout>default</layout>
15.  </pluginRepository>
16. </pluginRepositories>
```

If you already use a Maven Repository Manager such as Sonatype Nexus, Apache Archiva, or JFrog Artifactory, add the appropriate configuration to your project to reference this repository manager; and add Adobe's Maven Repository to your repository manager.

Installing and Configuring Eclipse

Eclipse is an open source Integrated Development Environment (IDE) used to edit your project source locally on your file system. While working with Adobe Experience Manager projects, once you install Eclipse, you will also need to install the following plugins:

- Maven Integration for Eclipse (M2E)—a Maven plugin
 - Note: Most eclipse versions come with this Maven plugin already.
- AEM plugin for eclipseMaven comes embedded in Eclipse with the M2E plugin.

Setting up the AEM Plugin for Eclipse

The AEM plugin for Eclipse is a plugin based on the Eclipse plugin for Apache Sling. It has the following benefits:

- Synchronizes both content and OSGi bundles
- Supports debugging and code hot-swapping
- Includes a project creation wizard to simplify bootstrapping of Adobe Experience Manager projects
- Enables edition of JCR properties
- Seamlessly integrates with Adobe Experience Manager instances through Eclipse Server Connector



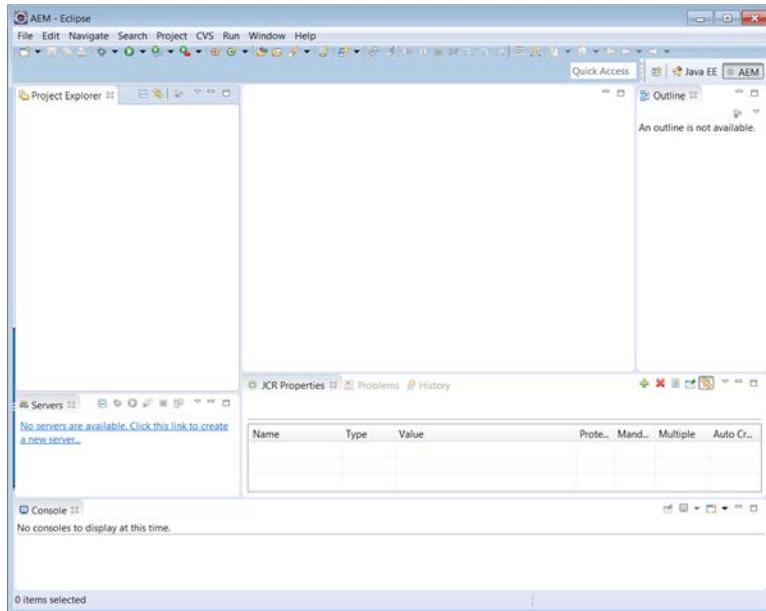
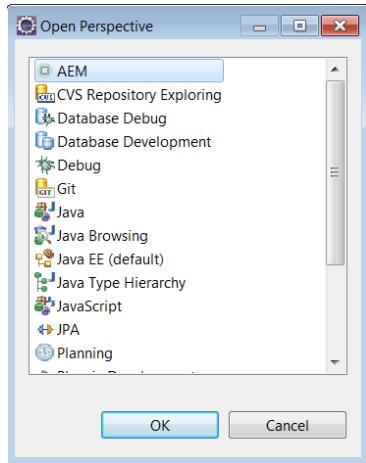
Perform Task – [Exercise 1 – Install and configure eclipse \(Optional\)](#), from Lab C.



Perform Task – [Exercise 2 – Install and Configure Eclipse AEM PLUGIN \(optional\)](#), from Lab C.

Configuring the AEM Perspective

The AEM perspective offers complete control over all your AEM projects and instances.



The AEM perspective makes the AEM Console as well as the JCR properties view available to you, allowing you to add and modify nodes and properties in your Adobe Experience Manager project.



Perform Task –[Exercise 3 – Create an AEM project using Maven Archetypes](#), from Lab C.

Configuring the AEM Console

The console displays the progress of repository synchronization. That is, you can see the progress of all check-in and check-out of nodes and properties to and from the repository.

To configure the AEM console, in the **Console** tab, click the **Open Console** icon, and select **AEM Console**. If the Console tab is not visible, select **Window > Show View > Console** then click **Open Console** and select **AEM Console**.



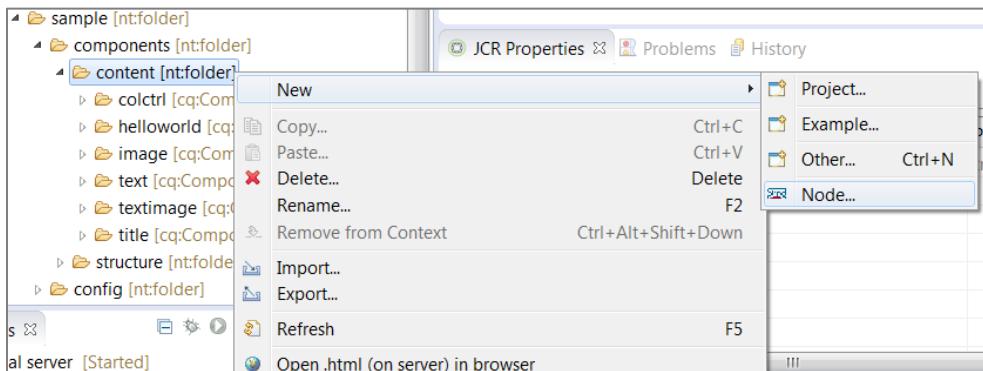
Working with JCR

With the AEM perspective, you can do the following manipulations to the JCR:

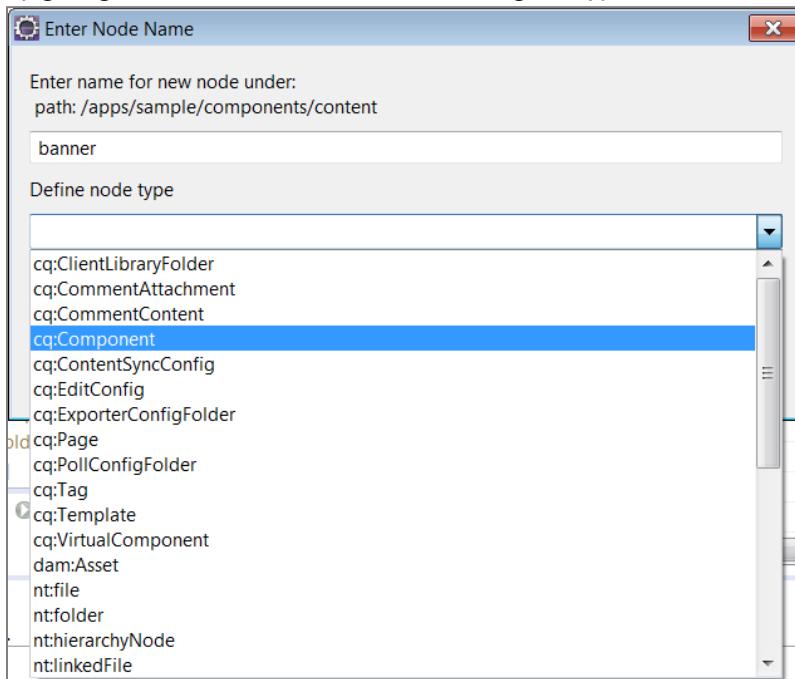
- Add a new node.
- Add or edit properties of a node.

Adding a New Node

You can add a new node to the repository:



- By giving a name for the node, and selecting the type of node:



When you click on a node in the Project Explorer, you can check its properties in the JCR Properties view as shown.

JCR Properties

Name	Type	Value	Prote...	Mand...	Multiple	Auto Cr...
jcr:primaryType	Name	cq:Component	true	true	false	true

Adding or Editing a Node Property - You can add a property by selecting the Insert a property icon from the toolbar. Once you add the details of the property, press Enter to save the changes. You can right-click on the property, and select Show in editor to view the underlying vault file in the editor window.

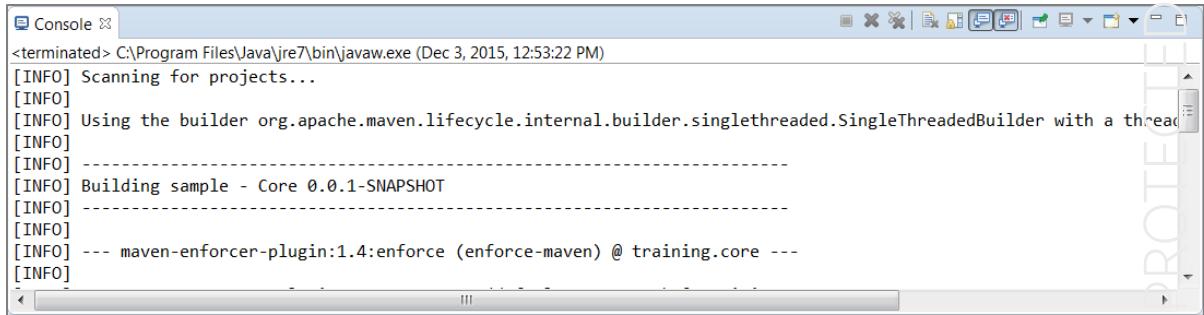
JCR Properties

Name	Type	Value	Prote...	Mand...	Multiple	Auto Cr...
jcr:primaryType	Name	cq:Component	true	true	false	true
jcr:title	String	Banner	false	false	false	false

Building and Deploying Your Project Using Maven

Once you complete a part of your development, you may want to manually build and deploy your content package or OSGi bundle. To do this, you can select the project file from the Project Explorer, navigate to Run > Run As > Maven Install.

You can check the progress of the command in the Console.



The screenshot shows a standard Eclipse IDE console window titled "Console". The output pane displays the following Maven build log:

```
<terminated> C:\Program Files\Java\jre7\bin\javaw.exe (Dec 3, 2015, 12:53:22 PM)
[INFO] Scanning for projects...
[INFO]
[INFO] Using the builder org.apache.maven.lifecycle.internal.builder.singlethreaded.SingleThreadedBuilder with a thread pool size of 1
[INFO]
[INFO] -----
[INFO] Building sample - Core 0.0.1-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-enforcer-plugin:1.4:enforce (enforce-maven) @ training.core ---
[INFO]
```

For more information on the AEM plugin, see <https://docs.adobe.com/docs/en/dev-tools/aem-eclipse.html>

Setting up your project

You now have all your development tools ready, and it is time to set up your project. In the following sections, you will learn the following:

- How to create an Adobe Experience Manager project using the AEM Archetype

Creating an Adobe Experience Manager Project Using the AEM Archetype

Every Adobe Experience Manager project you create has the following modules:

- **Core:** This is a Java bundle that contains all core functionality such as OSGi services, listeners, schedulers, as well as component-related Java code such as servlets or request filters.
- **Apps:** This module contains the /apps and /etc parts of the project, such as JS and CSS clientlibs, components, templates, runmode specific configs, as well as Hobbes tests.
- **Content:** This module contains sample content using the components from the apps module.
- **Tests:** This module is a set of Java bundles that contain JUnit tests that are executed on the server-side.
- **Launcher:** This module contains code that deploys the tests bundle to the server and triggers the remote JUnit execution.

Once your project is set up, you can use the Export to Server and Import to Server options to synchronize content between Eclipse and the Adobe Experience Manager server.



Perform Task – [Exercise 4 – Build and Deploy to Adobe Experience Manager](#), from Lab C.

LAB C – CONFIGURE THE DEVELOPMENT ENVIRONMENT

Scenario

Your team uses a central repository for storing all development artifacts. This repository will have the latest working code you can download into your local system. As such, at the end of the day, the code you develop will be tested and pushed to the repository. The result of this process is all teams can always share code constantly and continuously, with any conflicts being detected at an early stage.

Challenge

You need to verify you have the appropriate plugins and tools required for building an application using Eclipse.

Objectives

- Install and Configure Eclipse (Optional)
- Install and configure AEM Plugin (Optional)
- Create an AEM Project using Maven Archetypes
- Build and Deploy to Adobe Experience Manager
- Configure AEM Server in Eclipse

Prerequisites

- AEM installed on your machine:
 - Running Adobe Experience Manager Author instance

Directions

Complete the exercises that follow.

Exercise 1 – Install and configure Eclipse (Optional)

Overview

In this exercise, you will install and configure Eclipse. Eclipse is an open source software application used to edit the project source locally on your file system.



Note: You can skip this exercise if you use a Ready Tech Environment since Eclipse is already installed as part of the image.

IMPORTANT! Please use **Eclipse Luna** for this course. For this class, you must use the "Luna" version of Eclipse provided.

Steps:

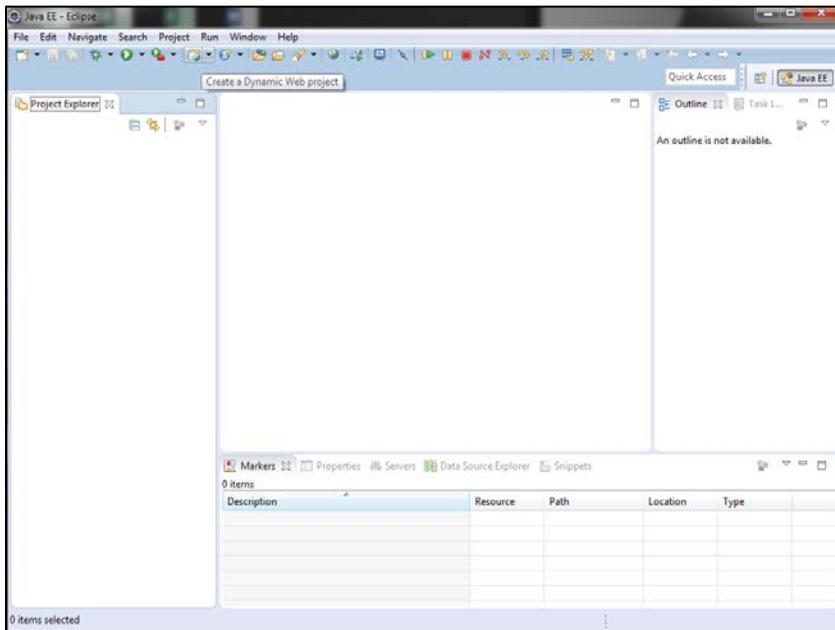
1. Extract files from the Distribution_Files.zip file to the destination directory of your choosing to begin the installation.
 - a. Navigate to the directory where you extracted the contents of the Eclipse installation zip file. For example, **C:\Program Files\Eclipse** on Windows, or **Applications/Eclipse** on the Mac. On the Mac, you may need to use the **sudo** command to do this.
 - b. Double-click **eclipse.exe** (or **eclipse.app**) to start Eclipse.
 - c. In the Workspace Launcher dialog:
 - Accept the default location for workspace
 - Check the box for "Use this as the default and do not ask again"
 - Click **OK**



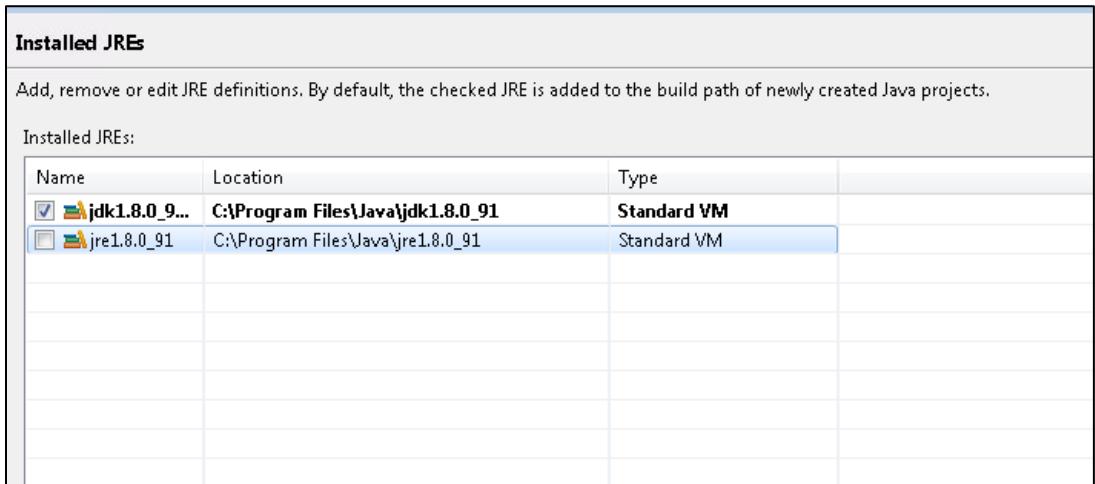
NOTE: You can only run a single instance of Eclipse at a time. When open, Eclipse creates a workspace where you can perform your development tasks. You cannot have two Eclipse workspaces open at the same time. If you try to open Eclipse again, while Eclipse is already open, a “Workspace Unavailable” dialog box opens, indicating a workspace is already in use or cannot be created.

- Click the **Workbench** logo in the upper-right corner to close the welcome screen and go to the main workbench.

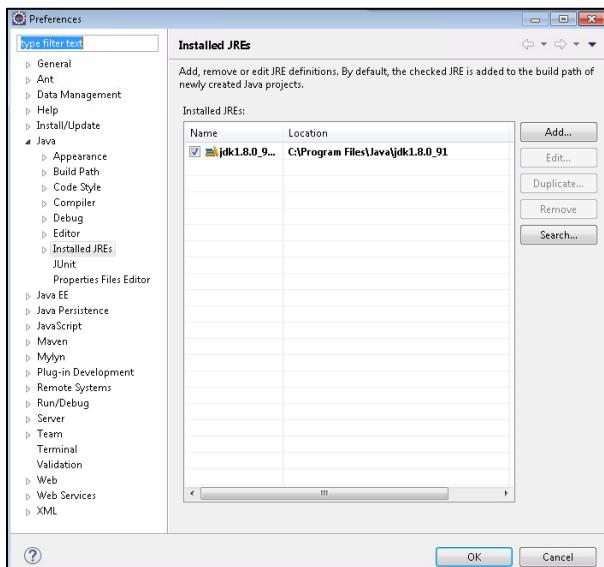




2. Set Eclipse's JRE.
 - a. Make sure Eclipse Luna is using the correct JDK. This may be set to a JRE, not a JDK.
 - b. Click Window > Preferences > Java > Installed JREs.
 - c. You should see something like the following path, but if not, you must provide the correct directory path to your JDK.



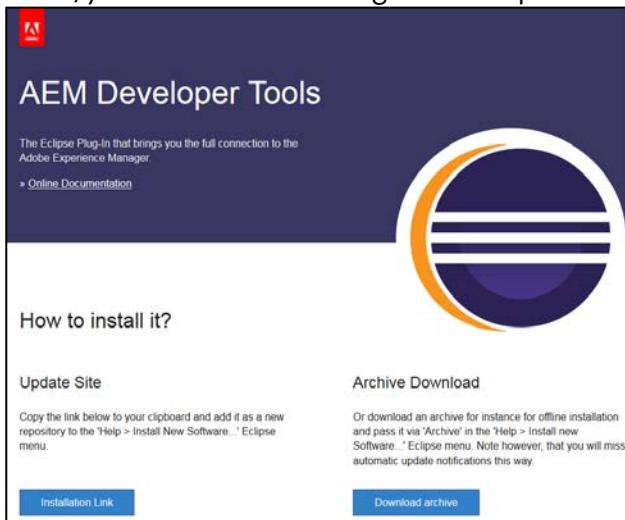
3. Remove the reference to the jre if it exists.
 - a. Select jre8 and then click Remove on the right side of the window.
 - b. When done, click OK and verify your settings should appear as follows (remember to use JDK 1.8):



Exercise 2 – Install and Configure Eclipse AEM PLUGIN (optional)

Overview

In this exercise, you will install and configure the Eclipse AEM plugin.



The AEM Eclipse plugin can be downloaded at <https://eclipse.adobe.com/aem/dev-tools/>.

For this exercise we will use a copy of the install archive provided in the Exercise Files.

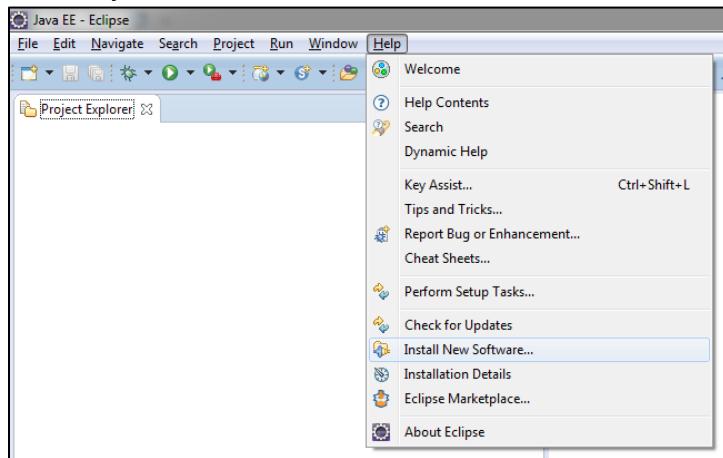


Note: You can skip this exercise if you use a Ready Tech Environment since AEM is already installed as part of the image.

Steps:

1. Open Eclipse Luna by double-clicking on **eclipse.exe** (or **eclipse.app**).

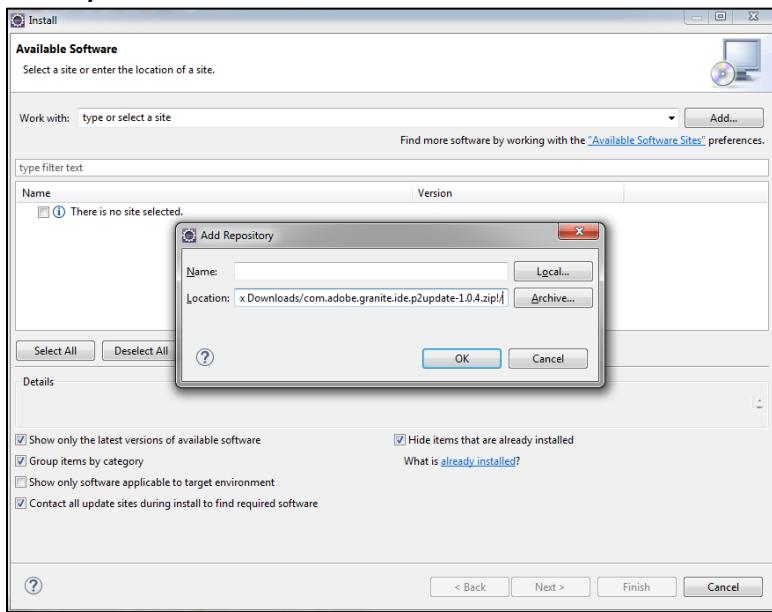
- a. Click **Help > Install New Software...**



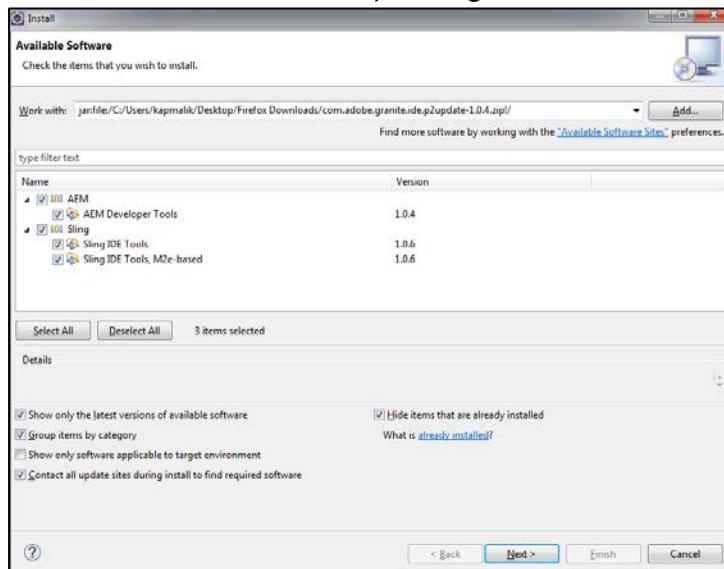
- b. Click **Add...**
- c. In the **Add Repository** popup, click **Archive...**; navigate to the repository archive and select the zip file (**com.adobe.granite.ide.p2update-1.0.4.zip**) provided for the plugin from **/Exercise_Files/04_Dev_Environment/**.

Tip: In a ReadyTech environment, look for the Supported Applications folder on the desktop and navigate to the following path: Supported Applications\Common\eclipse\plugins.

d. Click **Open** then click **OK**.



2. Select the Tools listed in the window by clicking **Select All** then click **Next**.

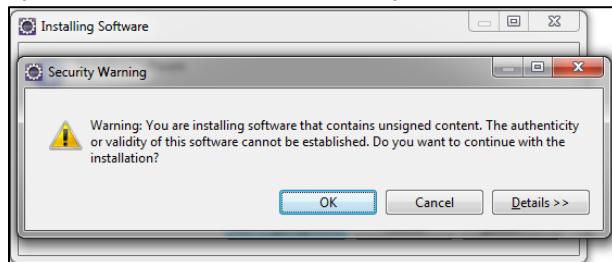


- An Install Details window opens, allowing you to review the tools you are about to install.
3. Click **Next**, which opens the Review Licenses screen.
 4. Click the radio button for "**I accept the terms of the license agreements**" then click **Finish**. An Installing Software dialog opens with a progress bar. The installation should only take a minute or two.



NOTE: Do not click Run in Background.

-
- In case you receive the following security warning, click **OK** to continue the installation. (After you click **OK**, the Installing Software dialog will re-open until the installation is completed.)

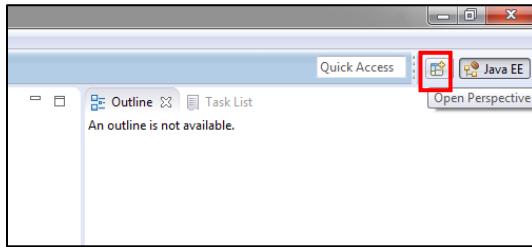


5. Click **Yes** to restart Eclipse to load the newly installed tools.

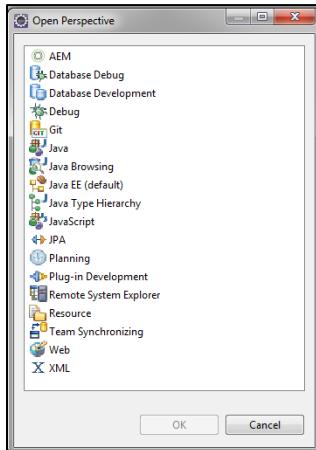


Note: This only takes about a minute. (If you see a dialog that asks if you want to keep the current location of your Eclipse install, click **OK**.) Eclipse opens.

6. Click **Workbench** in the upper-right corner again.
- Notice the new **AEM** perspective available in Eclipse. To check this, click the **Open Perspective** icon as shown.

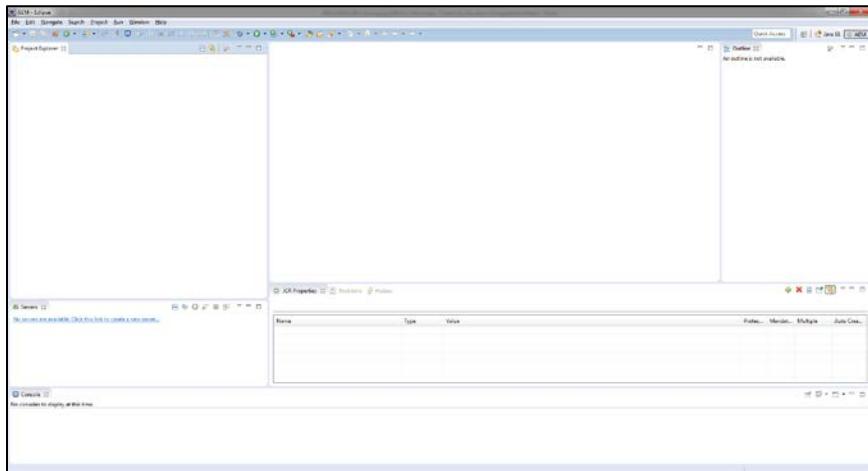


- Notice how the AEM perspective is at the top:



- Select **AEM** and click **OK**.

You will see a new perspective opened in Eclipse as follows:



7. Keep Eclipse open for the next exercise.

Review:

- At this point, you installed and configured Eclipse and the Eclipse AEM plugin.

These tools will help you work on Adobe Experience Manager projects in a team environment.

Exercise 3 – Create an AEM project using Maven Archetypes

Overview

In this exercise, you will create an AEM project using the maven archetypes.

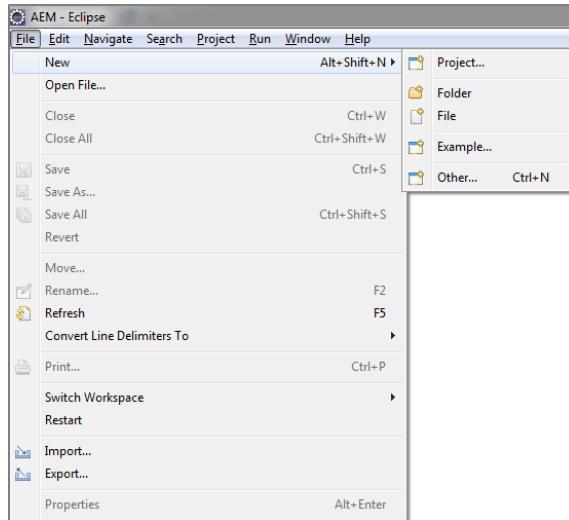
Steps:

1. Create a new project in Eclipse.

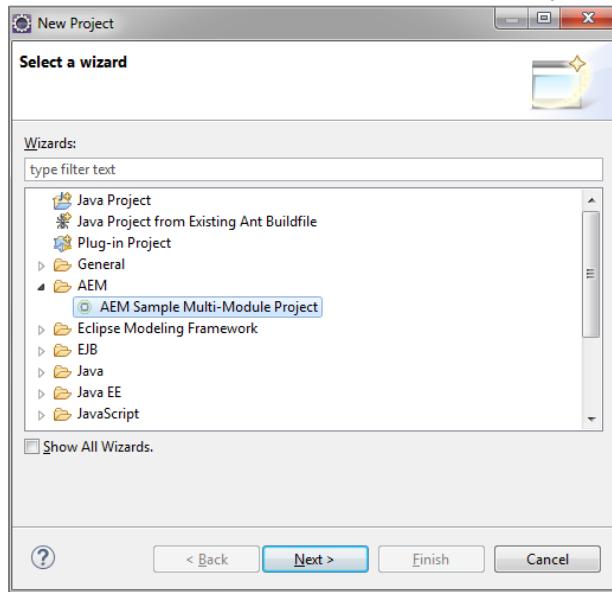


Note: Because we already installed the AEM plugin and configured the archetype in Eclipse, we must now make use of them and create an AEM project.

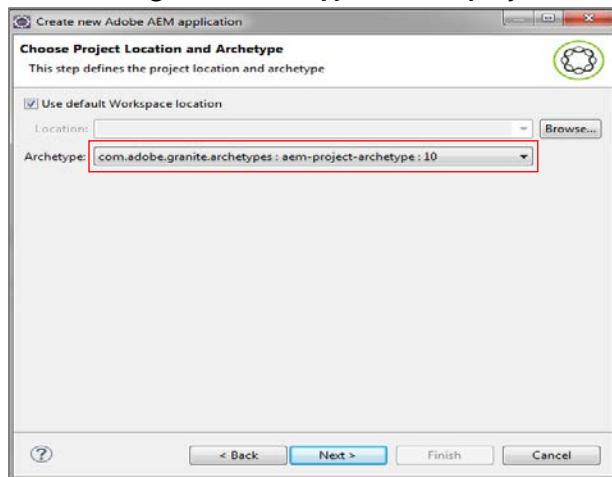
- a. If not open, open Eclipse and click **File > New > Project...**



- b. Select AEM > AEM Sample Multi-Module Project, and click Next.

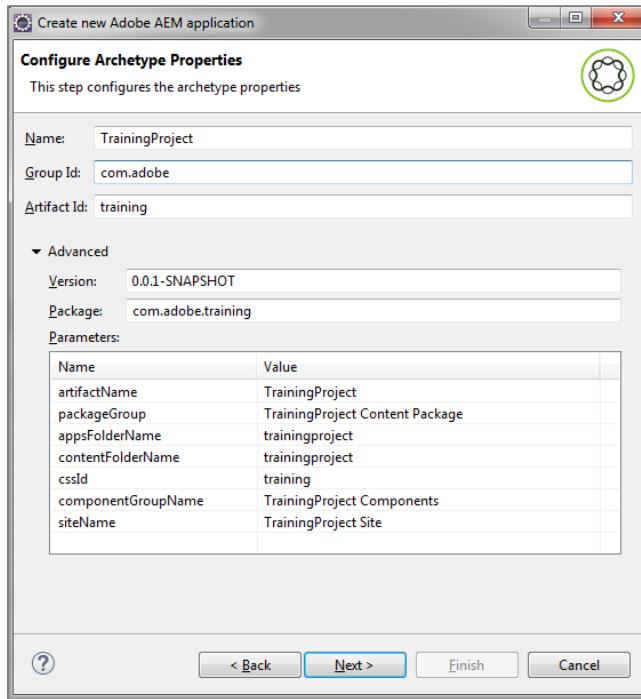


- c. If not already preselected, click the Archetype drop-down menu, and select com.adobe.granite.archetypes : aem=project-archetype:10 then click Next.

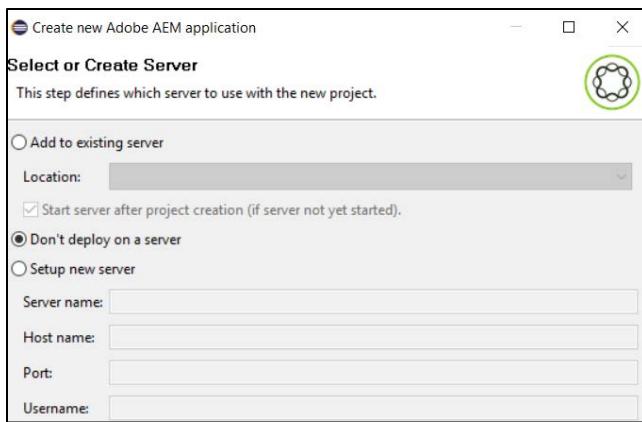


- d. In the Configure Archetype Properties window, replace the project details with:

- Name: **TrainingProject**
- Group Id: **com.adobe**
- Artifact Id: **training**

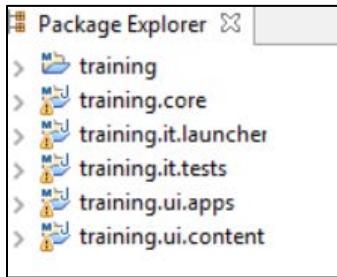


- e. Click **Next**.
- f. In the Create new Adobe AEM application dialog, select **Don't deploy on a server** then click **Finish**



Note: Wait for the process to complete. This might take a few minutes. If you see a "Not Responding" message, hold on – as it will complete the process.

g. Verify the project was created:



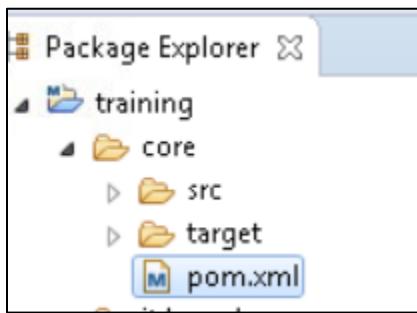
2. Resolve javax.inject and uber-jar dependencies for development.

a. Fix the **javax.inject** version dependency issue.



Note: training.core bundle does not resolve on AEM 6.3 due to different version of javax.inject. With AEM 6.3, the javax.inject package is exported with version 1.0.0 by org.apache.geronimo.specs.geronimo-atinject_1.0_spec.

- b. Add the following dependencies to the training/core/pom.xml to make javax.inject work in AEM 6.3.



```
<Import-Package>javax.inject;version=0.0.0,*</Import-Package>
```

to the <instructions> block of the maven-bundle-plugin as shown:

```
<plugin>
    <groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
    <extensions>true</extensions>
    <configuration>
        <instructions>
            <!-- Import any version of javax.inject, to allow running on multiple versions of AEM -->
            <Import-Package>javax.inject;version=0.0.0,*</Import-Package>
            <Sling-Model-Packages>
                com.adobe.training.core
            </Sling-Model-Packages>
        </instructions>
    </configuration>
```



Note: AEM Maven dependencies update through the UberJar. The "UberJar" JAR file contains all of the public Java APIs exposed by AEM. It includes

limited external libraries as well, specifically all public APIs available in AEM, which come from Apache Sling, Apache Jackrabbit, Apache Lucene, Google Guava, and two libraries used for image processing.

The UberJar only contains API interfaces and classes, meaning it only contains interfaces and classes exported by an OSGi bundle in AEM. It also contained a MANIFEST.MF file containing the correct package export versions for these exported packages, thus ensuring projects built against the UberJar have the correct package import ranges.

-
3. Replace the dependency to the aem-api package by the latest uber-jar.

- a. In the parent POM (training/pom.xml), replace this block:

```
<dependency>
  <groupId>com.adobe.aem</groupId>
  <artifactId>aem-api</artifactId>
  <version>6.0.0.1</version>
  <scope>provided</scope>
</dependency>
```

With:

```
<dependency>
  <groupId>com.adobe.aem</groupId>
  <artifactId>uber-jar</artifactId>
  <version>6.3.0</version>
  <classifier>apis</classifier>
  <scope>provided</scope>
</dependency>
```

- Save the file, and do the same for the other POMs (without <version> and <scope> blocks).



Note: This is only needed for the core POM; you can remove the dependency block for the other POMs.

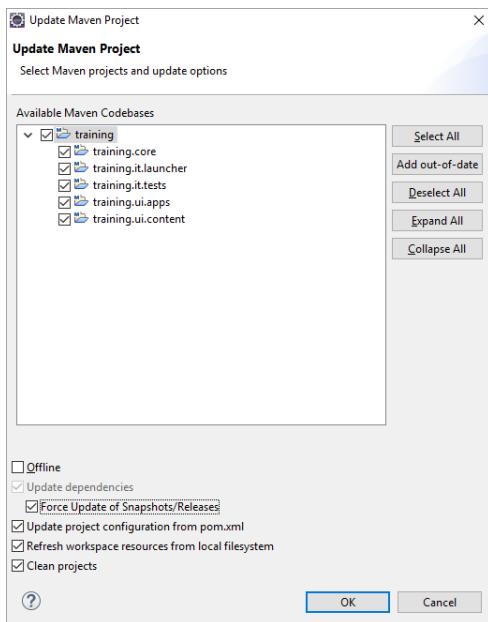
4. Alternatively, you can replace the POM Files (optional step).

Note: You can skip this step if you have already done step 3.

- a. Replace the following POM files in your Eclipse project with the files provided as part of the /Exercise_Files/- the folder is divided by individual chapters.
- b. In this case, you would search for the corresponding pom.xml file in /Exercise_Files/ 04_Dev_Environment > POM_Files.
 - Find and replace the contents of **training > it.tests > pom.xml** with /Exercise_Files/Files/04_Dev_Environment/POM_Files/**it-tests_pom.xml**
 - Find and replace the contents of the **parent POM** (training > pom.xml) with /Exercise_Files/Files/04_Dev_Environment/POM_Files/**pom.xml**
 - Find and replace the contents of **training > core > pom.xml** with /Exercise_Files/Files/04_Dev_Environment/POM_Files/**core_pom.xml**
 - Find and replace the contents of **training > ui.apps > pom.xml** with /Exercise_Files/Files/04_Dev_Environment/POM_Files/**ui-apps_pom.xml**
 - Find and replace the contents of **training > ui.content > pom.xml** with /Exercise_Files/Files/04_Dev_Environment/POM_Files/**ui-content_pom.xml**

5. Update the project

- a. Right-click **training**, and go to **Maven > Update Project**.
- b. Verify your codebase is selected (**training**).
- c. Select Force Update of Snapshots/Releases before clicking **OK**.



➤ Ignore any errors. We will fix them later.

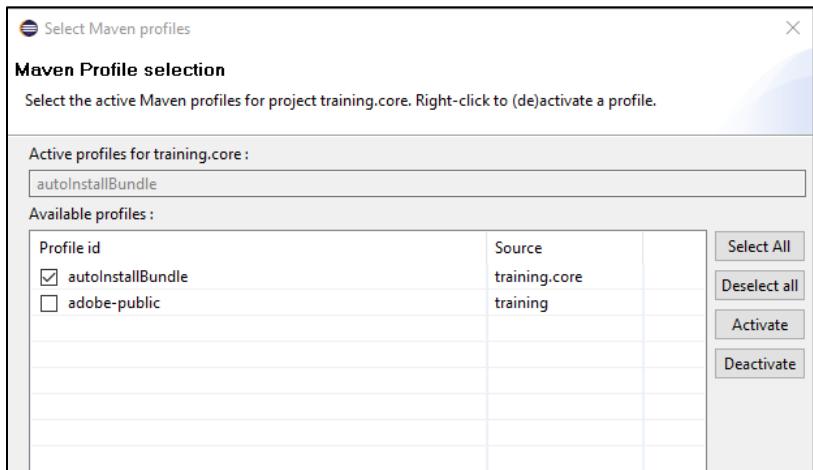
Exercise 4 – Build and Deploy to Adobe Experience Manager

Overview

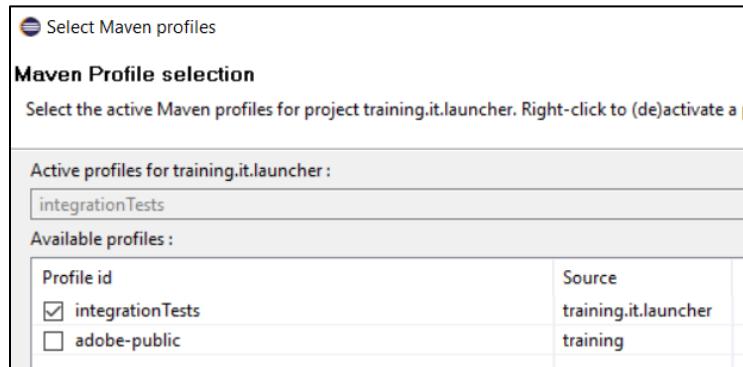
In this exercise, you will build the project and deploy it to the AEM server. You will do this by setting up the Maven profile for each one of your modules and then deploying the whole project.

Steps:

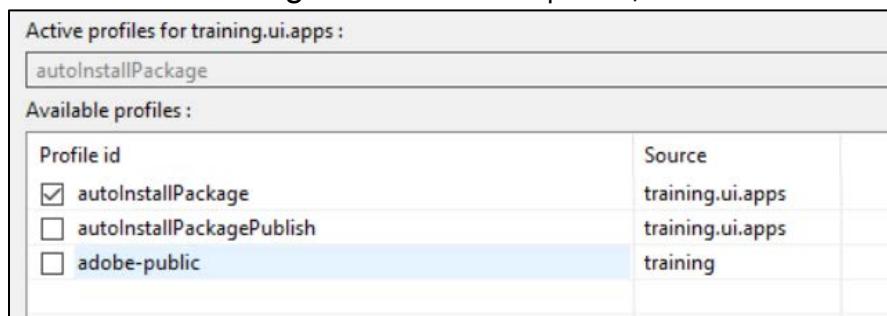
1. Set up the bundle.
 - a. Right-click **training.core**, and go to **Maven > Select Maven Profiles...**
 - b. In the Select Maven Profiles dialog box, select **autoInstallBundle** from the Available profiles, and click **OK**.



2. Set up the integrationTests.
 - a. Right-click **training.it.launcher**, and go to **Maven > Select Maven Profiles..**
 - b. Select **integrationTests** from the Available profiles, and click **OK**.



3. Set up the autoinstall package.
 - a. Right-click **training.ui.apps**, and go to **Maven > Select Maven Profiles...**
 - b. Select **autoInstallPackage** from the Available profiles, and click **OK**.



- c. Right-click **training.ui.content**, and go to **Maven > Select Maven Profiles...**
 - d. Select **autoInstallPackage** from the Available profiles, and click **OK**.
-

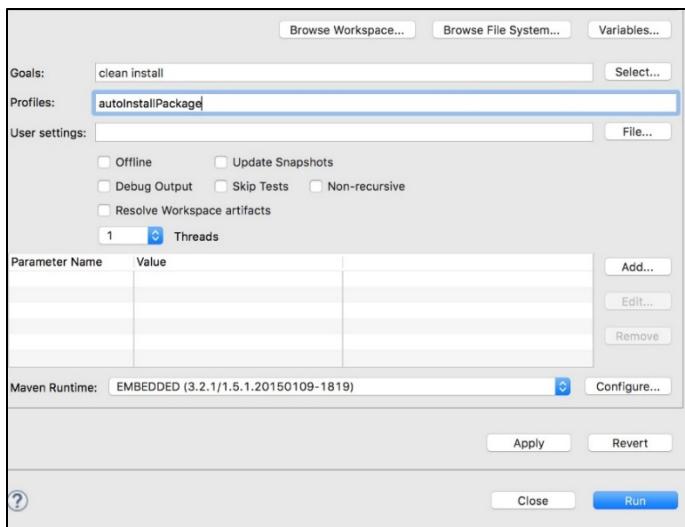


Note: You have now set up your Maven profiles.

- 4. Deploy your project into AEM.
 - a. Right-click **training** (the parent directory), and choose **Run As > Maven build...**
 - b. In the Edit configuration and launch dialog box, enter the Goals as **clean install** and in the Profiles field, enter: **autoInstallPackage**
-



Note: It is required to run the **autoInstallPackage** profile to ensure you have the correct content pages built within the repository. If you instead tried to run **autoInstallBundle** first, the build would fail to upload the bundle into Apache Felix.



- c. Click **Apply**, and then click **Run**.



Note: When first run, Maven will download all required dependencies locally to a .m2 directory. Depending on your Internet connection, this can take upwards of 15 minutes. Dependency download is a 1-time action if you do not add any new dependencies. Deploying to AEM after the initial deployment only takes a few seconds.

d. Verify the success message:

```
[INFO] --- maven-surefire-plugin:2.18.1:test (default-test) @ training.it.launcher ---
[INFO]
[INFO] --- maven-jar-plugin:2.5:jar (default-jar) @ training.it.launcher ---
[INFO] Building jar: C:\AEM\workspace123\training\it.launcher\target\training.it.launcher-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ training.it.launcher ---
[INFO] Installing C:\AEM\workspace123\training\it.launcher\target\training.it.launcher-0.0.1-SNAPSHOT.jar to
[INFO] Installing C:\AEM\workspace123\training\it.launcher\pom.xml to C:\Users\prpurush\.m2\repository\com\ad
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] training ..... SUCCESS [ 0.938 s]
[INFO] TrainingProject - Core ..... SUCCESS [ 4.635 s]
[INFO] TrainingProject - UI apps ..... SUCCESS [ 2.402 s]
[INFO] TrainingProject - UI content ..... SUCCESS [ 0.594 s]
[INFO] TrainingProject - Integration Tests Bundles ..... SUCCESS [ 0.500 s]
[INFO] TrainingProject - Integration Tests Launcher ..... SUCCESS [ 3.167 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
```

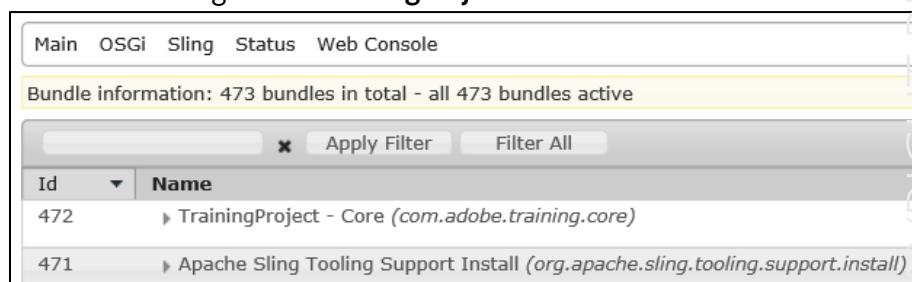
5. Verify the packages for both the **ui.apps** and **ui.content** content packages were installed in CRXDE lite and the core bundle installed in the Web Console.
 - a. Log in to AEM with the **admin** credentials.
 - b. Navigate to **Adobe Experience Manager > Tools > CRXDE Lite**.
 - c. Click the **Package** icon.



d. Verify the installed packages:



6. Go to **Web Console** (<http://localhost:4502/system/console/configMgr>) and select **OSGI > Bundles**.
 - a. Notice that the highest ID is **trainingProject.core**.



Main	OSGi	Sling	Status	Web Console
Bundle information: 473 bundles in total - all 473 bundles active				
<input type="button" value="x"/> Apply Filter <input type="button" value="Filter All"/>				
Id	Name			
472	▶ TrainingProject - Core (com.adobe.training.core)			
471	▶ Apache Sling Tooling Support Install (org.apache.sling.tooling.support.install)			

You have successfully deployed your project to AEM.

Exercise 5 – Configure the AEM Server in Eclipse

Overview

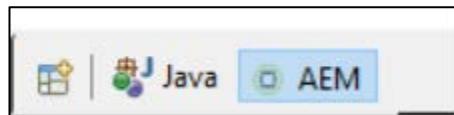
In this exercise, you will configure the AEM server in Eclipse to enable content synchronization. You can synchronize code (Java, HTML, jsp, css, and js) in Eclipse workspace with the code in the JCR. For example, assume you have application logic in Eclipse that represents a HTL component, like the Front End Brackets tool. You can synchronize the code in Eclipse with code in the Adobe Experience Manager JCR. That is, you can import code you write in Eclipse into the JCR. Likewise, if you make a change in the JCR using CRXDE Lite, you can export those changes to your Eclipse workspace (filesystem).

Sync from Eclipse to the JCR is “Exporting to the server” (auto or manual).

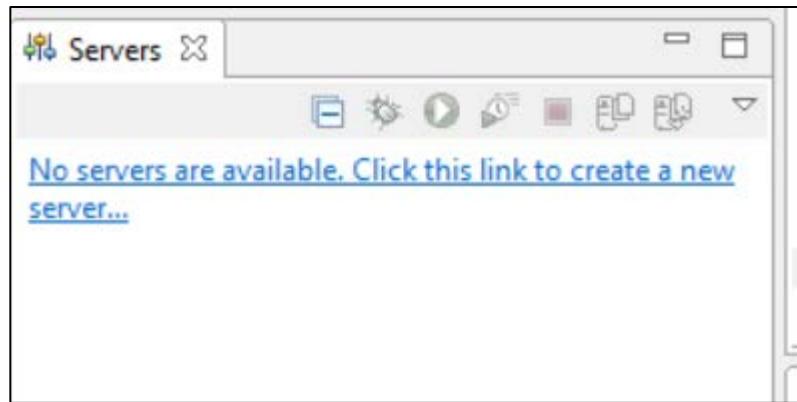
Sync from the JCR to Eclipse is “Importing from the Server” (manual only).

Steps:

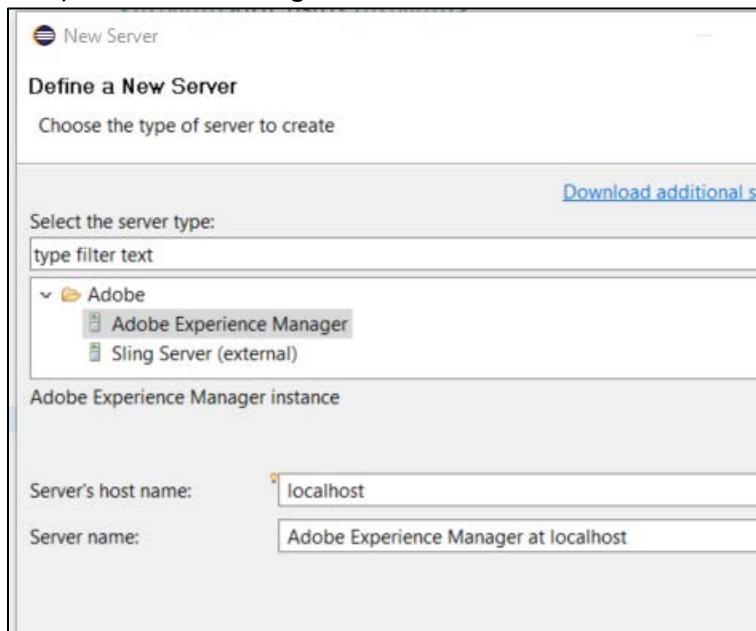
1. Create a new server in Eclipse.
 - a. In Eclipse, verify AEM Perspective is selected in the upper-right corner.



- b. On the Servers tab, click the **No Servers are available. Click this link to create a new server...** link.

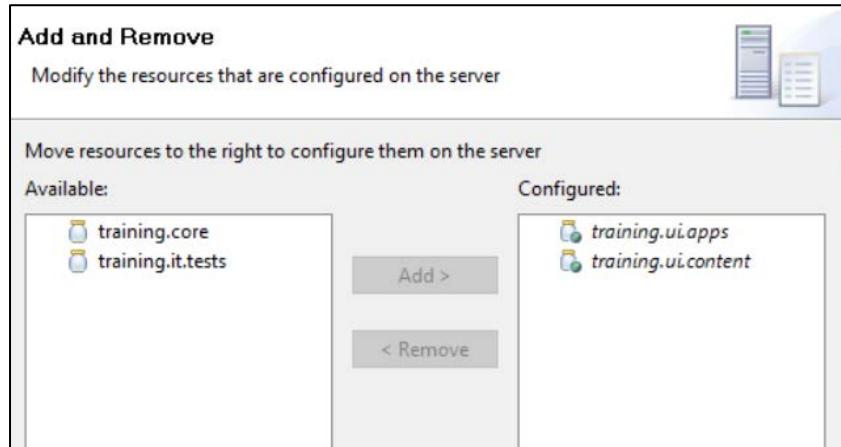


- c. In the Define a New Server dialog, select **Adobe > Adobe Experience Manager**.
- d. Accept the default settings.

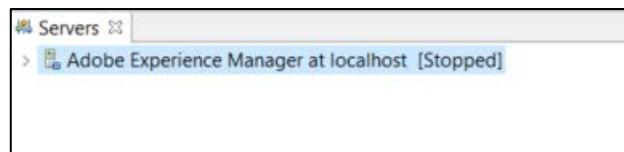


- e. Click **Next**.

- f. Select **training.ui.apps** and **training.ui.content** and click the **Add >** button to move the specified resources from the Available column to the Configured column as shown.

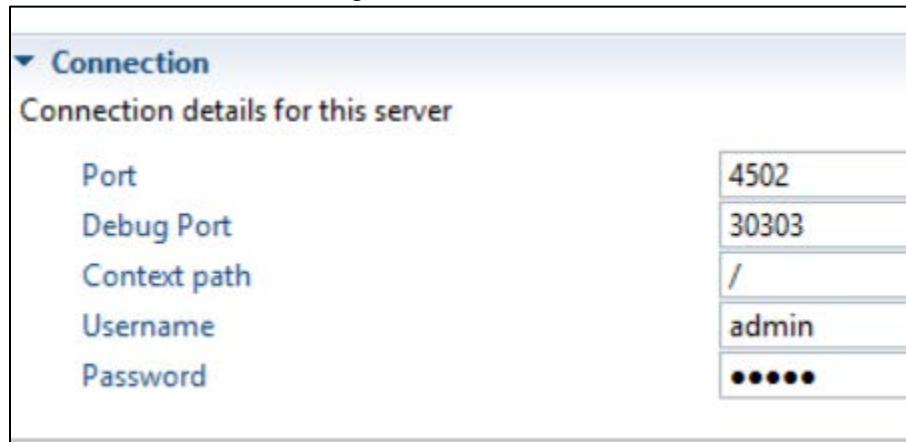


- g. Click **Finish**.
2. On the Servers tab, note how **AEM Server (Stopped)** is now available.



3. Modify the configuration for the AEM server.
- a. Double-click the **Adobe Experience Manager** to open it in the editor.

- b. In the **Connection** area, change **Port** to **4502**.



The screenshot shows the 'Connection' configuration dialog in Eclipse. It displays connection details for a server, specifically:

Setting	Value
Port	4502
Debug Port	30303
Context path	/
Username	admin
Password	*****



Note: The reason to change the AEM port is because your server is running on 4502.

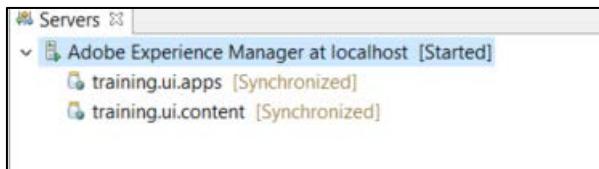
- c. Save the changes (**File > Save**).



Note: You have now created a connection in Eclipse to the AEM server running on your machine.

4. Right-click the server and click **Start**.

- a. Verify the Server has started.



The contents are now in Synch with AEM Server.

5. Export content from Eclipse to the JCR (automatically).

- a. In Project Explorer, navigate to: **training.ui.apps > src / main /content/jcr_root > apps> trainingproject > components > structure > page**
 - b. Double-click **page.html** to open.
 - c. Add the following line to the comment part:

"This is a change in Eclipse"

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

"This is a change in Eclipse"

- d. Save the changes.



Note: The moment you saved **page.html**, the AEM Server connection in Eclipse exported **page.html** to the JCR.

6. Back in CRXDE lite, browse to: **apps > trainingproject > components > structure > page > page.html** and double-click **page.html** to open it.
7. Verify the change in CRXDE Lite:

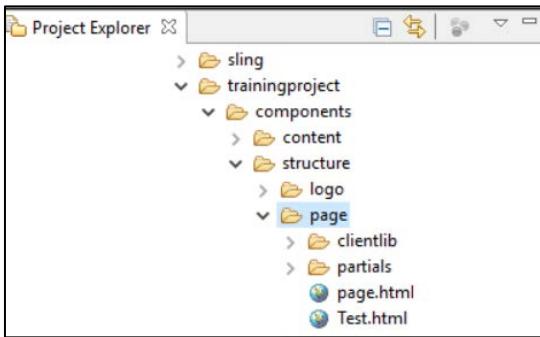
```
1 <!--/*
2      Copyright 2015 Adobe Systems Incorporated
3
4      Licensed under the Apache License, Version 2.0 (the "License");
5      you may not use this file except in compliance with the License.
6      You may obtain a copy of the License at
7
8          http://www.apache.org/licenses/LICENSE-2.0
9
10     Unless required by applicable law or agreed to in writing, software
11     distributed under the License is distributed on an "AS IS" BASIS,
12     WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13     See the License for the specific language governing permissions and
14     limitations under the License.
15
16     "This is a change in Eclipse"
17 *<!--&gt;
18 &lt;!DOCTYPE html&gt;</pre>
```

8. Import content from the JCR to Eclipse (manually).
 - a. In CRXDE lite, navigate to: **apps > trainingproject > components > structure > page**
 - b. Right-click the **page** node and select **Create > Create File**.
 - c. Create a page named **test.html**.
 - d. Click **OK** and save the changes (**Save All**).



Note: Make sure to click **Save All** after every change in CRXDE lite.

9. Back in Eclipse, under ui.apps, select `/apps/trainingproject/components/structure/page`, right-click and select **Import from Server**.
 - a. Accept the default and click **Finish**.
 - b. Verify Test.html appears in your project



Note: Step 4 is a very typical process in AEM Java Development to sync new content from the JCR to Eclipse. Remember that Eclipse is our master repo locally and anything created in the JCR that is a part of our project must be pulled back down into Eclipse. This is a very common process with config nodes, dialog structures, components, and clientlibs...things that are easier to create in CRXDE Lite.

Rule of Thumb: If you create something in CRXDE Lite that you want to keep, you must sync it back to Eclipse using the process above.

Review

In this lab, you performed the following:

- Installed Eclipse
- Installed AEM plugin for Eclipse
- Created a new project
- Built and deployed the project to AEM
- Configured AEM to perform content synchronization

5 CONFIGURING RUNMODES AND CONFIG NODES

Overview

This module explains the various types of run modes available with Adobe Experience Manager, and provides detailed descriptions on their configurations. This chapter also includes instructions on how to create configuration nodes.

Objectives

By the end of this module, you will be able to:

- Set different run modes
- Customize run modes
- Create config nodes in repository

Using Custom Run Modes

You can run your Adobe Experience Manager instance for specific purposes by using specific run modes. For example, author, publish, test, development, and so on. For run modes, you can:

- Define collections of configuration parameters for each run mode. A basic set of configuration parameters is applied to all run modes; you can then configure additional sets to meet your specific environment. These are applied as required.
- Define additional bundles to be installed for a specific mode.

Adobe Experience Manager has built-in author and publish run modes (do not remove these). If required, you can add additional custom run modes. For example, you can configure run modes for:

- **Environment:** local, development, test, and production
- **Location:** Berlin, Basel, Timbuktu
- **Company:** acme, partner, customer
- **Special system type:** importer

While you can change run modes after installation, several specific run modes, called installation run modes, cannot be altered once an Adobe Experience Manager instance is installed. These include author / publish and samplecontent / nosamplecontent. These two pairs of run modes are mutually exclusive (specifically, AEM can be defined as author or publish, but not both). Author can be combined with samplecontent and nosamplecontent, but not both at the same time.

Customized run modes can differentiate instances by purpose, stage of development, or location. Some examples of complex run modes (based on different locations and facilities) are:

- **author, development**
- **publish, test**

- **author, intranet, us**

All settings and definitions are stored in the repository, and activated by setting the appropriate run mode.

Setting Run Modes

It is possible to define specific run mode(s) that 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 must be set as VM options. For example, from the command line:

```
java -jar cq-quickstart.jar -r author,foo,dev
```

Alternatively, in the start script:

```
::* runmode(s)  
set CQ_RUMMODE=author,foo,dev
```

Or, by adding entries to the crx-quickstart/conf/sling.properties file. For example:

```
sling.run.modes=author,foo,dev
```

Configurations per Run Mode

To create separate configuration settings per run mode in the JCR, create folder names in the form: config.<runmode> they are used. For example: "config.publish":
`/apps/myapp/config.publish`

For systems with run-modes publish and berlin: `/apps/myapp/config.publish.berlin`

Configurations for Different Run Modes

Some examples of configuration settings that may be needed for different run modes:

Different mail server configurations per location:

```
config.basel/com.day.cq.mailer.DefaultMailService
```

```
config.berlin/com.day.cq.mailer.DefaultMailService
```

Enabling or disabling debugging per environment:

```
config.prod/com.day.cq.wcm.core.impl.WCMDebugFilter
```

```
config.dev/com.day.cq.wcm.core.impl.WCMDebugFilter
```

Additional Information on Run Modes

When using different configurations for separate run modes, the following apply:

- Partial configurations are not supported.
- The configuration with maximum matching run modes wins.

To avoid unexpected results:

- Always set all properties to avoid confusion.
- Use a type indicator (for example, {Boolean}, {String}) in every property.

Using Custom Run Modes with Configurations

In this section, you will examine how custom run modes affect bundle configuration. In this next exercise, you will explore the custom log file created from the archetype via config node.

Creating Configurations

You can use two methods to create configurations:

- Web Console
- Config node in the Repository

The following sections dive into the process of creating configuration nodes in the repository.

Creating Configurations in the Web Console

The Web Console is an out-of-the-box interface for OSGi configuration. Its UI enables you to edit properties by selecting from a predefined list. Any configurations made with this console are applied immediately, requires no restart, and applies to the current instance, regardless of the current run mode, or any subsequent changes to the run mode.

You can configure all bundles through the Configuration tab, which means you can use it to configure Adobe Experience Manager system parameters.

Creating Configurations in the Repository with nodes

You can define configurations for each run mode by creating specific nodes in the repository. These nodes are of type sling:OsgiConfig. Using this method, it is possible to share a configuration among several instances.

If you modify the configuration data in the repository, the changes are immediately applied to the relevant OSGi configuration as if the changes were made using the Web console, with the appropriate validation and consistency checks. This also applies to the action of copying a configuration from /libs to /apps.

You should consider these factors:

- Persistent Identity (PID) of the service
- Run modes—Check whether a specific run mode is required. If so, create the folder specific to that run mode. For example, config for all run modes, config.author for the author environment, and config.publish for the publish environment.
- Requirements of configurations or factory configurations
- Parameters—the individual parameters to be configured; including any existing parameter definitions that will need to be recreated.
- Existing configurations—customize existing configurations by copying the required configurations from the /libs folder to the /apps folder, and then modifying it in /apps. You can search for the required configurations using the Query tool available in CRXDE Lite.

Configuration Persistence

It is important to note a few details on how changes to the configurations are persisted.

- By default, you can find any change made to a configuration in /apps/system/config.

- Default settings in AEM are saved in * .config files under /crx-quickstart/launchpad/config.

Warning: You must never edit the folders or files under this folder.

Packaging Best Practices

After you create application packages, you need a process for deploying these to other environments. On a basic level, this can be moving the package zip file to a new environment, uploading it to the repository, and importing its contents. Alternatively, you can activate the package using the tools section of the Admin Console. However, you should give some consideration to managing this process to ensure your applications can be deployed efficiently and without any problems.

The following points are what you need to consider while creating packages:

- Verify your application code is separate from the configuration because 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.
- Ensure your application code does not depend on specific content because this may not exist in all environments.
- Code packages will be created in the development environment and will then travel from development to test to production. Content will generally travel from production to test to development to be used as test content so separate packages and processes may be needed.

Adding a New Configuration to the Repository

While creating config nodes, you must ensure their names are equal to the Persistent Identity (PID) of the configuration. For example, you can navigate to <http://localhost:4502/system/console/configMgr>, and see these names listed as service.pid properties. These configuration nodes must be child-nodes of node type sling:folder with a name starting with config followed with a dot. All the run-modes that the config applies to are also separated with a dot.

Examples: config.author, config.publish, config.author.dev, config.author.foo.dev, and so on.

To create a config node, you need to know the following:

- The Persistent Identity (PID) of the service. You can do this through the **Configurations** field in the Web console. The name is shown in brackets after the bundle name (or in the **Configuration Information** towards the bottom of the page).
- Whether a specific run mode is required. Create the folder:
 - Config—for all run modes
 - config.author—for the author environment
 - config.publish—for the publish environment
 - config.<run-mode>—as appropriate
- Whether a Configuration or Factory Configuration is necessary.
- The individual parameters to be configured; including any existing parameter definitions that will need to be recreated. Reference the individual parameter field in the Web console. The name is shown in brackets for each parameter.
- Does a configuration already exist in /libs? To list all configurations in your instance, use the Query tool in CRXDE Lite to submit the following SQL query:

```
select * from sling:OsgiConfig
```

If so, you can copy this configuration to `/apps/<yourProject>/`, then customize it in the new location.

You can create a config node in the repository using CRXDE Lite.

For each parameter you want to configure, create a property on this node.

Changes are applied as soon as the node is updated by restarting the service (as with changes made in the Web console).

Resolution of Config Nodes

The following sections show how to resolve the config nodes.

Resolution Order at Startup

The following order of precedence is used:

Repository nodes under `/apps/*/config`, either with type `sling:OsgiConfig` or property files.

Repository nodes with type `sling:OsgiConfig` under `/libs/*/config`.

Any config files from `<cq-installation-dir>/crx-quickstart/launchpad/config/` on the local file system.

This means project-specific configurations under `/apps` take precedence over `/libs`.

Resolution Order at Runtime

Configuration changes made while the system is running triggers a reload with the modified configuration. The following order of precedence applies:

1. Modifications made in the Web Console.
2. Modifications made in /apps.
3. Modifications made in /libs.

Resolution of Multiple Run Modes

For run mode-specific configurations, you can combine multiple run modes. For example, you can create configuration folders in the following style:

/apps/*/config.<runmode1>.<runmode2>/

Configurations in these folders will be applied if all run modes match a run mode defined at startup. For example, if an instance was started with the run modes

publish, dev, emea, configuration nodes in /apps/*/config.emea,
/apps/*/config.publish.dev/ and
/apps/*/config.publish.emea.dev/ will be applied, while configuration
nodes in /apps/*/config.publish.asean/ and
/config/publish.dev.emea.noldap/ will not be applied.

If multiple configurations for the same PID are applicable, the configuration with the highest number of matching run modes is applied. For example, if an instance was started with the run modes publish, dev, emea, and both /apps/*/config.publish/ and /apps/*/config.emea.publish/ define a configuration for com.day.cq.wcm.core.impl.VersionManagerImpl, the configuration in /apps/*/config.emea.publish/ will be applied.

LAB D – WORK WITH CUSTOM RUN MODE

Overview

In this lab, you will start a server with a custom run mode as shown:

Using Java Virtual Machine(JVM) arguments:

Example: `java -Xmx512m -jar aem-author-4502.jar -r=author,dev`

You will also create a configuration and examine the node details.

Objectives

- Start Adobe Experience Manager in Custom Run Mode
- Create a configuration node

Prerequisites

- AEM installed on your machine:
 - Running Adobe Experience Manager Author instance
 - An Eclipse project from the AEM Archetype

Directions

Complete the exercises that follow.

Exercise 1 – Start Adobe Experience Manager in Custom Run Mode

Overview

In this lab exercise, you will start the Adobe Experience Manager in Custom run mode.

Steps

1. Start Adobe Experience Manager in [author,dev] run mode.

- a. Stop the running author instance
-



Note: This method allows the user to activate the custom run mode while starting Adobe Experience Manager from the command line using the -r option.

- b. Issue the following command to launch an AEM instance with run mode set to author,dev and the GUI window “on”:
➤ `java -jar aem-author-4502.jar -r author,dev -gui`
- c. Verify the Adobe Experience Manager Author instance has started successfully.
- d. The start-up GUI window/dialog will change to something similar to the following:



2. Log in to AEM, click **Adobe Experience Manager** in the upper left, and navigate to **Tools > Operations > Web Console**.
 - a. Choose **Status > Sling Settings** from the menu bar.
 - b. Notice the value for the Run Modes has changed from the default values:

Adobe Experience Manager Web Console Sling Settings

Main OSGi Sling Status Web Console

Date: April 27, 2017 11:33:55 AM PDT

Apache Sling Settings

```
Sling ID = 31f75947-1061-4cc2-904b-86eeeb9a7cd0
Sling Name = Instance 31f75947-1061-4cc2-904b-86eeeb9a7cd0
Sling Description = Instance with id 31f75947-1061-4cc2-904b-86eeeb9a7cd0 and run modes [dev,
Sling Home = C:\adobe\AEM\author\crx-quickstart
Sling Home URL = file:/C:/adobe/AEM/author/crx-quickstart/
Run Modes = [dev, crx3, author, samplecontent, crx3star]
```

Review:

You should now be able to:

- Run the Adobe Experience Manager Author instance in custom run mode

Exercise 2 – Create a Configuration Node

Overview

In this lab exercise, you will create a configuration node for a custom run mode and verify the output.

You can start a server with a custom run mode as explained in the previous task. Now you have to create a custom configuration for one of the run modes you created. When you start an author instance, it always loads the projects.html page by default. In this exercise, you will change the default page to sites.html for a server in runmode [author] and crx/de/index.jsp for a server in runmode [author,dev].

STEPS

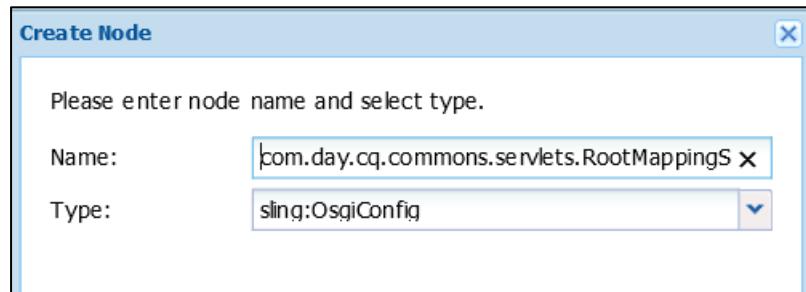
1. Create a configuration node.

Note: All OSGI configurations are under **Web Console > OSGI > Configuration**.

- a. Navigate to Tools > Operations > Web Console and select OSGI > Configuration.
- b. Find the configuration named **Day CQ Root Mapping**.
 - Expand **Day CQ Root Mapping** by clicking on it, and copy the value for **Persistent.Identity**:

Day CQ Root Mapping	
rootmapping.desc	
Target	/projects.html
Path	rootmapping.target.desc (rootmapping.target)
Configuration Information	
Persistent Identity (PID)	com.day.cq.commons.servlets.RootMappingServlet
Configuration Binding	jcrinstall:/libs/cq/platform/install/cq-commons-5.9.22.jar
	Day Communique 5 Commons Library (com.day.cq.cq-commons), Version 5.9.22

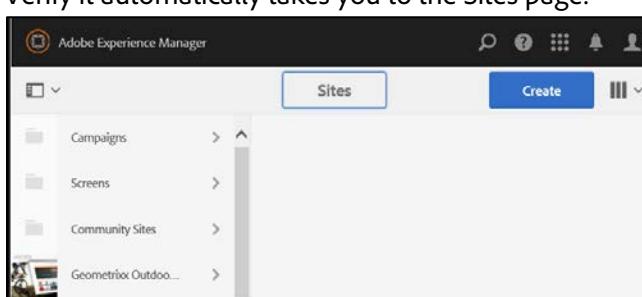
- You will create a configuration node for this configuration in CRDXE Lite. Keep this page open for reference.
2. Log in to AEM and navigate to **Tools > General > CRXDE Lite**.
- a. In CRDXE Lite, navigate to **apps > trainingproject** and select **config.author**
 - b. Right-click **config.author** and select **Create > Create Node**. Specify the following:
 - Name: **com.day.cq.commons.servlets.RootMappingServlet**
 - Type: **sling:OsgiConfig**



- c. Click **OK**
- d. Click **Save All** to save the configuration.
- e. With the **com.day.cq.commons.servlets.RootMappingServlet** node selected, navigate to the JCR Properties tab and create a property
 - Name = **rootmapping.target**
 - Type = **String**
 - Value= **/sites.html**

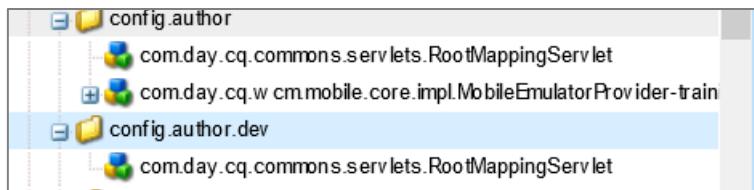
Properties		Access Control	Replication	Console	Build Info
	Name ▲	Type	Value		
1	jcr:created	Date	2017-02-21T09:55:20.493-08:00		
2	jcr:createdBy	String	admin		
3	jcr:primaryType	Name	sling:OsgiConfig		
4	rootmapping.target	String	/sites.html		

- f. Click **Add** and **Save All** to save the configuration.
3. Open a browser and enter this URL: <http://localhost:4502>
- a. Verify it automatically takes you to the Sites page:



Analysis: Because you put the configuration under config.author, AEM picks up the configuration because the server is in [author,dev] runmodes and this configuration has the most matching runmodes, [author].

4. Create another configuration node.
 - a. In CRDXE Lite, navigate **apps > trainingproject**
 - b. Right-click **trainingproject** and create a new folder named **config.author.dev**.
 - c. Save the changes.
 - d. Copy the node **RootMappingServlet** from the **config.author** folder:
 1. Select **config.author > com.day.cq.commons.servlets.RootMappingServlet**
 2. Right-click **com.day.cq.commons.servlets.RootMappingServlet** and choose **copy** from the menu bar.
 - e. Right-click **config.author.dev** and select **paste**.

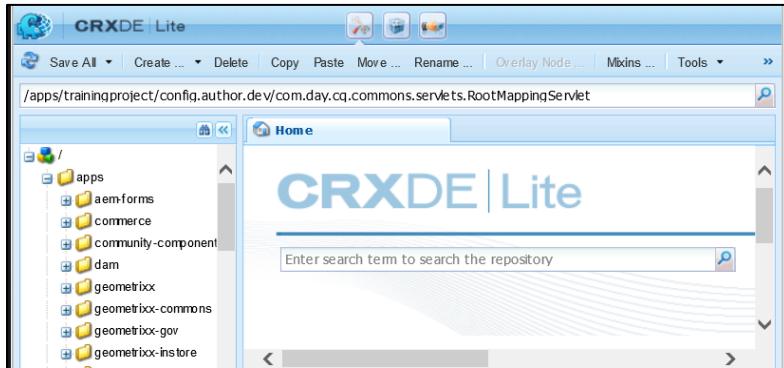


- f. Select **config.author.dev > com.day.cq.commons.servlets.RootMappingServlet** to view the properties.
- g. Change the **Value** for the property **rootmapping.target** from **/sites.html** to **/crx/de/index.jsp**, with the final result shown:

jcr:created	Date	2017-01-18T16:51:42.846-08:00
jcr:createdBy	String	admin
jcr:primaryType	Name	sling:OsgiConfig
rootmapping.target	String	/crx/de/index.jsp

- h. Save the changes.

5. Verify the change was made to the dev run mode.
 - a. Open a browser and enter this URL: <http://localhost:4502>
 - b. Notice now it automatically takes you to the CRXDE page.



Analysis: Because you put the configuration under config.author.dev, AEM picks up the configuration because the server is in [author,dev] runmodes and this configuration has the most matching runmodes, [author,dev].

6. Sync the configuration to your workspace.
 - a. In Eclipse, right-click **training.ui.apps**, select **AEM > Import from server...**
 - b. Verify the changes.



NOTE: You may need to ensure your server in Eclipse has started. If it is not, start it.

Review:

In this exercise, you did the following:

- Created a Configuration node

6 CONFIGURING CUSTOM LOG FILES

Overview

Adobe Experience Manager log files provide detailed information about the current system state. In addition to the default system log files, you can also create and customize your own log files. They can help you better track messages produced by your own applications and separate them from the default log entries.

Adobe Experience Manager offers you the possibility to configure:

- Global parameters for the central logging service.
- Request data logging; a specialized logging configuration for request information.
- Specific settings for the individual services. For example, an individual log file and format for the log messages.

This module provides instructions on configuring your own custom log files.

Objectives

By the end of this module, you will be able to:

- Create custom log files

Understanding the Logging System

Logging in to Adobe Experience Manager is based on Sling, and is supported by the org.apache.sling.commons.log bundle. This bundle has the following features: Implements the OSGi Log Service Specification and registers the LogService and LogReader services

- Exports four commonly used logging APIs:
 - Apache Commons Logging
 - Simple Logging Facade for Java (SLF4J)
 - log4j
 - java.util.logging
- Configures logging through Logback, which is integrated with the OSGi environment
- Allows logging to be configured both via editing Logback xml or via OSGi configurations

Types of Log Files in Adobe Experience Manager

The following list provides a brief description of the types of log files you will find within Adobe Experience Manager. These files are available in the installation directory: /crx-quickstart/logs

- access.log—registers all access requests sent to Adobe Experience Manager and the repository.
- audit.log—registers all moderation actions.
- error.log—registers all error messages.
- request.log—registers all access requests along with their responses.
- stderr.log—holds error messages generated during startup.
- stdout.log—holds logging messages indicating events during startup.

- upgrade.log—provides a log of all upgrade operations.

By default, the error, access, history and request logs rotate once per day. When this occurs, the existing log files are appended with a timestamp, and a new file is created.

In Adobe Experience Manager, you can view the log files through the Web Console at: <http://localhost:4502/system/console/slinglog>

The screenshot shows the 'Log Support' section of the AEM Web Console. It displays two tables: one for 'Logger' configurations and one for 'Appender' configurations.

Logger (Configured via OSGI Config)

Log Level	Log File	Logger	Configuration
INFO	logs\upgrade.log	com.adobe.cq.upgradexecutor com.day.cq.compat.codeupgrade com.adobe.cq.upgrades	
INFO	logs\audit.log	org.apache.jackrabbit.oak.audit org.apache.jackrabbit.core.audit	
DEBUG	logs\auditlog.log	com.adobe.granite.audit	
INFO	logs\history.log	log.history	
INFO	logs\error.log	ROOT	
INFO	logs\request.log	log.request	
WARN	logs\error.log	org.apache.pdfbox	
INFO	logs\access.log	log.access	

Add new Logger

Appender

Appender	Configuration
File : [/logs/request.log] C:\Users\mogra\Desktop\AEM Loads\Blank Site\crx-quickstart\logs\request.log	
File : [/logs/access.log] C:\Users\mogra\Desktop\AEM Loads\Blank Site\crx-quickstart\logs\access.log	
File : [/logs\auditlog.log] C:\Users\mogra\Desktop\AEM Loads\Blank Site\crx-quickstart\logs\auditlog.log	
File : [/logs\upgrade.log] C:\Users\mogra\Desktop\AEM Loads\Blank Site\crx-quickstart\logs\upgrade.log	
File : [/logs\error.log] C:\Users\mogra\Desktop\AEM Loads\Blank Site\crx-quickstart\logs\error.log	
File : [/logs\history.log] C:\Users\mogra\Desktop\AEM Loads\Blank Site\crx-quickstart\logs\history.log	
File : [/logs\audit.log] C:\Users\mogra\Desktop\AEM Loads\Blank Site\crx-quickstart\logs\audit.log	

Loggers and Writers

Adobe Experience Manager's logging system consists of two elements—a Logging Logger and a Logging Writer. The Writer persists the data provided by the Logger to a configurable location. Usually, it is a file. For example, the error.log file is created by the Writer on a rotational basis. The Logger collects data from different components inside Adobe Experience Manager, filters them by requested severity level, and redirects the output to a configured Writer.

Adobe Experience Manager lets you configure two types of settings:

Global logging. This defines:

- logging level
 - central log file location
 - number of versions saved
 - version rotation; either maximum size or a time interval
 - format used when writing the log messages
- Loggers and Writers for an individual service. This defines:
 - specific logging level
 - individual log file location
 - number of versions to be kept
 - version rotation; either maximum size or the time interval
 - format used when writing the log messages
 - logger (the OSGi service supplying the log messages)

Individual Service Loggers and Writers

You can channel log messages for a single service into a separate file. Adobe Experience Manager uses the following process to write log messages to a file:

1. OSGi service (logger) writes a log message.
2. Logging Logger takes this message and formats it according to your specification.
3. Logging Writer writes all these messages to the physical file you defined.

These elements are linked by the following parameters:

- Logger (Logging Logger): Defines the service(s) generating the messages.
- Log File (Logging Logger): Defines the physical file for storing the log messages. This parameter links a Logging Logger with a Logging Writer.
- Log File (Logging Writer): Defines the physical file the log messages will be written to.

Standard Loggers and Writers

Once Adobe Experience Manager is installed, you have access to the following Writers and Loggers:

Logger	Links To
Apache Sling Customizable Request Data Logger <code>(org.apache.sling.engine.impl.log.RequestLoggerService)</code> Writes messages about requests to the request.log file.	Apache Sling Request Logger <code>(org.apache.sling.engine.impl.log.RequestLogger)</code> Writes messages to either request.log or access.log .
Apache Sling Logging Logger Configuration <code>(org.apache.sling.commons.log.LogManager.factory.config)</code> Writes information messages to logs/error.log .	Apache Sling Logging Writer Configuration (Writer) <code>(org.apache.sling.commons.log.LogManager.factory.writer)</code>
Apache Sling Logging Logger Configuration <code>(org.apache.sling.commons.log.LogManager.factory.config.649d51b7-6425-45c9-81e6-2697a03d6be7)</code> Writes Warning messages to <code>..../logs/error.log</code> for the service org.apache.pdfbox .	Does not link to any specific Writer. It will create and use an implicit Writer with default configuration.

Creating Your Own Loggers and Writers

You can define your own Logger / Writer pair as follows:

1. Create a new instance of the Factory Configuration Apache Sling Logging Logger Configuration.
 - a. Specify the Log File.
 - b. Specify the Logger.
 - c. Configure the other parameters as required.
2. Create a new instance of the Factory Configuration Apache Sling Logging Writer Configuration.
 - a. Specify the Log File.
 - b. Configure the other parameters as required.



Note: If you do not create a Logging Writer configuration for your custom logger, the default writer configuration will be applied where log files are archived once a day and archived log files will have an extension of the date. There are a few different ways you can create custom log files.

3. An OSGi configuration node in the JCR. This is recommended as best practice so the configuration can be put under version management and easily transferred via content package from environment to environment
4. Web Console – Configuration. Although this is possible, Adobe recommends using this console as a view only console for configurations and consider creating a configuration node in the JCR.
5. Web Console – Sling Log Support. This is another console you can view the current log files on a server. This console can be great to create a quick log file with a specific Logger. The ability to ad hoc log files is helpful and beneficial to development, debugging, and troubleshooting. You should create permanent log files with a configuration node.

LAB E - CONFIGURE CUSTOM LOGGERS

Overview

In this lab, you will examine the configuration of the custom logger for your project. You will also create a log file that will log users logging into the AEM server.

Objectives

- Observe Project Logger Config Node
- Create Temporary Loggers

Prerequisites

- AEM installed on your machine:
- Running Adobe Experience Manager Author instance
- An Eclipse project from the AEM Archetype

Directions

Complete the exercises that follow.

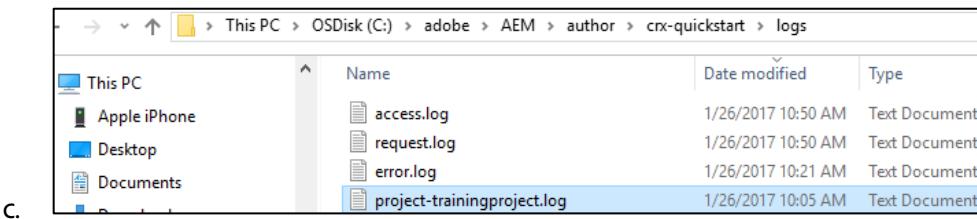
Exercise 1 – Observe Project Logger Config Node

Overview

In this exercise, you will observe the custom log file and the logger configuration node that created it. This configuration node was created by the AEM Archetype.

Steps

1. Observe the custom log.
 - a. Open the Adobe Experience Manager instance folder (**C:\adobe\AEM\author**).
 - b. Navigate to: **\crx-quickstart\logs\project-trainingproject.log**



Note: This is the custom log file that was created from the AEM Archetype for this project. By default, all log messages from this course are located in this log file.

2. Observe the configuration node in Eclipse that created the custom log.

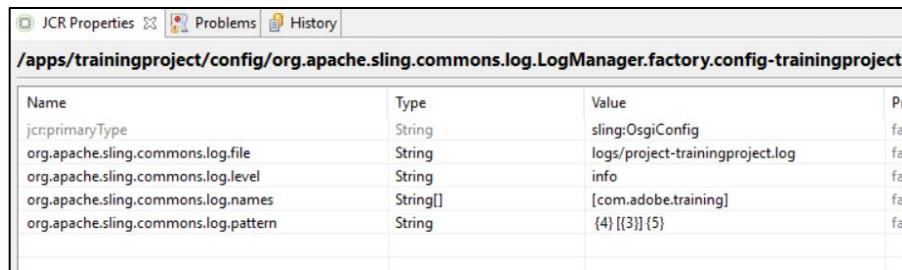
a. In Project Explorer, navigate to:

training.ui.apps>src/main/content/jcr_root>apps>trainingproject/config

b. Double-click org.apache.sling.commons.log.LogManager.factory.config-trainingproject.xml to open it in the Eclipse text editor.

c. Click the **JCR Properties** tab.

d. Review the properties for the log:



The screenshot shows the Eclipse IDE's JCR Properties tab for a specific configuration node. The node path is /apps/trainingproject/config/org.apache.sling.commons.log.LogManager.factory.config-trainingproject. The table displays the following properties:

Name	Type	Value	Pr
jcr:primaryType	String	sling:OsgiConfig	fa
org.apache.sling.commons.log.file	String	logs/project-trainingproject.log	fa
org.apache.sling.commons.log.level	String	info	fa
org.apache.sling.commons.log.names	String[]	[com.adobe.training]	fa
org.apache.sling.commons.log.pattern	String	{4} {[3]} {5}	fa

3. Verify the config was synched to AEM when the project was originally deployed.

e. Open Adobe Experience and click Adobe Experience Manager in the upper left.

4. Navigate to: Tools > General > CRXDE lite
- Browse to: apps > trainingproject > config >
org.apache.sling.commons.log.LogManager.factory.config-trainingproject



- Verify the values for the log properties:

Properties		Access Control	Replication	Console	Build Info
Name	Type				
1 jcr:created	Date	2017-01-13T09:47:06.934-08:00			
2 jcr:createdBy	String	admin			
3 jcr:primaryType	Name	sling:OsgiConfig			
4 org.apache.sling.commo...	String	logs/project-trainingproject.log			
5 org.apache.sling.commo...	String	info			
6 org.apache.sling.commo...	String[]	com.adobe.training			
7 org.apache.sling.commo...	String	{0,date,yyyy-MM-dd HH:mm:ss.SSS} {4} [{3}] {5}			

5. Observe the configuration node is registered in the Web Console.
 - a. Go to: <http://localhost:4502/system/console/configMgr>
 - b. Use your browser to search for and find **Sling Logging Logger Configuration**.
 - c. Under this configuration, find the configuration called: **logs/project-trainingproject.log: info**
 - d. Click to expand it, and verify the properties are the same as the configuration node.

The screenshot shows the 'Apache Sling Logging Logger Configuration' page. At the top, a note says: 'Configure Loggers with levels, pattern and destination. See <http://sling.apache.org/site/logging.html> for more detailed documentation and description.' Below this, there are several configuration sections:

- Log Level:** Information (Root Logger log level setting: org.apache.sling.commons.log.level)
- Log File:** logs/project-trainingproject.log (The name and path of the log file. If this is empty, logging goes to standard output (the console). If this path is relative it is resolved below \${sling.home} - org.apache.sling.commons.log.file)
- Message Pattern:** {0,date,yyyy-MM-dd HH:mm:ss.SSS} {4} {3} {5} (Message Pattern for formatting the log messages. For complete details refer to <http://logback.qos.ch/manual/layouts.html#ClassicPatternLayout> org.apache.sling.commons.log.pattern)
- Logger:** com.adobe.training (The logger names applicable for this logger configuration. Each logger name applies for any child category unless configured otherwise. E.g. a logger name of org.apache.sling applies to logger org.apache.sling if no specific logger is defined for org.apache.sling. org.apache.sling.commons.log.names)
- Additivity:** (If set to false then logs from these loggers would not be sent to any appender attached higher in the hierarchy (org.apache.sling.commons.log.additivity))

Below these sections is a 'Configuration Information' section:

Persistent Identity (PID)	org.apache.sling.commons.log.LogManager.factory.config.7c70446c-a918-44a7-ad57-e4e84ca3c91d
Factory Persistent Identifier (Factory PID)	org.apache.sling.commons.log.LogManager.factory.config
Configuration Binding	slinginstall(C:\adobe\AEP\author\xrx-quickstart\launchpad\startup\!\org.apache.sling.commons.log-4.0.6.jar)
	Apache Sling SLF4J Implementation (Logback) (org.apache.sling.commons.log), Version 4.0.6

Exercise 2 – Create Temporary Loggers

Overview

In this exercise, you will create a logger and verify the corresponding log file is created.

Steps

1. Create a logging logger.

- Open the Adobe Experience Manager instance folder (**C:\adobe\AEM\author**).
- Navigate to **crx-quickstart\logs** to see the default logs:

 access.log	12/4/2015 5:03 PM	Text Document	3 KB
 audit.log	11/23/2015 10:58 ...	Text Document	0 KB
 auditlog.log	11/23/2015 10:58 ...	Text Document	0 KB
 error.log	12/4/2015 5:04 PM	Text Document	0 KB
 history.log	12/4/2015 3:38 PM	Text Document	3 KB
 request.log	12/4/2015 5:03 PM	Text Document	3 KB
 stderr.log	12/4/2015 5:03 PM	Text Document	8 KB
 stdout.log	12/4/2015 5:03 PM	Text Document	7 KB
 upgrade.log	12/4/2015 3:39 PM	Text Document	1 KB



Note: After the new logger is created, you can see the log file in this folder. You can configure the log file location anywhere you want on your file system; however, we will use the current directory for this task.

2. Open the Log Support console at: <http://localhost:4502/system/console/slinglog>

a. Click Add new Logger.

The screenshot shows the 'Log Support' section of the AEM web console. At the bottom right of the table, there is a red box around the 'Add new Logger' button. A new row has been added to the table, corresponding to the step described in the text.

Logger (Configured via OSGi Config)				
Log Level	Log File	Logger	Configuration	
INFO	logs\upgrade.log	com.adobe.cq.upgradesexecutor com.day.cq.compat.codeupgrade com.adobe.cq.upgrades		
INFO	logs\audit.log	org.apache.jackrabbit.oak.audit org.apache.jackrabbit.core.audit		
DEBUG	logs\auditlog.log	com.adobe.granite.audit		
INFO	logs\history.log	log.history		
INFO	logs\error.log	ROOT		
INFO	logs\request.log	log.request		
WARN	logs\error.log	org.apache.pdfbox		
INFO	logs\access.log	log.access		
Add new Logger				

A new row is highlighted:

The screenshot shows the same 'Logger' table from the previous screenshot, but now the new row for 'INFO logs\error.log' is highlighted with a yellow background. This highlights the change made in the previous step.

Logger (Configured via OSGi Config)				
Log Level	Log File	Logger	Configuration	
WARN	logs\error.log	org.apache.pdfbox		
INFO	logs\upgrade.log	com.adobe.cq.upgradesexecutor com.day.cq.compat.codeupgrade com.adobe.cq.upgrades		
INFO	logs\access.log	log.access		
INFO	logs\history.log	log.history		
INFO	logs\audit.log	org.apache.jackrabbit.oak.audit org.apache.jackrabbit.core.audit		
DEBUG	logs\auditlog.log	com.adobe.granite.audit		
INFO	logs\error.log	ROOT		
INFO	logs\request.log	log.request		
INFO	logs\error.log			
<input style="background-color: yellow; color: black; border: none; padding: 2px 10px; margin-right: 10px;" type="button" value="INFO"/> logs\error.log <input style="border: none; padding: 2px 10px; margin-right: 10px;" type="button" value="+"/> <input style="border: none; padding: 2px 10px;" type="button" value="Save"/> <input style="border: none; padding: 2px 10px;" type="button" value="Cancel"/>				

- b. In the **Log Level** drop-down menu (where INFO is the default), select **Debug**.
- c. Enter the Log File (name) as: **logs\LoginTrace.log**
- d. Enter the Logger as: **org.apache.sling.auth.core.impl.SlingAuthenticator**
- e. Click **Save**. You will see the logger is created successfully.

Logger (Configured via OSGi Config)				
Log Level	Log File	Logger	Configuration	
INFO	logs\upgrade.log	com.adobe.cq.upgradesexecutor com.day.cq.compat.codeupgrade com.adobe.cq.upgrades		
INFO	logs\audit.log	org.apache.jackrabbit.oak.audit org.apache.jackrabbit.core.audit		
DEBUG	logs\audit.log.log	com.adobe.granite.audit		
INFO	logs\history.log	log.history		
DEBUG	logs\LoginTrace.log	org.apache.sling.auth.core.impl.SlingAuthenticator		
INFO	logs\error.log	ROOT		
INFO	logs\request.log	log.request		

3. Navigate to **\crx-quickstart\logs** directory again (**\crx-quickstart\logs**) and see if the **LoginTrace.log** was created successfully.

access.log	12/4/2015 6:44 PM	Text Document	5 KB
audit.log	11/23/2015 10:58 ...	Text Document	0 KB
auditlog.log	11/23/2015 10:58 ...	Text Document	0 KB
error.log	12/5/2015 6:06 PM	Text Document	17 KB
history.log	12/4/2015 3:38 PM	Text Document	3 KB
LoginTrace.log	12/4/2015 6:44 PM	Text Document	5 KB
project-trainingproject.log	12/4/2015 5:07 PM	Text Document	2 KB
request.log	12/4/2015 6:44 PM	Text Document	4 KB
s7access-2015-12-04.log	12/4/2015 5:06 PM	Text Document	0 KB
stderr.log	12/4/2015 5:05 PM	Text Document	5 KB
stdout.log	12/4/2015 5:05 PM	Text Document	857 KB
upgrade.log	12/4/2015 5:07 PM	Text Document	1 KB

- a. Log out from Adobe Experience Manager, and then log in again.
- b. Observe how the log file contains entries as you configured them in the logger settings.

Note: This will ensure the events are captured in the log file.

Review:

- In this lab, you did the following:
 - Observed the project logger config node
 - Created a temporary logger

7 DEEP DIVE IN TO OSGI

Overview

This module deep dives into the Open Service Gateway initiative (OSGi) architecture. You will learn about the benefits of using OSGi, components, and annotations available in OSGi, as well as the configurable services.

Objectives

By the end of this module, you will be able to:

- Implement a Bundle Activator
- Create an OSGi service and use the service
- Create a OSGi configuration in a Java class
- Create an OSGi Event Handler

OSGi - An Overview

The OSGi Service Platform is a Java-based application server for networked devices. This non-proprietary service platform spans:

- Mobile phones
- Vehicles
- Telematics
- Embedded appliances
- Residential gateways
- Industrial computers
- Desktop PCs
- High-end servers (including mainframes)

Open Services Gateway Initiative (OSGi) is one of the fundamental layers in the Adobe Experience Manager technology stack. It is used to control the composite bundles of Adobe Experience Manager and their configuration. OSGi allows applications to be created from smaller, reusable, and collaborative components, which can then be deployed.

Bundles can be managed as they can be installed, started, or stopped individually, and interdependencies between these modules are automatically handled. Each of the OSGi components is contained in one of the bundles.

OSGi is a set of specifications that create a development model where applications are composed of different reusable components. This also means that the components can hide their implementations from other components, and communicate through services. These services are objects that are shared between components.

To summarize, an OSGi application is a collection of bundles that interact using service interfaces:

- Bundles may be independently developed and deployed.
- Bundles and their associated services may appear or disappear at any time.

There are a number of advantages to using OSGi Service Platform. Some of them are:

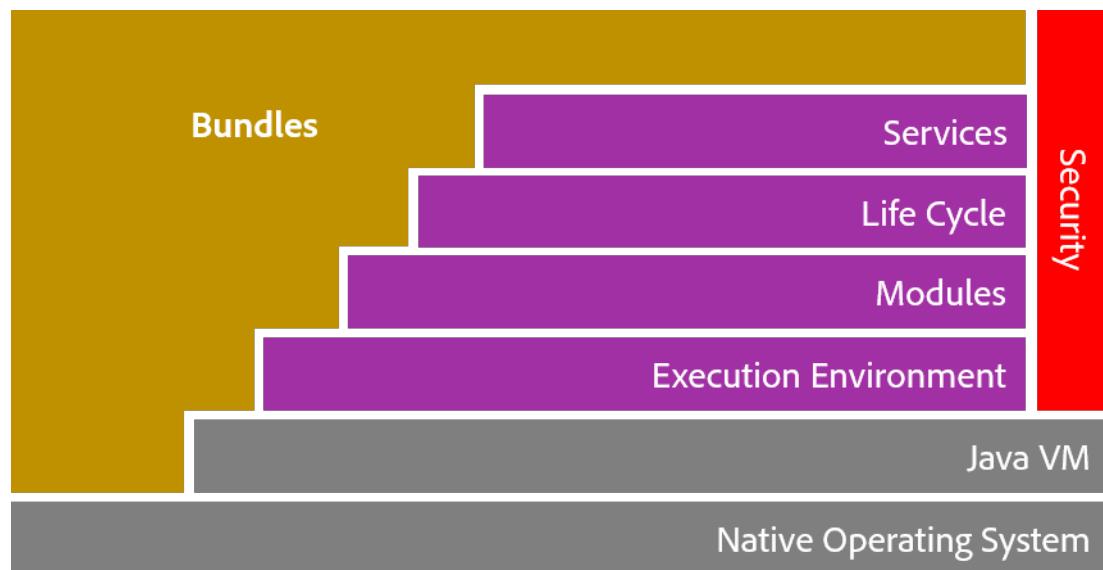
- Adding an OSGi Service Platform to a networked device enables you to manage the lifecycle of the software components in the device from anywhere in the network.
- OSGi simplifies the process of development and maintenance of a large number of configurations.
- Multiple java-based components can work efficiently within a single Java Virtual Machine (JVM). This technology provides an extensive security model so components can run in a shielded environment.

Other advantages:

- Remote component management
- Secure execution environment
- Cooperation between applications
- Commercial off-the-shelf components
- Simplified Deployment
- Multi-vendor interoperability
- Dynamic nature

OSGi Architecture

The OSGi has a layered model that is depicted in the following figure.



- Bundles: Bundles are the OSGi components made by the developers.
- Services: The services layer connects bundles in a dynamic way by offering a publish-find-bind model for plain old Java objects.
- Life-Cycle: The API to install, start, stop, update, and uninstall bundles.
- Modules: The layer that defines how a bundle can import and export code.
- Security: The layer that handles the security aspects.
- Execution Environment: Defines what methods and classes are available in a specific platform.

The following sections explain a few of the above concepts.

Bundles

Bundles are built on Java's existing standard way of packaging together classes and resources—the JAR file (.jar). An OSGi bundle is just a JAR file, with additional metadata added to the manifest file. The OSGi metadata is provided as header information in the META-INF/MANIFEST.MF file. The additional information consists of:

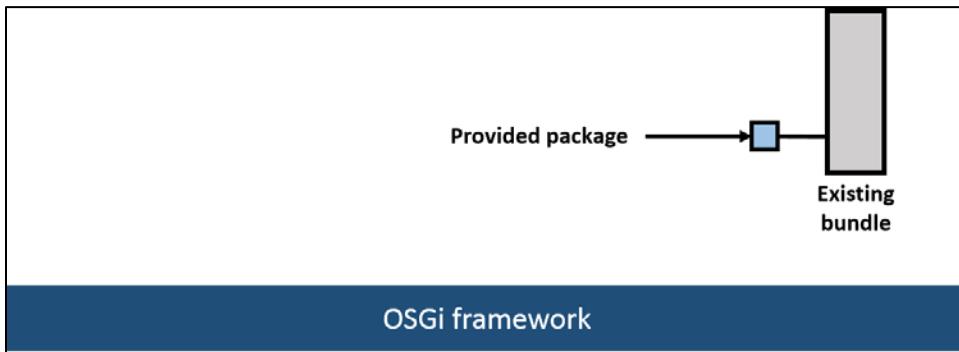
- Bundle name(s):
 - A "symbolic" name used by OSGi to determine the bundle's unique identity
 - An optional, human-readable, descriptive name
- Bundle version
- The list of services imported and exported by this bundle
- Optional additional information, such as:
 - Minimum Java version the bundle requires
 - Vendor of the bundle
 - Copyright statement
 - Contact address, and so on

The bundles are loosely coupled.

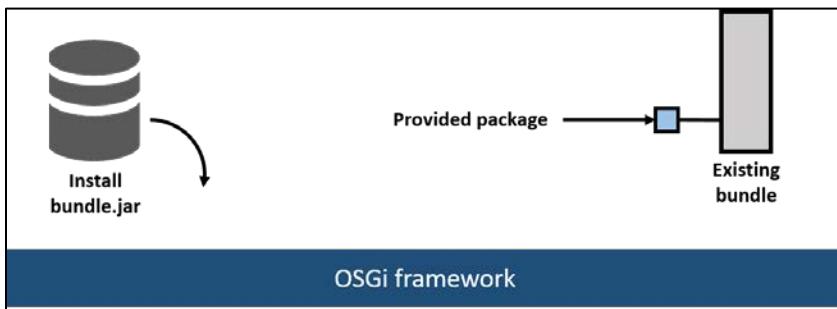
- It includes package imports and exports with versions.
- Dependencies are independent from the bundle organization.
- "Someone" provides the self-describing package.
- OSGi provides error management for unresolved bundles.
- Modular thinking is required during application design.
- It requires proper metadata and consistent version management.

Dependency Management Resolution

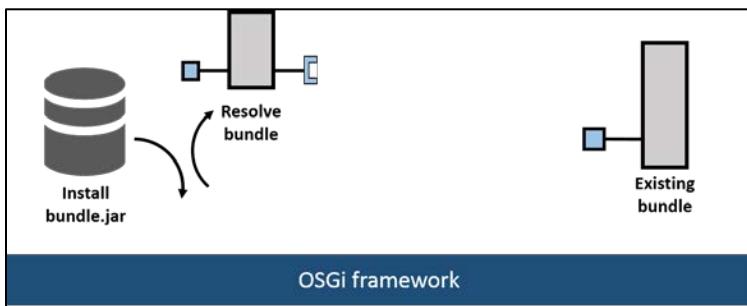
A bundle is present in the container.



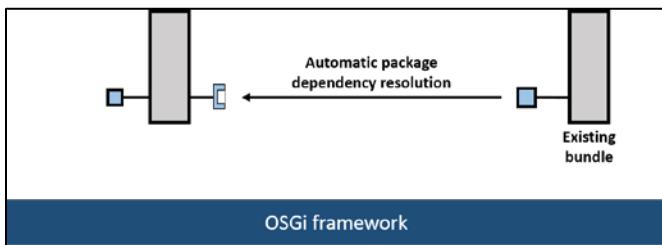
When a new bundle is provided, it is installed into the OSGi container.



The OSGi container resolves the new bundle.



The container provides automatic dependency resolution.



Modules

Modularity is at the core of the OSGi specifications and is embodied in the bundle concept. You should be familiar with the term “bundle” in Java, which means a JAR file. The difference between a Java bundle and an OSGi bundle is that in Java, the contents of the JAR are completely visible to all other JARs. In OSGi, it “hides” the JAR contents unless they are explicitly exported. If a bundle wants to use another JAR, it must explicitly import the parts it needs. Therefore, there is no sharing.

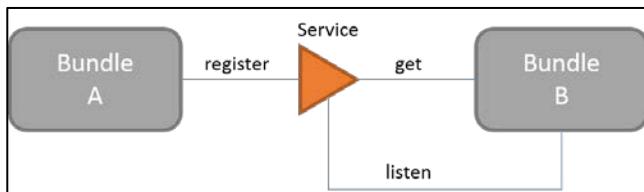
Though the code hiding and explicit sharing provides many benefits (for example, allowing multiple versions of the same library being used in a single VM), the code sharing is only there to support OSGi services model. The services model is about bundles that collaborate.

Services

A bundle can create an object and register it with the OSGi service registry under one or more interfaces. Other bundles can go to the registry and list all objects that are registered under a specific interfaces or class.

A bundle can register a service, get a service, and listen for a service to appear or disappear. Any number of bundles can register the same service type, and any number of bundles can get the same service.

This is shown in the following figure.



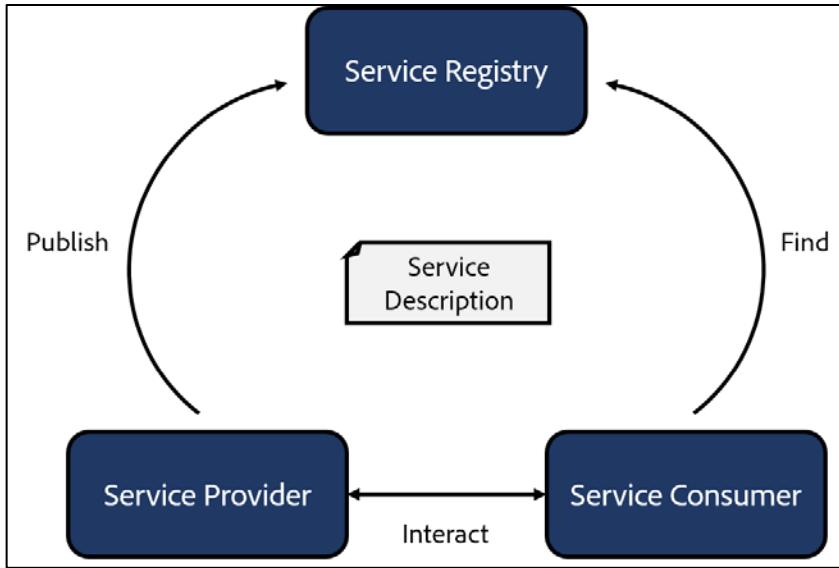
Each service registration has a set of standard and custom properties. An expressive filter language is available to select only the services in which you are interested. You can use properties to find the proper service or they can play other roles at the application level.

Services are dynamic. This means a bundle can “decide” to withdraw its service from the registry while other bundles are still using this service. Bundles using such a service must then ensure they no longer use the service object and drop any references. OSGi applications do not require a specific start ordering in their bundles.

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” is injected.

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

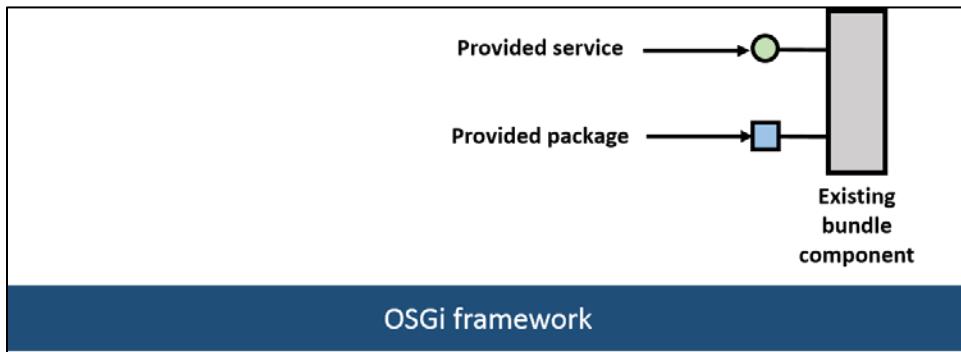


Instead of registering a listener/consumer object with the service provider, the consumer creates an object that implements the OSGi listener interface, providing a method that should be called when the event of interest occurs. When the desired event occurs, the service provider requests a list of all the services of the same object type 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 the OSGi framework is aware of bundle status and lifecycle and will unregister a bundle's services when the bundle stops.

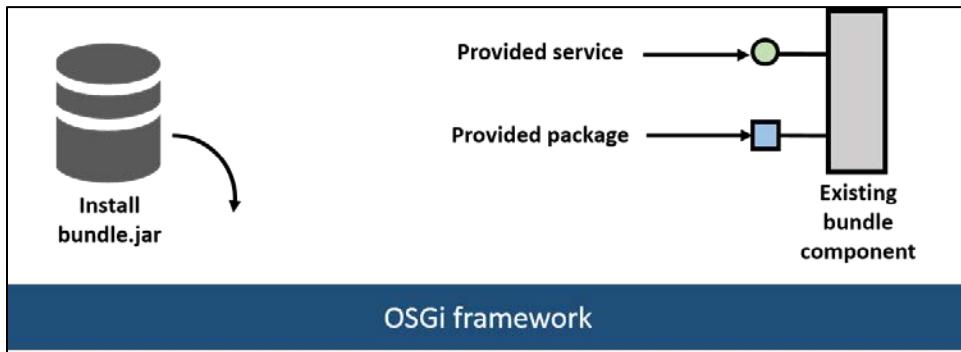
Dynamic Service Lookup

The following steps show how the lookup is managed for dynamic services.

- Existing service



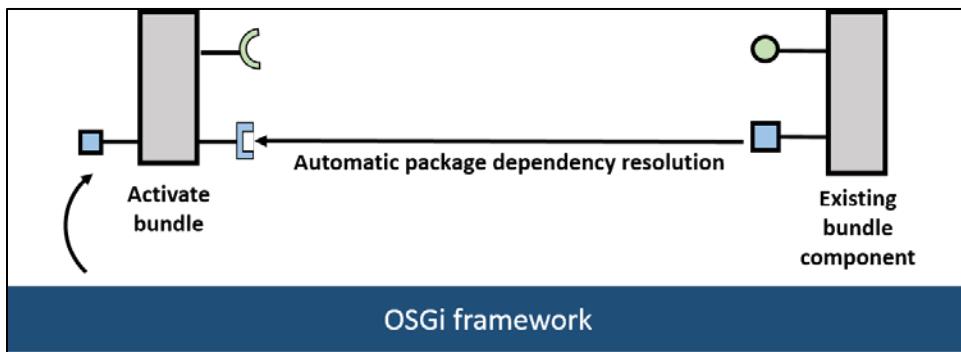
- Install new bundle.



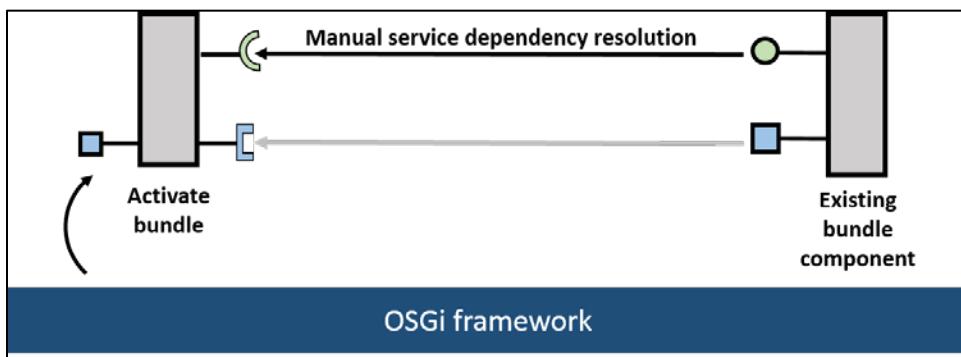
- Activate new bundle.



d. Automatic resolution of package dependency.



e. 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 that defines the life cycle of the service. A service object may be instantiated when the bundle is started and is automatically removed when the bundle is stopped.

The advantages of OSGi services are as follows:

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

Declarative Services

Declarative Services are a part of the OSGi container, and simplify the creation of components that publish and/or reference OSGi Services.

Features:

- No need to write explicit code to publish or consume services.
- Components that publish services are delayed, meaning the service implementation class is not loaded or instantiated until the service is actually requested by a client.
- Components have their own lifecycle, bounded by the lifecycle of the bundle in which they are defined.
- Components can automatically receive configuration data from Configuration Admin.

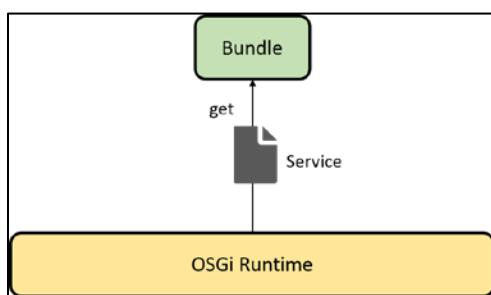
Components are declared using XML configuration files contained in the respective bundle and listed in the Service-Component bundle manifest header. Declarative Services resolve the bundle through the XML configuration files. You may handwrite and register these configuration files.

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 files, 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 and stops services
- invokes the optional activation and deactivation method
- provides access to bundle configuration



Deployment

Bundles are deployed on an OSGi framework—the bundle runtime environment. It is a collaborative environment, where the bundles run in the same VM and can actually share code. The framework uses the explicit imports and exports to wire up the bundles so they do not have to concern themselves with class loading. A simple API allows bundles to install, start, stop, and update other bundles, as well as enumerate the bundles and their service usage.

Benefits of OSGi

- **Reduced Complexity:** Bundles are the components of OSGi, and these bundles are modules. Due to the lack of interdependencies between these modules, developers have more freedom to change the modules at a later stage, which helps to reduce the number of bugs and simplify development.
- **Reuse:** The OSGi component model makes it very easy to use many third-party components in an application. An increasing number of open source projects provide their JARs readymade for OSGi. Commercial libraries are also available as readymade bundles.
- **Real World:** The OSGi framework is dynamic. As the real world is highly dynamic, having dynamic services that come and go make the services a perfect match for many real world scenarios. Applications can therefore reuse the powerful primitives of the service registry in their own domain. This not only saves writing code, it also provides global visibility, debugging tools, and more functionality implemented for a dedicated solution.
- **Easy Deployment:** The OSGi technology is not just a standard for components. It also specifies how components are installed and managed. The standardized management API makes it very easy to integrate OSGi technology in existing and future systems.
- **Dynamic Updates:** The OSGi component model is a dynamic model. You can install, start, stop, update, and uninstall bundles without bringing down the whole system.
- **Adaptive:** The dynamic service model of OSGi allows bundles to find out what capabilities are available on the system and adapt the functionality they can provide. This makes code more flexible and resilient to changes.
- **Transparency:** The management API provides access to the internal state of a bundle as well as how it is connected to other bundles. For example, most frameworks provide a command shell that shows this internal state. You can stop parts of the applications to debug a certain problem or bring in diagnostic bundles. OSGi applications can often be debugged with a live command shell.
- **Versioning:** In the OSGi environment, all bundles are carefully versioned and only bundles that can collaborate are wired together in the same class space. This allows various bundles to function with their own library.
- **Simple:** A number of easy-to-use annotations tell the runtime how a particular class wants to use the dynamics, configuration, and dependencies on other services. The defaults completely hide the dynamics and OSGi. This simple model allows the gradual use of more advanced features.

- **Small:** The OSGi Release 4 Framework can be implemented in about a 300KB JAR file. This is a small overhead for the amount of functionality added to an application by including OSGi. Therefore, OSGi runs on a large range of devices—from very small to small, to mainframes. It only asks for a minimal Java VM to run and adds very little on top of it.
- **Fast:** One of the primary responsibilities of the OSGi framework is loading the classes from bundles. OSGi pre-wires bundles and knows for each bundle exactly which bundle provides the class. This lack of searching is a significant speed-up factor at startup.
- **Secure:** The OSGi security model leverages the fine grained security model but improves the usability by having the bundle developer specify the requested security details in an easily audited form while the operator of the environment remains fully in charge.
- **Non-Intrusive:** Applications (bundles) in an OSGi environment are left on their own. They can use virtually any facility of the VM without the OSGi restricting them. Best practice in OSGi is to write Plain Old Java Objects and for this reason, there is no special interface required for OSGi services—even a Java String object can act as an OSGi service. This strategy makes application code easier to port to another environment.
- **Runs Everywhere:** Two issues are taken care of by the OSGi specification. First, the OSGi APIs do not use classes that are not available on all environments. Second, a bundle does not start if it contains code that is not available in the execution environment.



Perform Task –[Exercise 1- Implement a Bundle Activator](#) from **Lab G**.

Understanding Sling Events

Events are used to trigger jobs or workflows. Sling enables you to manage these events within your application. 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 simple and lightweight. Sending an event involves generating the event object and calling the event admin. Receiving an event requires implementation of a single interface and a declaration, through properties, on the interested topics. Each event is associated with an event topic, and event topics are hierarchically organized.

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

Various types of events can be handled:

- Predefined events include Framework events. For example, a Bundle event, which indicates a change in a bundle's lifecycle.
- Service events, which indicate a service lifecycle change
- Configuration events, which indicate a configuration was updated or deleted.
- JCR observation events
- Application-generated events
- Events from messaging systems (~JMS)
- External events

You can find details regarding to the events in the Web Console, under the Sling and OSGi tabs.

<http://localhost:4502/system/console/events>

<http://localhost:4502/system/console/slingevent>

Listening to OSGi Events

The event mechanism is a mechanism that must:

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

Events are received by the event mechanism and distributed to registered listeners. Concepts like durable listeners, guarantee of processing, and so on are not part of the event mechanism itself.

To listen to OSGi events in Sling, you must register an `org.osgi.service.event.EventHandler` service with an `event.topics` property that describes which event topics the handler is interested in.

For example:

```
1. @scr.property name="event.topics"
2. valueRef="ReplicationAction.EVENT _ TOPIC"
```

or

```
2. @scr.property name="event.topics"
3. valueRef="org.apache.sling.api.SlingConstants.TOPIC _ RESOURCE _ ADDED"
```

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

```
1. @Component
2. @Property(name = "event.topics", value =
3. ReplicationAction.EVENT_TOPIC)
4. @Service(value = EventHandler.class)
5. public class MyEventListener implements JobConsumer,EventHandler {
6. public void handleEvent(Event event) {
7. if (EventUtil.isLocal(event)) {
8. JobUtil.processJob(event, this);
9. }
10. }
```

Components and Annotations in OSGi

The following sections describe the purpose and use of components in OSGi, as well as the annotations used for service components and Java compilers.

7.1.1 Components

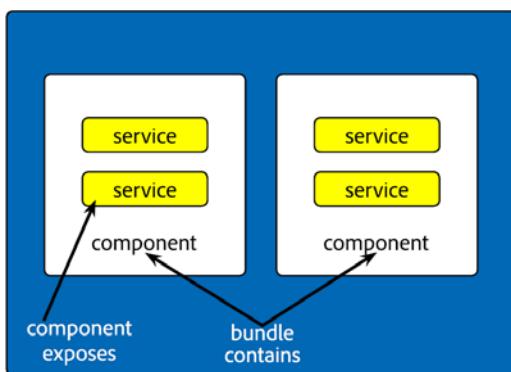
Components are the main building blocks for OSGi applications. Components in OSGi are, by definition, provided by a bundle. A bundle will contain and provide one or more components.

Therefore, a component is a:

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

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

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



A component is similar to a runtime service. They can publish themselves as a service and/or can have dependencies on other components and services. These dependencies will influence their lifecycle because 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 service can be consumed/used by components and other services. Basically, a bundle needs the following to become a component:

- XML file, where you describe the service the bundle provides and the dependencies of other services of the OSGi framework
- Manifest file header entry to declare the bundle behaves as a component
- Activate and Deactivate methods in the implementation class (or bind and unbind methods)
- Service Component Runtime (SCR). A service of the OSGi framework to manage these components: Declarative service of Equinox or Apache Felix SCR



NOTE: By default, a device is only started if someone else uses it. The immediate flag forces a service start. The OSGi Runtime calls the bundle - not the other way around. Do not confuse OSGi components with Adobe Experience Manager components!

As a best practice, always upload the bundle using the JCR. That way, the release engineers and system administrators have one common mechanism for managing bundles and configurations.

Bundles

You can upload bundles into the Felix container through the 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 deletion of the Felix bundle by Sling. Subsequent replacement of the JAR file with a new version in the JCR will create the new version of the bundle in the Felix container.

Annotations

You can annotate OSGi components using the annotations provided by the [org.apache.felix.dependencymanager.annotation](#) bundle.

The following annotations are supported:

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

@Component

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

The `@Component` annotation is the only required annotation.

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

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

@Component Modifiers

The following are some of the attributes you can use with the @Component annotation.

- **metatype:** Indicates whether the Metatype Service data is generated or not. If this parameter is set to true, then the Metatype Service is generated in the metatype.xml file for this component. The properties defined in @Property are configurable in the Web Console or sling:OsgiConfig nodes.
- **immediate:** Indicates whether the component is immediately activated.
- **enabled:** Indicates whether the component is enabled when the bundle starts.
- **label:** Serves as a title for the object described by the metatype. You can localize this name by prepending a % sign to the name.
- **description:** Provides a description for the object described by the metatype. You can localize this name by prepending a % sign to the name.

Example:

```
@Component (metatype=true, immediate=true, label="Hello Bundle",  
description="This is a bundle")
```

@Activate, @Deactivate, and @Modified

The OSGi Declarative Service allows you to specify the name for the activate, deactivate, and modified methods. In the following code, note the @Activate and @Deactivate annotations. These actions specify what happens when the component is activated and deactivated.

1. package com.adobe.osgitraining.impl;
2. import org.apache.felix.scr.annotations.Activate;
3. import org.apache.felix.scr.annotations.Component;
4. import org.apache.felix.scr.annotations.Deactivate;
5. @Component
6. **public class** MyComponent {
7. @Activate
8. **protected void** activate() {

```
9. // do something
10. }
11. @Deactivate
12. protected void deactivate() { // do something
13. }
14. }
```

@Service

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

```
1. package com.adobe.osgitraining.impl;
2. import org.apache.felix.scr.annotations.Component;
3. import org.apache.felix.scr.annotations.Service;
4. import org.osgi.service.event.EventHandler
5. @Component
6. @Service(value=EventHandler.class)
7. public class MyComponent implements EventHandler {
```

@Reference

The `@Reference` annotation defines references to other services. These other services (consumed services) are made available to the component by the SCR. This annotation declares the `<reference>` elements of the component declaration. The `@Reference` annotation may be declared on a Class level or on any Java field to which it might apply. Depending on where the annotation is declared, the parameters may have different default values.

@Property

The `@Property` annotation defines properties that 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 declares `<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.

```

1. @Component
2. @Service(value=EventHandler.class)
3. @Properties({
4.     @Property(name="event.topics", value="*",
5.     propertyPrivate=true),
6.     @Property(name="event.filter", value="(event.
7. distribute=*"),
8.     propertyPrivate=true)
9. })
10. public class DistributeEventHandler
11. implements EventHandler {
12.     protected static final int DEFAULT_CLEANUP_PERIOD = 15
13.     @Property(intValue=DEFAULT_CLEANUP_PERIOD)
14.     private static final String PROPERTY_CLEANUP_
15.     PERIOD=cleanup.period;
16.     @Reference
17.     protected ThreadPool threadPool;
18.     @Activate
19.     protected void activate (final Map<String, Object>props){
20.         this.cleanupPeriod = toInt(props.get(PROP_CLEANUP_PERIOD));
21.     }

```

Additionally, you may set properties 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

Note that the following annotations are defined by the Java compiler.

@Override

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

@SuppressWarnings

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

Configurable Services

The OSGi Configuration Admin Service provides for configuration storage and for the delivery of the configuration automatically or on demand to clients. Configuration objects are identified by 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 UI to configure the service and/or component.

LAB F – IMPLEMENT OSGI CONFIGURATION

Overview

In this lab, you will implement a bundle activator class, create and use a custom service, and code in SCR properties. You will also listen to and handle OSGi events posted in the Event Admin Console.

Objectives

- Implement a Bundle activator
- Create and use a custom service
- Code OSGi configurations
- Handle OSGI Events

Prerequisites

- AEM installed on your machine:
 - Running Adobe Experience Manager Author instance
 - An Eclipse project from the AEM Archetype

Directions

Complete the exercises that follow.

Exercise 1- Implement a Bundle Activator

Overview

In this exercise, you will implement a bundle activator class that exposes a start method when the bundle is started and run a stop method when the bundle is stopped.

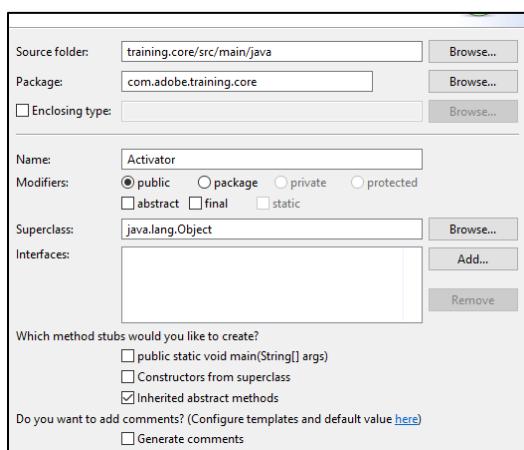
Steps

1. Create a new java class named Activator.



Note: To Activate code inside our bundle, we need to implement the org.osgi.framework.BundleActivator interface.

- a. In Eclipse, navigate to *training.core > src/main/java*
- b. Right-click **com.adobe.training.core** and choose *New > Class*



- Name: **Activator**

- c. Accept all other default settings and click **Finish**.



Note: The code is provided as part of the Exercise_Files under /Exercise_Files/07_Deep_Dive_OSGi/. Copy and paste the code from the exercise file referenced to Activator.java in Eclipse. Do not copy the code from this exercise book. The following is for illustrative purposes only.

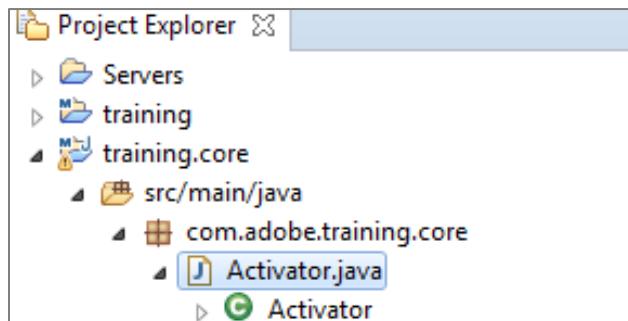
```

1. package com.adobe.training.core;
2.
3. import org.osgi.framework.BundleActivator;
4. import org.osgi.framework.BundleContext;
5.
6. import org.slf4j.Logger;
7. import org.slf4j.LoggerFactory;
8.
9. public class Activator implements BundleActivator {
10.     private final Logger logger = LoggerFactory.getLogger(getClass());
11.
12.     /*
13.      * (non-Javadoc)
14.      * @see org.osgi.framework.BundleActivator#start(org.osgi.framework.BundleContext)
15.     */
16.     public void start(BundleContext context) throws Exception {
17.         logger.info("#####Bundle Started#####");
18.     }
19.
20.     /*
21.      * (non-Javadoc)
22.      * @see org.osgi.framework.BundleActivator#stop(org.osgi.framework.BundleContext)
23.     */
24.     public void stop(BundleContext context) throws Exception {
25.         logger.info("#####Bundle Stopped#####");
26.     }
27.
28. }

```



NOTE: When you click Finish, the .java extension is added to the file name. Activator.java uses the start() and stop() methods to create log entries based on the bundle status.



2. Add the bundle to the POM file.

- Navigate to **training.core > pom.xml** and edit **pom.xml**
- Add the following line to the configuration section as shown (around line 47):

<Bundle-Activator>com.adobe.training.core.Activator</Bundle-Activator>

```
<!--
<Embed-Dependency>
    artifactId1,
    artifactId2;inline=true
</Embed-Dependency>
-->
<Sling-Model-Packages>com.adobe.training.core</Sling-Model-Packages>
<Bundle-Activator>com.adobe.training.core.Activator</Bundle-Activator>
</instructions>
</configuration>
</plugin>
</plugins>
```

c. Save the changes



Note: This line needs to be added for the Activator class to work.

3. Do a project update by right-clicking on **training.core** and selecting *Maven > Update Project*.

4. Deploy the project into AEM.

- In Project Explorer, right-click **training.core** and choose *Run As > Maven install*
- Verify the bundle installed successfully:

The screenshot shows the Eclipse IDE's Console view with the title 'Console'. The output window displays the results of a Maven 'install' command. The log shows the following information:

```
<terminated> C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Jan 27, 2017, 12:21:54 PM)
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ training.core
[INFO] Installing C:\AEM\workspace123\training\core\target\training.core-0.0.1-SNAPSHOT.jar to C:\Users\...
[INFO]
[INFO] --- maven-bundle-plugin:2.5.3:install (default-install) @ training.core
[INFO] Installing com/adobe/training/core/0.0.1-SNAPSHOT/training.core-0.0.1-SNAPSHOT.jar ...
[INFO] Writing OBR metadata
[INFO]
[INFO] --- maven-sling-plugin:2.1.0:install (default) @ training.core -
[INFO] Installing Bundle com.adobe.training.core(C:\AEM\workspace123\tr...
[INFO] Bundle installed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.767 s
[INFO] Finished at: 2017-01-27T12:22:02-08:00
[INFO] Final Memory: 25M/295M
[INFO] -----
```

5. Examine the log file on custom logger.
 - a. On your desktop, navigate to \crx-quickstart\logs
 - b. Open **project-trainingproject.log** using a text editor
 - c. Scroll down to the bottom and observe the log messages indicating the bundle has started:



```
File Edit Format View Help
27.01.2017 12:22:03.086 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent STOPPING
27.01.2017 12:22:03.086 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent STOPPED
27.01.2017 12:22:03.102 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent UNRESOLVED
27.01.2017 12:22:03.102 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent UPDATED
27.01.2017 12:22:03.121 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent RESOLVED
27.01.2017 12:22:03.121 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent STARTING
27.01.2017 12:22:03.121 *INFO* [OsgiInstallerImpl] com.adobe.training.core.Activator
#####
#####Bundle Started#####
27.01.2017 12:22:03.121 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service
[com.adobe.training.core.servlets.SimpleServlet,4894, [javax.servlet.Servlet]] ServiceEvent REGISTERED
27.01.2017 12:22:03.137 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service
[com.adobe.training.core.filters.LoggingFilter,4897, [javax.servlet.Filter]] ServiceEvent REGISTERED
27.01.2017 12:22:03.137 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service
[com.adobe.training.core.schedulers.SimpleScheduledTask,4899, [java.lang.Runnable]] ServiceEvent
REGISTERED
27.01.2017 12:22:03.137 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service
[com.adobe.training.core.listeners.SimpleResourceListener,4900, [org.osgi.service.event.EventHandler]]
ServiceEvent REGISTERED
27.01.2017 12:22:03.137 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent STARTED
```

- d. Close the log file and keep Eclipse open

You have successfully implemented a bundle activator in this exercise.

Exercise 2- Create and Use a Custom Service

Overview

In this exercise, you will create and use a custom service.

Steps

1. Create a new java class named **DeveloperInfo**.

- a. In Eclipse, navigate to *training.core > src/main/java*.
- b. Right-click **com.adobe.training.core** and choose *New > Class*.
 - Name: **DeveloperInfo**
 - Accept all other default settings and click **Finish**



Note: The code is provided as part of the Exercise_Files under /Exercise_Files/07_Deep_Dive_OSGi/. Copy and paste the code from the exercise file referenced to DeveloperInfo.java in Eclipse. Do not copy the code from this exercise book. The following is for illustrative purposes only.

```
1. package com.adobe.training.core;
2. /**
3. * Service interface to get the information about the bundle Developer
4. *
5. * Example HTL:
6. * <h3 data-sly-
use.devInfo="com.adobe.training.core.DeveloperInfo">Developer Info: ${devInfo.DeveloperInfo}
</h3>
7. *
8. * Example code can be inserted into a HTL component:
9. * /apps/trainingproject/components/structure/page/partials/main.html
10. *
11. * Example JSP:
12. * com.adobe.training.core.DeveloperInfo devInfo = sling.getService(com.adobe.training.core.De
veloperInfo.class)
13. * <h3>Developer Info: <%= devInfo.getDeveloperInfo() %></h3>
14. *
15. * @author Kevin Nennig (nennig@adobe.com)
16. */
17. public interface DeveloperInfo {
18.     public String getDeveloperInfo();
19. }
```

- Save the changes

2. Create another java class named **DeveloperInfoImpl**.

- a. In Eclipse, navigate to *training.core > src/main/java*.
- b. Right-click **src/main/java** and choose **New > Package**.
- c. Name: **com.adobe.training.core.impl**
- d. Click **Finish**.
- e. Right-click on **com.adobe.training.core.impl** and choose **New > class**.
 - Name: **DeveloperInfoImpl**
 - Accept all other default settings and click **Finish**.



Note: The code is provided as part of the Exercise_Files under /Exercise_Files/07_Deep_Dive_OSGi/. Copy and paste the code from the exercise file referenced to DeveloperInfoImpl-v1.java in Eclipse. Do not copy the code from this exercise book. The following is for illustrative purposes only.

```

1. package com.adobe.training.core.impl;
2.
3. import java.util.Map;
4.
5. import org.apache.felix.scr.annotations.Activate;
6. import org.apache.felix.scr.annotations.Component;
7. import org.apache.felix.scr.annotations.Deactivate;
8. import org.apache.felix.scr.annotations.Modified;
9. import org.apache.felix.scr.annotations.Service;
10. import org.slf4j.Logger;
11. import org.slf4j.LoggerFactory;
12.
13. import com.adobe.training.core.DeveloperInfo;
14.
15. /**
16. * Simple component implementation of the DeveloperInfo Service.
17. * @author Kevin Nennig (nennig@adobe.com)
18. */
19. @Component(metatype = true, label = "Training Developer Info")

```

```

20. @Service(DeveloperInfo.class)
21. public class DeveloperInfoImpl implements DeveloperInfo{
22.     private final Logger logger = LoggerFactory.getLogger(getClass());
23.
24.     @Activate
25.     //http://blogs.adobe.com/experiencedelivers/experience-
26.     management/osgi_activate_deactivatesignatures/
27.     protected void activate(Map<String, Object> properties) {
28.         configure(properties, "Activiated");
29.     }
30.
31.     @Modified
32.     protected void modified(Map<String, Object> properties) {
33.         configure(properties, "Modified");
34.     }
35.     @Deactivate
36.     protected void deactivated(Map<String, Object> properties) {
37.         logger.info("#####Component (Modified) Good-bye");
38.     }
39.
40.     protected void configure(Map<String, Object> properties, String status) {
41.         logger.info("#####Component (" +status+ ") " + getDeveloperInfo());
42.     }
43.
44.     public String getDeveloperInfo(){
45.         return "Hello! I do not know who my developer is. I am a product of random devel-
46.         opment!!!";
47.     }

```

- Save the changes
- f. Examine the method `getDeveloperInfo()` in the class.
3. Deploy the project into AEM.
- a. In Project Explorer, right-click `training.core` and choose ***Run AS > Maven install***.
 - Verify the bundle has installed successfully in the Console log at the bottom of the Eclipse application window.

4. Verify the bundle installed successfully in the OSGI.
 - a. Go to the Web Console: <http://localhost:4502/system/console>
 - b. Expand the TrainingProject – Core (com.adobe.training.core) by clicking the arrow.
 - c. Scroll down and look for the Service, **com.adobe.training.core.DeveloperInfo**.
 - d. Verify this service exists in the Bundles:

Service ID 4906	Vendor: Adobe Types: com.adobe.training.core.DeveloperInfo Service PID: com.adobe.training.core.impl.DeveloperInfoImpl Component Name: com.adobe.training.core.impl.DeveloperInfoImpl Component ID: 2639 Vendor: Adobe
-----------------	---

This indicates it was successfully installed in the OSGI.

5. Test the java logic by implementing the service on a web page.
 - a. Back in Eclipse, under Project Explorer, navigate *training.ui.apps > src/main/content/jcr_root > apps > trainingproject > components > structure > page > partials*
 - b. Open **main.html** in Text Editor
 - c. Add the following line to the main.html page:

```
<h3 data-sly-use.devInfo="com.adobe.training.core.DeveloperInfo">Developer Info:  
${devInfo.DeveloperInfo} </h3>
```

Note: This code is also available in the file under Exercise_Files/07_Deep_Dive_OSGi/html.txt.

```
<div class="page_title" data-sly-resource="${'title' @ decorationTagName='div'}></div>  
<h3 data-sly-use.devInfo="com.adobe.training.core.DeveloperInfo">Developer Info: ${devInfo.DeveloperInfo} </h3>  
<div class="page_par" data-sly-resource="par"></div>
```



Note: With HTL, we can expose the service. Here we are exposing our DeveloperInfo service. Once the service is exposed, you can call the method from the service.

- d. Save the changes.

6. Deploy the project into AEM.



Note: If you already setup your AEM Server connection in the Servers tab in Eclipse and started it, the main.html file would have synced with the JCR immediately after save.

- a. To verify whether your file successfully saved to the JCR, you can open CRXDE Lite to `/apps/trainingproject/components/structure/page/partials/main.html` and see if that file contains your change.
- b. Alternatively, you can force an update with a full redeploy to the JCR:
- c. In Project Explorer, right-click `training.ui.apps` and choose **Run AS > Maven Install**.
➤ Verify the bundle has installed successfully.

7. Test the service.

- a. Log in to AEM and click **Adobe Experience Manager** in the upper-left.
- b. Click *Navigation > Sites*.
- c. Select *TrainingProject Site > English*.

The screenshot shows the AEM navigation interface. At the top, there are buttons for Create, Edit, Properties, Lock, and Copy. Below that is a tree view of sites. The 'TrainingProject Site' node is selected and expanded. On the right, its properties are shown: a blue checkmark icon, the text 'English en', and below it 'Français fr'. The other site nodes in the tree are: 'Campaigns campaigns', 'Screens screens', 'Community Sites sites', 'We.Retail we-retail', and 'TrainingProject Site trainingproject'.

At the top of the page, you will see a navigational bar with options.

- d. Click **Edit**.
- e. Verify you see the message:

The screenshot shows a user interface for managing a service component. At the top right, there are language selection buttons for English, Spanish, and Français. The main content area has a red header bar with the text "HelloWorldModel says:" followed by the message "Hello World!". Below this, it displays the instance ID "This is instance: 10d0b03e-d358-4748-b521-0780bb4b22a5" and the resource type "Resource type is: trainingproject/components/content/helloworld". A section titled "Lorem ipsum" contains placeholder text about a ipsum dolor sit amet sentence. At the bottom left, there is a small "Save" button.

You have successfully created and consumed a custom service.

Exercise 3- Code OSGi Configurations

Overview

In this exercise, you will code a custom OSGi configurations for an OSGi component.

Steps

1. Modify the java class named **DeveloperInfoImpl**.
 - a. In Eclipse, navigate to **training.core > src/main/java > com.adobe.training.core.impl**
 - b. Double-click **DeveloperInfoImpl.java**
 - c. Copy the contents from
/Exercise_Files/07_Deep_Dive_OSGi /DeveloperInfoImpl-v2.java to **DeveloperInfoImpl.java** in Eclipse.



Note: The code is provided as part of the Exercise_Files under **/Exercise_Files/07_Deep_Dive_OSGi/**. Copy and paste the code from the exercise file referenced to **DeveloperInfoImpl-v2.java** in Eclipse. Do not copy the code from this exercise book. The following is for illustrative purposes only.

```
1. package com.adobe.training.core.impl;
2.
3. import java.util.Arrays;
4. import java.util.Map;
5.
6. import org.apache.felix.scr.annotations.Activate;
7. import org.apache.felix.scr.annotations.Component;
8. import org.apache.felix.scr.annotations.Deactivate;
9. import org.apache.felix.scr.annotations.Modified;
10. import org.apache.felix.scr.annotations.Property;
11. import org.apache.felix.scr.annotations.PropertyOption;
12. import org.apache.felix.scr.annotations.PropertyUnbounded;
13. import org.apache.felix.scr.annotations.Service;
14. import org.apache.sling.commons.osgi.PropertiesUtil;
15. import org.slf4j.Logger;
16. import org.slf4j.LoggerFactory;
17.
18. import com.adobe.training.core.DeveloperInfo;
19.
```

```
20. /**
21. * Component implementation of the DeveloperInfo Service. This gets the developer info from the OSGi Configuration
22. * There are 4 OSGi Configuration Examples:
23. * -Boolean
24. * -String
25. * -String Array
26. * -Dropdown
27. *
28. * @author Kevin Nennig (nennig@adobe.com)
29. */
30. @Component(metatype = true, label = "Training Developer Info")
31. @Service(DeveloperInfo.class)
32. public class DeveloperInfoImpl implements DeveloperInfo{
33.     private final Logger logger = LoggerFactory.getLogger(getClass());
34.
35.     //Boolean Field
36.     @Property(label = "Show Info", description = "Should the Developer information be shown?", boolValue = false)
37.     public static final String PROPERTY_BOOLEAN = "show.info";
38.     //String Field
39.     @Property(label = "Name", description = "Name of the Developer")
40.     public static final String PROPERTY_STRING = "name";
41.     //String Array Field
42.     @Property(label = "Hobbies", description = "List your favorite Hobbies", unbounded = PropertyUnbounded.ARRAY)
43.     public static final String PROPERTY_ARRAY = "hobbies";
44.     //Dropdown Field
45.     private static final String OPTION_1 = "HTML";
46.     private static final String OPTION_2 = "Java";
47.     private static final String OPTION_3 = "JSP";
48.     private static final String OPTION_4 = "HTML";
49.     private static final String OPTION_5 = "JavaScript";
50.     @Property(label = "Language", description = "Favorite Language Preference", options={
51.         @PropertyOption(name=OPTION_1,value=OPTION_1),
52.         @PropertyOption(name=OPTION_2,value=OPTION_2),
53.         @PropertyOption(name=OPTION_3,value=OPTION_3),
54.         @PropertyOption(name=OPTION_4,value=OPTION_4),
55.         @PropertyOption(name=OPTION_5,value=OPTION_5)}
56.     )
57.     public static final String PROPERTY_DROPDOWN = "language.preference";
58.
59.     //local variables to hold OSGi config values
60.     private boolean showDeveloper;
61.     private String developerName;
```

```
62. private String[] developerHobbies;
63. private String langPreference;
64.
65. @Activate
66. //http://blogs.adobe.com/experiencedelivers/experience-
management/osgi_activate_deactivatesignatures/
67. protected void activate(Map<String, Object> properties) {
68.     configure(properties, "Activiated");
69. }
70.
71. @Modified
72. protected void modified(Map<String, Object> properties) {
73.     configure(properties, "Modified");
74. }
75.
76. @Deactivate
77. protected void deactivated(Map<String, Object> properties) {
78.     logger.info("#####Component (Modified) Good-
bye " + developerName);
79. }
80.
81. protected void configure(Map<String, Object> properties, String status) {
82.     showDeveloper = PropertiesUtil.toBoolean(properties.get(PROPERTY_BO
OLEAN), false);
83.     developerName = PropertiesUtil.toString(properties.get(PROPERTY_STRIN
G), "Scott Reynolds");
84.     developerHobbies = PropertiesUtil.toStringArray(properties.get(PROPERTY
_ARRAY), new String[]{"Triangles", "Circles"});
85.     langPreference = PropertiesUtil.toString(properties.get(PROPERTY_DROP
DOWN), OPTION_1);
86.
87.     logger.info("#####Component (" + status + ") " + getDeveloperInf
o());
88. }
89.
90. /**
91. * Method used to show how OSGi configurations can be brought into a OSGi
component
92. */
93. public String getDeveloperInfo(){
94.     if(showDeveloper)
95.         return "Created by " + developerName
96.             + ". Hobbies include: " + Arrays.toString(developerHobbies)
97.             + ". Preferred programming language in AEM is: " + langPreference;
98.     return "";
99. }
```

```

100.
101. /*
102. * Method used to show a simple OSGi service/component relationship
103. public String getDeveloperInfo(){
104.     return "Hello! I do not know who my developer is. I am a product of random development!!!";
105. }
106. */
107. }
```

2. Examine the modified code. Notice the name of the component at the beginning of the class Training Developer Info.



Note: This is name of the configuration that will appear in the Web Console under OSGI > Configuration.

```

    * -Dropdown
    *
    * @author Kevin Nennia (knennia@adobe.com)
    */
@Component(metatype = true, label = "Training Developer Info")
@Service(DeveloperInfo.class)
```

- a. Examine the SCR annotation called property.

This annotation creates the configuration properties in the OSGI configuration console.

Common property types include:

- Boolean
- String
- String[]
- Dropdown

- b. Examine the various methods used in the class such as activate(), modified(), deactivate(), configure() and getDeveloperInfo() to understand the logic behind it
- c. Save the changes.

3. Deploy the project into AEM.
 - a. In Project Explorer, right-click **training.core** and choose **Run AS > Maven install**
 - b. Verify the bundle installed successfully.

4. Test to see if your OSGi component, **DeveloperInfoImpl**, has an OSGi configuration now.

- a. Navigate to the Web Console in AEM (<http://localhost:4502/system/console/configMgr>).
- b. Under **OSGi > Configuration**, search for **Training Developer Info**

Training Developer Info	
Description for com.adobe.training.core.impl.DeveloperInfoImpl	
Show Info	<input type="checkbox"/>
▲ Should the Developer information be shown? (show.info)	
Name	<input type="text"/>
▲ Name of the Developer (name)	
Hobbies	<input type="text"/>
▲ List your favorite Hobbies (hobbies)	
Language	<input type="text" value="HTL"/>
▲ Favorite Language Preference (language.preference)	
Configuration Information	
Persistent Identity (PID)	com.adobe.training.core.impl.DeveloperInfoImpl
Configuration Binding	Unbound or new configuration

- Enter the following values for the fields:
- Show Info: True (Checked)
- Name: **Scott Reynolds**
- Hobbies: **Hiking**
- Language: **HTL**
- Save the changes.



Note: Even though in this exercise we modified the configuration in the OSGi configurations console, Adobe recommends creating OSGi configuration nodes in the JCR for production.

5. Verify the information saved in OSGI.
 - a. Log in to AEM and navigate to Sites
 - b. Choose *TrainingProject Site > English > Edit* to open the page in a new tab in your browser.
 - c. Verify the following message:

The screenshot shows the AEM edit interface for the 'English' site. At the top right, there are language selection buttons for 'English' and 'Français'. On the left, there's a red vertical bar with the 'Adobe' logo. The main content area has a green header 'ENGLISH'. Below it, a section titled 'Developer Info' contains the text: 'Developer Info: Created by Scott Reynolds. Hobbies include: [Hiking, camping, Biking]. Preferred programming language in AEM is: HTL'. Underneath this, a section titled 'Service Component' displays the output of a HelloWorldModel component. The output text is:
HelloWorldModel says:
Hello World!
This is instance: da565b8a-f210-40db-9579-7446210153de
Resource type is: trainingproject/components/content/helloworld

Exercise 4 – Handle OSGi Events

Overview

In this exercise, you will “listen” for the replication OSGi event. You will write a class to handle events for the replication action which might be a publish or unpublish event. When the publish or unpublish action happens, this event handler will be triggered.

Steps

1. Create a new java class named **ReplicationListener**.
 - a. In Eclipse, navigate to **training.core > src/main/java**
 - b. Right-click **com.adobe.training.listeners** and choose **New > Class**
 - Name: **ReplicationListener**
 - Accept all other default settings and click **Finish**.



Note: The code is provided as part of the Exercise_Files under /Exercise_Files/07_Deep_Dive_OSGi/. Copy and paste the code from the exercise file referenced to ReplicationListener.java in Eclipse. Do not copy the code from this exercise book. The following is for illustrative purposes only.

```
1. package com.adobe.training.core.listeners;
2.
3. import java.util.HashMap;
4.
5. import org.apache.felix.scr.annotations.Component;
6. import org.apache.felix.scr.annotations.Property;
7. import org.apache.felix.scr.annotations.Reference;
8. import org.apache.felix.scr.annotations.Service;
9.
10. import org.osgi.service.event.Event;
11. import org.osgi.service.event.EventHandler;
12. import org.slf4j.Logger;
13. import org.slf4j.LoggerFactory;
14. import com.day.cq.replication.ReplicationAction;
15. import com.day.cq.replication.ReplicationActionType;
16.
```

```
17. import org.apache.sling.event.jobs.JobManager;
18.
19. @Component(immediate = true)
20. @Service(value = EventHandler.class)
21. @Property(name = "event.topics", value = ReplicationAction.EVENT_TOPIC)
22.
23. public class ReplicationListener implements EventHandler {
24.     private final Logger logger = LoggerFactory.getLogger(getClass());
25.
26.     //dictates to the JobManager which JobConsumer will be instantiated in line 42
27.     private static final String TOPIC = "com/adobe/training/core/replicationjob";
28.
29.     @Reference
30.     private JobManager jobManager;
31.
32.     @Override
33.     public void handleEvent(final Event event) {
34.         ReplicationAction action = ReplicationAction.fromEvent(event);
35.
36.         if (action.getType().equals(ReplicationActionType.ACTIVATE)) {
37.             if (action.getPath() != null)
38.             {
39.                 try {
40.                     // Create a properties map that contains things we want to pass through the job
41.
42.                     HashMap<String, Object> jobprops = new HashMap<String, Object>();
43.                     jobprops.put("PAGE_PATH", action.getPath());
44.                     // Add the job
45.                     jobManager.addJob(TOPIC, jobprops);
46.                     logger.info("=====Topic: "+TOPIC+" with payload: "+action.getPath()
47.                         ()+" was added to the Job Manager");
48.
49.                 } catch (Exception e) {
50.                     logger.error("===== ERROR CREATING JOB : NO PAYLOAD WAS DEF
51. INED");
52.                     e.printStackTrace();
53.                 }
54.             }
55.         }
56.     }
57. }
```

2. Examine the code.

- a. Examine the method `handleEvent()` and observe the log message.

```
try {
    // Create a properties map that contains things we want to pass through the job
    HashMap<String, Object> jobprops = new HashMap<String, Object>();
    jobprops.put("PAGE_PATH", action.getPath());
    // Add the job
    jobManager.addJob(TOPIC, jobprops);
    logger.info("*****Topic: '"+TOPIC+"' with payload: '"+action.getPath()+"' was added to the Job Manager");
}
```

- b. Save the changes.

3. Deploy the project into AEM.

- a. In Project Explorer, right-click **training.core** and choose **Run As > Maven install**

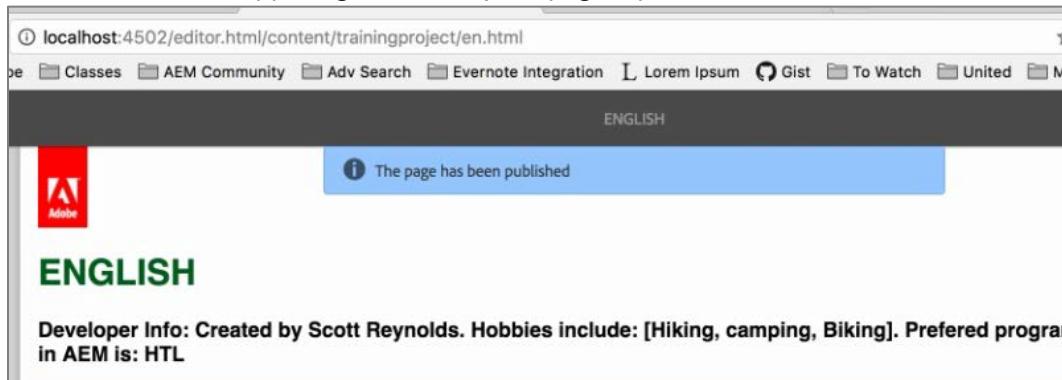
- Verify the bundle installed successfully:

```
<terminated> C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Jan 27, 2017, 12:21:54 PM)
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ train
[INFO] Installing C:\AEM\workspace123\training\core\target\training.core
[INFO] Installing C:\AEM\workspace123\training\core\pom.xml to C:\Users\...
[INFO]
[INFO] --- maven-bundle-plugin:2.5.3:install (default-install) @ traini
[INFO] Installing com/adobe/training.core/0.0.1-SNAPSHOT/training.core-
[INFO] Writing OBR metadata
[INFO]
[INFO] --- maven-sling-plugin:2.1.0:install (default) @ training.core -
[INFO] Installing Bundle com.adobe.training.core(C:\AEM\workspace123\tr
[INFO] Bundle installed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.767 s
[INFO] Finished at: 2017-01-27T12:22:02-08:00
[INFO] Final Memory: 25M/295M
[INFO] -----
```

4. Test the event handler by publishing a page.

- a. Log in to AEM and navigate to **Sites** and select **TrainingProject Site**.
- b. Open the **English** version of the page by selecting it, and clicking **Edit** in the action bar.
- c. Click the **Page Information** icon in the upper-left and select **Publish Page**. The Publish page appears, with All Assets and asset.jpg selected.

- d. Click **Publish** in the upper-right and verify the page is published:



5. Verify the **ReplicationListener** class logged the publish action.

- a. On your desktop, navigate to the **/crx-quickstart/logs** folder

- Open **project-trainingproject.log**.
- Scroll down and examine the log message:

```
[org.osgi.service.event.EventHandler] ServiceEvent REGISTERED
26.01.2017 23:46:04.672 +INFO+ [OsgiInstallerImpl] com.adobe.training.core Service [com.adobe.training.core.listeners.SimpleResourceListener,5735,
[org.osgi.service.event.EventHandler] ServiceEvent REGISTERED
26.01.2017 23:46:04.673 +INFO+ [OsgiInstallerImpl] com.adobe.training.core BundleEvent STARTED
26.01.2017 23:47:56.049 +INFO+ [0:0:0:0:0:1 [1485503275994] GET /content/trainingproject/en.html HTTP/1.1]
com.adobe.training.core.impl.DeveloperInfoImpl #####Component (Activated) Created by Scott Reynolds. Hobbies include: [Hiking, camping,
Biking]. Preferred programming language in AEM is: HTL
26.01.2017 23:51:40.697 +INFO+ [Thread-12] com.adobe.training.core.listeners.ReplicationListener =====Topic: 'com/adobe/training/core/
replicationjob' with payload: '/content/trainingproject/en' was added to the Job Manager
```

- b. This indicates your class is successful.

6. Verify the event in the Web Console.

- a. In AEM, navigate to the Web Console
- b. Select *OSGi > Events* from the menu

c. Notice all the events in the event log:

	resourceType	slingevent:job
1/26/2017, 11:51:40 PM org/apache/sling/api/resource/Resource/REMOVED		path /var/eventing/jobs/assigned/da565b8a-f210-40
	resourceRemovedAttributes	[event.job.retrydelay, cq:type, slingevent:appli
	event.topics	org/apache/sling/api/resource/Resource/REMOVED
	userId	admin
	:time	0
	event.job.retrydelay	60000
	cq:type	ACTIVATE
	slingevent:application	da565b8a-f210-40db-9579-7446210153de
	jcr:created	java.util.GregorianCalendar[time=1485503500625,ar
	slingevent:created	java.util.GregorianCalendar[time=1485503500622,ar
	event.job.queueName	com.day_cq.replication_job_test_and_target
	event.job.queuedTime	java.util.GregorianCalendar[time=1485503500622,ar
	jcr:createdBy	admin
	sling:resourceType	slingevent:Job
	slingevent:eventId	2017/1/26/23/51/da565b8a-f210-40db-9579-74462:
	event.job.application	da565b8a-f210-40db-9579-7446210153de
	event.job.retries	-1
	cq:path	/content/trainingproject/en
	cq:user	admin
	event.topics	org/apache/sling/event/notification/job/FINISHED
	event.job.startedTime	java.util.GregorianCalendar[time=1485503500633,ar
	jcr:primaryType	slingevent:Job
	event.job.retryCount	0
	event.job.topic	com/day/cq/replication/job/test_and_target
	cq:time	1485503500515
	timestamp	1485503500655
1/26/2017, 11:51:40 PM org/apache/sling/api/resource/ResourceResolverMapping/CHANGED	event.topics	org/apache/sling/api/resource/ResourceResolverMapping/CHAN
1/26/2017, 11:51:40 PM org/apache/sling/api/resource/ResourceResolverMapping/CHANGED	event.topics	org/apache/sling/api/resource/ResourceResolverMapping/CHAN
1/26/2017, 11:51:40 PM org/osgi/framework/ServiceEvent/REGISTERED	Service (id=5742, objectClass=org.apache.sling.event.jobs.jmx.StatisticsMBean)	
1/26/2017, 11:51:40 PM org/apache/sling/event/notification/job/ADDED	event.topics	org/apache/sling/event/notification/job/ADDED

d. Search for **training** and locate the following event entry:

- event.topics=com/day/cq/replication
- paths=[/content/trainingproject/en]
- type=ACTIVE

	userId	admin
	resourceType	slingevent:Job
1/26/2017, 11:51:40 PM org/apache/sling/api/resource/Resource/CHANGED	path	/content/trainingproject/en/jcr:co
	event.topics	org/apache/sling/api/resource/Resource/CHANGED
	resourceChangedAttributes	[cq:lastReplicated]
	userId	replication-service
	resourceType	trainingproject/components/structure
1/26/2017, 11:51:40 PM com/day/cq/replication	modificationDate	java.util.GregorianCalendar[time=(4855)135
	event.topics	com/day/cq/replication
	paths	[/content/trainingproject/en]
	type	ACTIVE
	userId	admin

Review:

- In this lab, you did the following:
 - Implemented an activator class
 - Created and used a custom service
 - Coded SCR properties
 - Listened for OSGi events and handled the events

8 DEEP DIVE INTO SLING

Overview

This module deep dives into the Sling architecture, focusing on Sling Models, Servlets, Job scheduling, and event Handling. The Lab Activity at the end of the module has hands-on exercises for writing servlets, event handlers, and job scheduling.

Objectives

By the end of this module, you will:

- Have in-depth knowledge of Sling architecture
- Write servlets, and job schedulers
- Use Sling Models in your project

The Sling Architecture

Apache Sling is an open source web application framework that makes it easy to develop content-oriented applications. Sling can be described as:

- Representational State Transfer (REST)-based web framework
- Content-driven, using a JCR content repository
- Powered by OSGi
- Using Scripts or Java Servlets to process HTTP requests

Sling applications are a set of OSGi bundles that use the OSGi core and compendium services. Apache Felix OSGi framework and console provide a dynamic runtime environment in which code and content bundles can be loaded, unloaded, and reconfigured at runtime.

Sling is resource-oriented, and maps into JCR nodes. In Sling, a request URL is first resolved to a resource, and then based on the resource, it selects the actual Servlet or script to handle the request.

RESTful Architecture

REST is a style of software architecture for distributed hypermedia systems such as the World Wide Web. Systems using REST architecture communicate through Hyper Text Transfer Protocol (HTTP), using GET and POST methods. The REST interface involves a collection of resources with identifiers. For example, /training/english.

REST is an architectural style:

- For network-based applications
- To induce the same architectural properties as the World Wide Web

Addressable resources present a uniform interface that allows transfers of state; for example, reading and updating of the resource's state. The best example of a RESTful architecture is the web, where resources have Uniform Resource Identifiers (URIs) and the uniform interface is HTTP.

This architectural style has the following properties:

- Performance and network efficiency
- Scalability
- Simplicity of interfaces
- Modifiability of components
- Visibility of communication
- Portability of components
- Reliability in resistance to internal failure

For most cases, a framework like HTTP (addressable resources plus "verbs" plus standard ways to transmit metadata) is all you need to build a distributed application. HTTP is a rich application protocol that gives you capabilities like content negotiation and distributed caching. RESTful web applications try to leverage HTTP in its entirety using specific architectural principles.

Advantages of REST

The following are known advantages of REST architecture.

- Uses well-documented, well-established, well-used technology and methodology
- Resource-centric rather than method-centric
- URI accessible by anyone
- No multiple protocols
- No specific format for response payload
- Uses the inherent HTTP security model
- Easy restriction of methods to URLs by firewall configuration, unlike other XML over HTTP messaging formats

Creating System Users

Prior to AEM 6.2, system users were not required to access OSGi services in AEM. After deprecating `session = repository.loginAdministrative(null);` in AEM 6.0, Adobe now suggests using service users to be created and configured for all sessions tied to OSGi services. Here is what the problem and solution are as identified in the Apache Sling Documentation.

Problem

To access the data storage in the Resource Tree and/or the JCR Repository, authentication is required to properly set up access control and guard sensitive data from unauthorized access. For regular request processing, this authentication step is handled by the Sling [Authentication](#) subsystem.

On the other hand, there are also some background tasks to be executed with access to the resources. Such tasks cannot in general be configured with user names and passwords: Neither hard coding the passwords in the code nor having the passwords in – more or less – plain text in some configuration is considered good practice.

To solve this problem for services to identify themselves and authenticate with special users properly configured to support those services.

The solution presented here serves the following goals:

- Prevent overuse and abuse of administrative ResourceResolvers and/or JCR Sessions
- Allow services access to ResourceResolvers and/or JCR Sessions without requiring to hard-code or configure passwords
- Allow services to use *service users* which have been specially configured for service level access (as is usually done on unixish systems)
- Allow administrators to configure the assignment of service users to services

Concept

A **SERVICE** is a piece or collection of functionality. Examples of services are the Sling queuing system, Tenant Administration, or a Message Transfer System. Each service is identified by a unique **SERVICE NAME**. Since a service will be implemented in an OSGi bundle (or a collection of OSGi bundles), services are named by the bundles providing them.

A Service may be comprised of multiple parts, so each part of the service may be further identified by a **SUBSERVICE NAME**. This Subservice Name is optional, though. Examples of **SUBSERVICE NAME** are names for subsystems in a Message Transfer System such as accepting messages, queueing messages, and delivering messages.

Ultimately, the combination of the **SERVICE NAME** and **SUBSERVICE NAME** defines the **SERVICE ID**. It is the **SERVICE ID**, which is finally mapped to a Resource Resolver and/or JCR Repository user ID for authentication.

Therefore, the actual service identification (service ID) is defined as:

1 service-id = service-name [":" subservice-name].

The service-name is the symbolic name of the bundle providing the service.

Implementation

The implementation in Sling of the **SERVICE AUTHENTICATION** concept described above consists of three parts:

ServiceUserMapper

The first part is a new OSGi Service ServiceUserMapper. The ServiceUserMapper service allows for mapping *Service IDs* comprised of the *Service Names* defined by the providing bundles and optional *Subservice Name* to ResourceResolver and/or JCR Repository user IDs. This mapping is configurable such that system administrators are in full control of assigning users to services.

The ServiceUserMapper defines the following API:

```
1 String getServiceUserID(Bundle bundle, String subServiceName);
```

ResourceResolverFactory

The second part is support for service access to the Resource Tree. To this avail, the ResourceResolverFactory service is enhanced with a new factory method.

```
1 ResourceResolver getServiceResourceResolver(Map<String, Object> authenticationInfo) throws  
2 LoginException;
```

This method allows for access to the resource tree for services where the service bundle is the bundle actually using the ResourceResolverFactory service. The optional Subservice Name may be provided as an entry in the authenticationInfo map.

In addition to having new API on the ResourceResolverFactory service to be used by services, the ResourceProviderFactory service is updated with support for Service Authentication: Now, new API is required though, but additional properties are defined to convey the service to authenticate for.



Perform Exercise – [Exercise 3 – Create a System User](#), from Lab G.

Working with Sling Servlets

Servlets can be registered as OSGi services. Sling applications use scripts or Java servlets, selected based on simple name conventions, to process HTTP requests in a RESTful way. Being a REST framework, Sling is oriented around resources, which usually map into JCR nodes. With Sling, a request URL is first resolved to a resource, and then based on the resource, it selects the actual servlet or script to handle the request.

The GET method has default behaviors, and therefore requires no additional parameters. By decomposing the URL, Sling can determine the resource URL and other information that can be used to process that resource.

The POST method is handled by the Sling PostServlet. The PostServlet enables writes to a data store (usually the JCR) directly from HTML forms or from the command line, using cURL.

You can view the existing servlets through the Web Console, or you can click here:

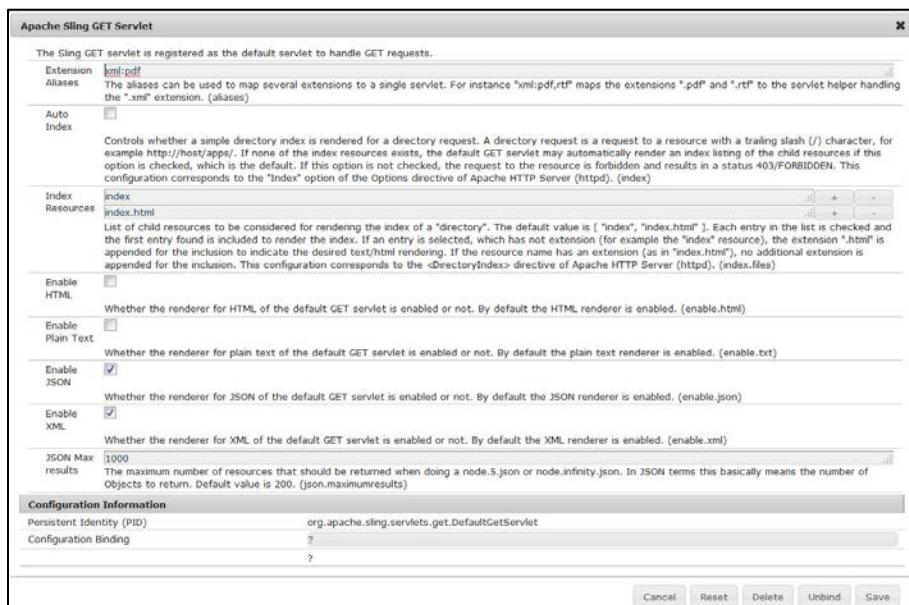
<http://localhost:4502/system/console/configMgr>

Configuring the Default Sling GET Servlet

Sling provides default renderers for the following mime types of the default GET servlet:

- txt
- JSON
- docview.xml
- sysview.xml
- html

You can use the Configuration tab of the Web Console to configure the GET servlet.



Configuring the Sling POST Servlet

The Sling POST Servlet provides a RESTful interface that lets you manipulate data content stored in the Adobe Experience Manager JCR. This servlet maps nodes in the JCR repository to URIs and allows applications to modify JCR content. A Sling POST Servlet is performed on the client using JavaScript.

To create content in the JCR, send an http POST request with the path of the node where you want to store the content and the actual content, sent as request parameters. For example, consider the following script:

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



NOTE: This code will work only if the `com.adobe.granite.csrf.impl.CSRFFilter` filter is defined in the OSGi configurations.

The above lines will set the title and text properties on a node in the location `/mycontent`. If this node does not exist, it will be created; otherwise, the existing content at that URL would be modified.

You can perform a similar operation using the following cURL commands:

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

You can use the Configuration tab of the Web Console to configure the Sling POST servlet.

The Sling POST Servlet is registered as the default servlet to handle POST requests in Sling.

Date Format

- EEE MMM dd yyyy HH:mm:ss 'GMT'Z
- ISO8601
- yyyy-MM-dd'T'HH:mm:ss.SSSZ
- yyyy-MM-dd'T'HH:mm:ss
- yyyy-MM-dd
- dd.MM.yyyy HH:mm:ss
- dd.MM.yyyy

Node Name Hint Properties

- title
- jcr:title
- name
- description
- jcr:description
- abstract
- text
- jcr:text

Maximum Node Name Length

20

Checkin New Versionable Nodes

If true, newly created versionable nodes or non-versionable nodes which are made versionable by the addition of the mix:versionable mixin are checked in. By default, false. (servlet.post.checkinNewVersionableNodes)

Auto Checkout Nodes

If true, checked in nodes are checked out when necessary. By default, false. (servlet.post.autoCheckout)

Auto

✓

Cancel Reset Delete Unbind Save



Perform Task – [Exercise 1 – Work with Servlets](#), from Lab G.



Perform Task – [Exercise 2 – Create a Sling Model](#) from Lab G.

SlingRepository

The third part is an extension to the SlingRepositoryservice interface to support JCR Repository access for services:

- 1 Session loginService(String subServiceName, String workspace) throws LoginException,
- 2 RepositoryException;

This method allows for access to the JCR Repository for services where the service bundle is the bundle actually using the SlingRepository service. The additional Subservice Name may be provided with the subServiceName parameter.

Deprecation of administrative authentication

Originally the ResourceResolverFactory.getAdministrativeResourceResolver and SlingRepository.loginAdministrative methods were defined to provide access to the resource tree and JCR Repository. These methods proved to be inappropriate because they allowed for much too broad access.

Consequently, these methods are being deprecated and will be removed in future releases of the service implementations.

The following methods are deprecated:

- ResourceResolverFactory.getAdministrativeResourceResolver
- ResourceProviderFactory.getAdministrativeResourceProvider
- SlingRepository.loginAdministrative

The implementations in Sling's bundle will remain implemented in the near future. However, there will be a configuration switch to disable support for these methods: If the method is disabled, a `LoginException` is always thrown from these methods. The JavaDoc of the methods is extended with this information.

Understanding the SLING Resolution Process

Sling is resource-oriented, and maintains all resources in the form of a virtual tree. A resource is usually mapped to a JCR node, but can also be mapped to a file system or database. Some of the common properties that a resource can have are:

- Path – Includes the names of all resources beginning from the root, each separated by a slash (/). It is similar to a URL path or file system path.
- Name – This is the last name in the resource path.
- Resource Type - Each resource has a resource type that is used by the Servlet and Script resolver to find the appropriate Servlet or Script to handle the request for the Resource.

When 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.

Resource First Request Processing

When a request URL comes in, it is first resolved to a resource. Then, based on the resource, it selects the servlet or script to handle the request.

Basic Steps of Processing Requests

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. After the content is determined, the script or servlet to be used to handle the request is determined in cascading resolution that checks in order of priority:

1. Properties of the content item itself
2. The HTTP method used to make the request
3. A simple naming convention within the URL that provides secondary information

For every URL request, the following steps are performed to get it resolved:

- a) Decompose the URL
- b) Search for a file indicated by the URL
- c) Resolve the Resource
- d) Resolve the rendering script/servlet
- e) Create rendering chain
- f) Invoke rendering chain

Decomposing the URL

Processing is done based on the URL requests submitted by the user. The elements of the URL are extracted from the request to locate and access the appropriate script.

Example - URL: <http://myhost/tools/spy.printable.a4.html/a/b?x=12>

The above URL can be decomposed into the following components:

Protocol	Host	Content path	Selector(s)	Extension	/	Suffix	?	Param(s)
http://	myhost	tools/spy	.printable.a4	html	/	a/b	?	x=12

The following table describes the components.

Protocol	Hypertext transfer protocol
Host	Name of the website
Content path	Path specifying the content to be rendered. It is used in combination with the extension.
Selector(s)	Used for alternative methods of rendering the content.
Extension	Content format; also specifies the script to be used for rendering.
Suffix	Can be used to specify additional information.

Param(s)	Any parameters required for dynamic content.
----------	--

Mapping Request to Resources

Once the URL is decomposed, the content node is located from the content path. This node is identified as the resource, and performs the following steps to map to the request.

Consider the URL request: `http://myhost/tools/spy.html`

- Sling checks whether a node exists at the location specified in the request. In the above example, it searches for the node **spy.html**.
- If no node is found at that location, the extension is dropped and the search is repeated. For example, it searches for the node **spy**.
- If no node is found, then Sling will return the http code 404 (Not Found).
- If a node is found, then the sling resource type for that node is extracted, and used to locate the script to be used for rendering the content.

Locating and Rendering Scripts

All scripts are stored in either the /apps or /libs folder, and are searched in the same order. If no matching script is found in either of the folders, then the default script is rendered. When the resource is identified from the URL, its resource type property is located and the value extracted. This value is either an absolute or a relative path that points to the location of the script to be used for rendering the content.

For multiple matches of the script, the script with the best match is selected. The more selector matches, the better.

Example

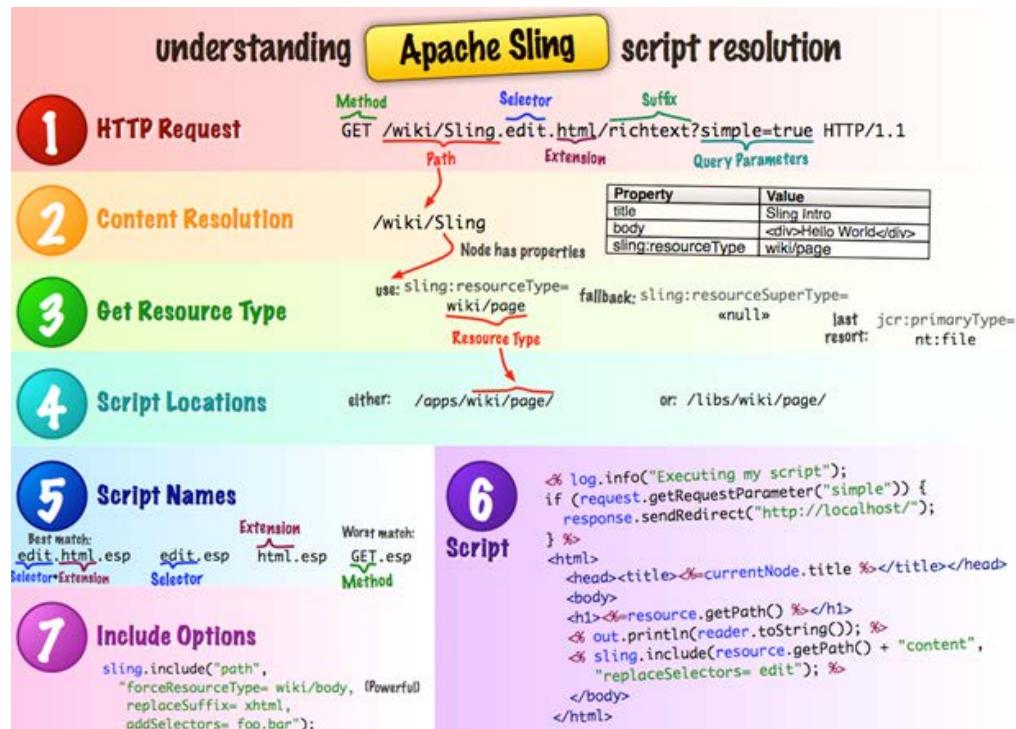
<p><u>Files in repository under hr/jobs</u></p> <pre> 1. GET.jsp 2. jobs.jsp 3. html.jsp 4. print.jsp 5. print.html.jsp 6. print/a4.jsp 7. print/a4/html.jsp 8. print/a4.html.jsp </pre>	<p><u>REQUEST</u> URL: /content/corporate/jobs/developer.print.a4.html sling:resourceType = hr/jobs</p> <p><u>RESULT</u> Order of preference: 8 > 7 > 6 > 5 > 4 > 3 > 2 > 1</p>
--	---

Super types are also taken into consideration when trying to locate 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:

1. sling:resourceSuperType property of the resource.
2. sling:resourceSuperType property of the node to which the sling:resourceType points.

To summarize how a script is resolved:



The Resource Resolver

The Resource Resolver is a part of Sling that resolves incoming requests to actual or virtual resources. For example, a request for /training/english will be resolved to a corresponding JCR node. However, if you do not want to expose the internal JCR structure, or if it cannot be resolved to a JCR node, you can define a set of resolver rules through the Web Console. You can also use the standard OSGi configuration mechanisms in the CRX to define a set of resolver rules.

The Resource Resolver abstracts:

- The path resolution
- Access to the persistence layer(s)

Resource mapping is used to define redirects, vanity URLs, and virtual hosts. You can access the Resolver Map entries through the Resource Resolver tab of the Web Console. You can access the console with this link: <http://localhost:4502/system/console/jcrresolver>

Mappings for Resource Resolution

The following node types help with the definition of redirects and aliases.

Node Types	Description
sling:ResourceAlias	Mixin node type defines the sling:alias property, and may be attached to any node, which does not allow the setting of a property named sling:alias .
sling:MappingSpec	Mixin node type defines the sling:match, sling:redirect, sling:status, and sling:internalRedirect properties.
sling:Mapping	The primary node type used to construct entries in /etc/map.

The following properties are important when dealing with new resources.

Properties	Description
sling:match	Defines a partial regular expression used on a 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 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.

Example of resource mapping

Consider 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\.\d*"
        | +- 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"
    +- regexmap
        +- sling:match = "$1.example.com/$2"
        +- sling:internalRedirect = "/content/([^\/]+)/(.*)"
```

Based on the above definition, we get the following mappings:

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.

Adapting Resources

An 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 Resource Resolver 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.adaptTo method with the javax.jcr.Node class object.

Adaptable.adaptTo() can be implemented in multiple ways:

- By the object itself, implementing the method itself and mapping to certain objects
 - By an AdapterFactory which can map arbitrary objects
- A combination of both

Publishing Events

To publish an event, you need to:

1. Get the EventAdmin service.
2. Create the event object (with a topic and properties).
3. Send the event (synchronously or asynchronously).

Sending Job Events

A job event is an event with the topic /org/apache/sling/event/job and the topic of the event is stored in the event.job.topic property. Job events are always processed, and are used in situations such as sending notification messages, or post-processing images. These events must be handled only once.

To send a job event, the service needs to implement:

- the org.osgi.service.event.EventHandler interface
- the org.apache.sling.event.JobConsumer interface

Implementing Event Handling

Event handling can be done at any of the following levels:

- JCR level, with observation
- Sling level, with event handlers and jobs
- Adobe Experience Manager level, with workflows and launchers



Perform Task – [Exercise 4 – Create a Job Consumer for a Topic](#), from Lab G.

Working with Sling Schedules

The Sling Scheduler allows you to easily schedule jobs within your application. Jobs can be executed at a specific time, regularly at a given period, or at the time given by a cron expression by leveraging the Sling scheduler service. It makes use of the open source Quartz library.

OSGi Service Fired by Quartz

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

```

1. package <package name>;
2. @Component
3. @Service (interface="java.lang.Runnable")
4. @Property (name="scheduler.expression" value="0 0/10 * * * ?", type="String")
5. public class MyScheduledTask implements Runnable {
6.     public void run() {
7.         //place events to run here.
8.     }
9. }
```

- The `@Component` annotation is used to register the component. This should 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 Jobs

The following sections describe various methods of scheduling jobs.

Scheduling at periodic times

Instead of specifying a Quartz expression for the time interval, in simpler cases, we can just specify a time period, and the job will run repeatedly at this interval in seconds.

For example, for a 10-second interval:

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

Preventing concurrent execution

If you do not want a concurrent execution of the job, you can prevent it using the `scheduler.concurrent` parameter:

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

Scheduling Jobs programmatically

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

```
1. @Reference  
2. private Scheduler scheduler;  
3. this.scheduler.addJob("myJob", job, null, "0 15 10 ? * MON FRI", true);  
4. // periodic:  
5. this.scheduler.addPeriodicJob("myJob", job, null, 3*60, true);  
6. // one time  
7. this.scheduler.fireJobAt("myJob", job, null, fireDate);
```

Working with Sling Models

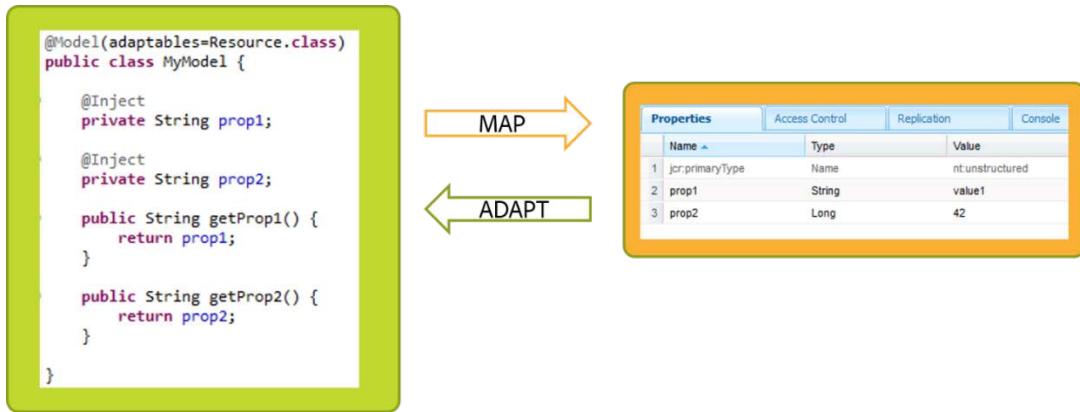
Sling Models let you map Java objects to Sling resources. When developing an Adobe Experience Manager project, you can define a model object (a Java object) and map that object to Sling resources. They have the following objectives:

- Entirely annotation-driven.
- Use standard annotations where possible.
- Pluggable
- OOTB, support resource properties, SlingBindings, OSGi services, request attributes
- Adapt multiple objects
- Support both classes and interfaces.
- Work with existing Sling infrastructure

Using Sling Models

Basically, you would use Sling Models when you have one of the following situations:

- You have a model object (as a Java class or interface)
- You need to use it with Adobe Experience Manager without creating your own adapters
- You need to map your Plain Old Java Object (POJO) into a Sling Resource



Benefits of Using Sling Models

- Saves time from creating own adapters (avoiding boilerplate code)
- Support both classes and interfaces
- Support OOTB (out-of-the-box) resource properties (via ValueMap), SlingBindings, OSGi services, request and attributes
- Allows you to adapt multiple objects - minimal required Resource and SlingHttpServletRequest
- Client doesn't know/care that these objects are different than any other adapter factory
- Works with existing Sling infrastructure (for example, not require changes to other bundles)
- Provides the ability to mock dependencies with tools like Mockito @InjectMocks

Understanding the Sling Model

Sling Models allow developers to inject method return values (in an interface) and fields (in a class) based on Sling resources and OSGi services. Out-of-the-box, they support resource properties (via ValueMap), SlingBindings, OSGi services, and request attributes. Most injections are available when adapting either a Resource or SlingHttpServletRequest object.

Sling models are entirely annotation-driven. A Sling Model is implemented as an OSGi bundle; a Java class located in the OSGi bundle is annotated with @Model and the Adaptable class (for example, @Model(adaptables = Resource.class)). The data members (fields) use @Inject annotations.

```

1. import javax.inject.Inject;
2.
3. import org.apache.sling.api.resource.Resource;
4. import org.apache.sling.models.annotations.Model;
5.
6. @Model(adaptables=Resource.class)
7. public class MyModel {
8.

```

```

9.     @Inject
10.    private String prop1;
11.
12.    @Inject
13.    private String prop2;
14.
15.    public String getProp1() {
16.        return prop1;
17.    }
18.
19.    public String getProp2() {
20.        return prop2;
21.    }
22. }
```

Sling Models let you to access Sling content without creating your own adapters, thus avoiding a large amount of boilerplate code.

In most cases, using a Sling Model Interface will require less code to accomplish the same task. In the case of Sling Model classes, the most common case where you would want to use them is if you need to do any filtering or manipulation of the values being returned. When using a class for your model, you can inject the values into the fields and generate a formatted return value.

In the simplest case, the class is annotated with @Model and the adaptable class. Fields that need to be injected are annotated with @Inject:

```

1.  @Model(adaptables=Resource.class)
2.  public class MyModel {
3.
4.    @Inject
5.    private String propertyName;
```

In this case, a property named propertyName will be looked up from the Resource (after first adapting it to a ValueMap) and it is injected.

For an interface, it is similar:

```

1.  @Model(adaptables=Resource.class)
2.  public interface MyModel {
3.
4.    @Inject
5.    String getPropertyName();
6. }
```

Client code does not need to be aware that Sling Models is being used. It just uses the Sling Adapter framework:

```
MyModel model = resource.adaptTo(MyModel.class)
```

If the field or method name doesn't exactly match the property name, you can use @Named:

```

1.  @Model(adaptables=Resource.class)
2.  public class MyModel {
3.
```

```

4.     @Inject @Named("secondPropertyName")
5.     private String otherName;
6. }
```

Annotation Usage

Sling Model Annotation	Code Snippet	Injector
@Model	@Model(adaptables = Resource.class)	Class is annotated with @Model and the adaptable class.
@Inject	@Inject private String propertyName; (class) @Inject String getPropertyName(); (interface)	A property named "propertyName" will be looked up from the Resource (after first adapting it to a ValueMap) and it is injected, else will return null if property not found.
@Default	@Inject @Default(values="AEM") private String technology;	A default value (for Strings, Arrays, primitives, and so on).
@Optional	@Inject @Optional private String otherName	@Injected fields/methods are assumed to be required. To mark them as optional, use @Optional for example, resource or adaptable may or may not have property.
@Named	@Inject @Named("title") private String pageTitle;	Inject a property whose name does not match the Model field name.
@Via	@Model(adaptables=SlingHttpServletRequest.class) Public interface SlingModelDemo { @Inject @Via("resource") String getPropertyName(); }	Use a JavaBean property of the adaptable as the source of the injection //Code snippet will return request.getResource().adaptTo(ValueMap.class).get("propertyName", String.class)
@Source	@Model(adaptables=SlingHttpServletRequest.class) @Inject @Source("script-bindings") Resource getResource();	Explicitly tie an injected field or method to a particular injector (by name). Can also be on other annotations. //Code snippet will ensure that "resource" is retrieved from the bindings, not a request attribute.
@PostConstruct	@PostConstruct	Methods to call upon model option

t	<pre>protected void sayHello() { logger.info("hello"); }</pre>	creation (only for model classes).
---	--	------------------------------------

Injector Specific Annotations

To create a custom injector, simply implement the org.apache.sling.models.spi.Injector interface and register your implementation with the OSGi service registry. Here's a list of the available injectors:

<https://sling.apache.org/documentation/bundles/models.html#available-injectors>

Sometimes, it is necessary to use customized annotations that aggregate the standard annotations described above. This generally has the following advantages over using the standard annotations:

- Less code to write (only one annotation is necessary in most of the cases)
- More robust (in case of name collisions among the different injectors, you must make sure the right injector is used)
- Better IDE support (because the annotations provide elements for each configuration which is available for that specific injector; for example, filter only for OSGi services)

The following annotations are provided, which are tied to specific injectors:

Annotation	Supported Optional Elements	Injector
@ScriptVariable	optional and name	script-bindings
@ValueMapValue	optional, name and via	valuemap
@ChildResource	optional, name and via	child-resources
@RequestAttribute	optional, name and via	request-attributes
@ResourcePath	optional, path, and name	resource-path
@OSGiService	optional, filter	osgi-services
@Self	optional	self
@SlingObject	optional	sling-object

Injectors Lookup in Web Console

List of available injectors in the Felix console:

<http://localhost:4502/system/console/status-slingmodels>



Adobe Experience Manager Web Console Sling Models

Main OSGi Sling Status Web Console

Date: January 7, 2016 9:49:03 PM CET

[Download As Text](#) [Download As Zip](#) [Download Full Text](#) [Download Full Zip](#)

Sling Models Injectors:
resource-resolver - org.apache.sling.models.impl.injectors.ResourceResolverInjector
script-bindings - org.apache.sling.models.impl.injectors.BindingsInjector
valuesource - org.apache.sling.models.impl.injectors.ValueMapInjector
resource-path - org.apache.sling.models.impl.injectors.ResourcePathInjector
child-resources - org.apache.sling.models.impl.injectors.ChildResourceInjector
request-attributes - org.apache.sling.models.impl.injectors.RequestAttributeInjector
osgi-service - org.apache.sling.models.impl.injectors.OSGiServiceInjector
sling-object - org.apache.sling.models.impl.injectors.SlingObjectInjector
self - org.apache.sling.models.impl.injectors.SelfInjector

Sling Models Inject Annotation Processor Factories:
org.apache.sling.models.impl.injectors.BindingsInjector
org.apache.sling.models.impl.injectors.ValueMapInjector
org.apache.sling.models.impl.injectors.ResourcePathInjector
org.apache.sling.models.impl.injectors.ChildResourceInjector
org.apache.sling.models.impl.injectors.RequestAttributeInjector
org.apache.sling.models.impl.injectors.OSGiServiceInjector
org.apache.sling.models.impl.injectors.SlingObjectInjector
org.apache.sling.models.impl.injectors.SelfInjector

Sling Models Implementation Pickers:
org.apache.sling.models.impl.FirstImplementationPicker

Debugging

In order to see logging specific to Sling Models, add a logger for the package org.apache.sling.models, set the log level to TRACE and dump into appropriate log file.



Perform Task – Exercise 5 – Clean up Nodes with a Sling Scheduler, from Lab G.

LAB G – EXPLORE DIFFERENT AREAS OF SLING

Scenario

You need to create servlets that can display the title of the page, as well as the properties of the nodes using Sling servlet capabilities.

When a page is replicated to the publish server, the action needs to be logged in the log file.

Monitor your repository for unwanted resources created by Adobe Experience Manager processes, and remove them accordingly.

You need to manage custom configurations using Adobe Experience Manager repository.

Display the name of a user stored in a different node in the repository.

Challenge

You need to use best practices while creating servlets, event handlers, and job schedules.

Objectives

- Work with Servlets
- Create a Sling Model
- Create a System User
- Create a JobConsumer for a Topic
- Clean-up Nodes with a Sling Scheduler

Prerequisites

- AEM installed on your machine:
- Running Adobe Experience Manager Author instance
- An Eclipse project from the AEM Archetype

Directions

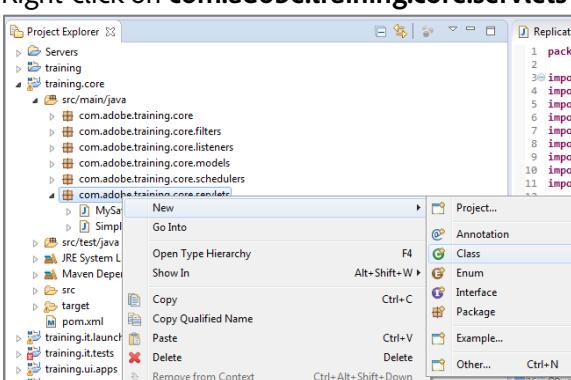
Complete the exercises that follow.

Exercise 1 – Work with Servlets

Overview

In this exercise, you will write a servlet that serves only DOGET requests from a specific URI or resource type. The response shall contain the JCR repository properties as JSON.

Steps

1. Open Eclipse and navigate to **training.core > src/main/java**
 - a. Right-click on **com.adobe.training.core.servlets** and select **New > Class**

 - b. Create a java class named **TitleSlingServlet** and click **Finish**.
 - c. Paste the following code in the exercises file and save the changes by selecting **File > Save**.



NOTE: The code is provided as part of the Exercise_Files under /Exercise_Files/08_Deep_Dive_Sling/. Copy and paste the code from the exercise files to Eclipse. Do not copy from this exercise book. The following is for illustrative purposes only.

```

1. package com.adobe.training.core.servlets;
2.
3. import java.io.IOException;
4. import javax.servlet.ServletException;
5. import org.apache.felix.scr.annotations.sling.SlingServlet;
6. import org.apache.sling.api.SlingHttpServletRequest;
7. import org.apache.sling.api.SlingHttpServletResponse;
8. import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
9.
10. @SlingServlet(paths="/bin/trainingproject/titleServlet")
11.
12.
13.
14. public class TitleSlingServlet extends SlingSafeMethodsServlet {

```

```

15.
16. private static final long serialVersionUID = 1L;
17.
18. @Override
19. protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException {
20.     response.setHeader("Content-Type", "text/html");
21.     response.getWriter().print("<h1>Sling Servlet injected this title</h1>");
22.     response.getWriter().close();
23. }
24.

```



Note the servlet **Path** mentioned in the code is **/bin/trainingproject/titleservlet**, so the request can be served on this path.

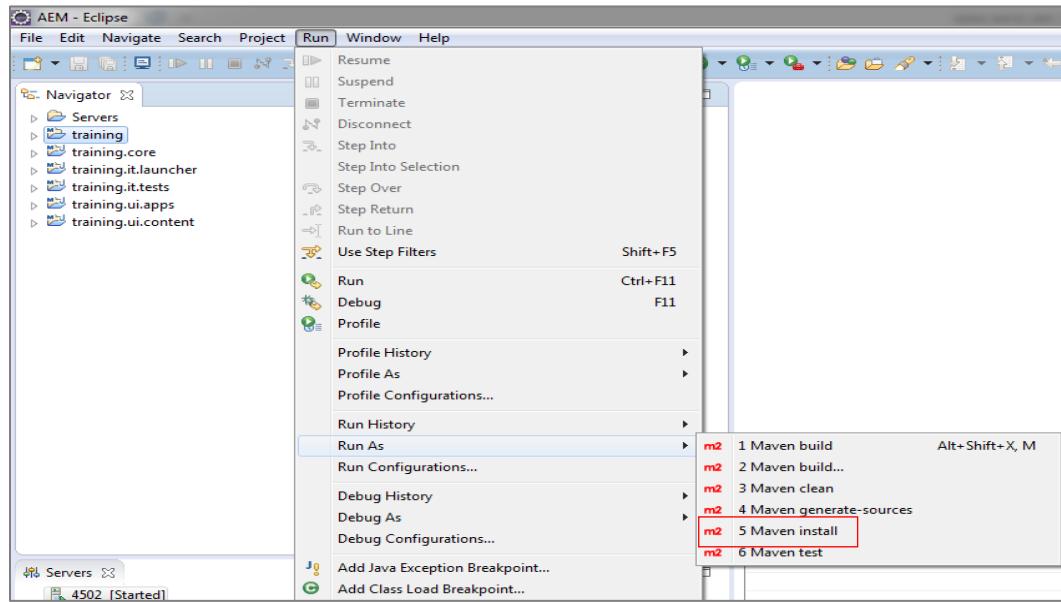
The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer view displays the project structure under 'training.core'. In the center, the code editor shows the 'TitleSlingServlet.java' file with the following code:

```

1 package com.adobe.training.core.servlets;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import org.apache.sling.scr.annotations.SlingServlet;
6 import org.apache.sling.api.SlingHttpServletRequest;
7 import org.apache.sling.api.SlingHttpServletResponse;
8 import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
9
10 @SlingServlet(paths="/bin/company/titleservlet")
11
12 public class TitleSlingServlet extends SlingSafeMethodsServlet {
13     private static final long serialVersionUID = -3968092666512058118L;
14
15     @Override
16     protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException {
17         response.getWriter().print("<h1>Sling Servlet injected this title</h1>");
18     }

```

2. Select training.core project and click *Run As > Maven install*



Eclipse will build the project and install it on the AEM server.

3. Verify you can call the title servlet via: <http://localhost:4502/bin/trainingproject/titleservlet>



4. Change the servlet implementation by changing the code in **TitleSlingServlet.java** (as shown below) and then saving your changes.

- a. Comment out the following line:

```
//@SlingServlet(paths="/bin/trainingproject/title servlet")
```

- b. Uncomment the following line:

```
@SlingServlet(resourceTypes="/apps/trainingproject/components/content/title", extensions="html")
```

```

1. package com.adobe.training.core.servlets;
2.
3. import java.io.IOException;
4. import javax.servlet.ServletException;
5. import org.apache.felix.scr.annotations.sling.SlingServlet;
6. import org.apache.sling.api.SlingHttpServletRequest;
7. import org.apache.sling.api.SlingHttpServletResponse;
8. import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
9.
10. //@SlingServlet(paths="/bin/trainingproject/title servlet")
11. //TitleSlingServlet using resourceTypes and extensions
12. @SlingServlet(resourceTypes="/apps/trainingproject/components/content/title", extensions="html")
13.
14.
15. public class TitleSlingServlet extends SlingSafeMethodsServlet {
16.
17.     private static final long serialVersionUID = 1L;
18.
19.     @Override
20.     protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response)
21.         throws ServletException, IOException {
22.         response.setHeader("Content-Type", "text/html");

```

```
22.     response.getWriter().print("<h1>Sling Servlet injected this title</h1>");
23.     response.getWriter().close();
24. }
25. }
```



NOTE: You can call the servlet as a resource type. Notice how we changed the **@SlingServlet** to use a **resourceType** and extension. Although using the **path** attribute is easy, it is more efficient to use a **resourceType** with a Sling Servlet and allows servlets to be easily injected into pages.

3. Right-click **training.core**, and perform **Run As > Maven install** to build the project.
 - a. Open <http://localhost:4502/content/trainingproject/en.html> and see the output as follows.
4. Notice how the servlet finds all resources on the page using **/apps/trainingproject/components/content/title** and injects the response from the servlet.

The screenshot shows the AEM authoring interface with three instances of the "SLING SERVLET INJECTED THIS TITLE" component. Each instance displays a different title and developer information. The first instance shows "Developer Info: Created by Scott Reynolds. Hobbies include: [Hiking, camping, Biking]. Preferred programming language in AEM is: HTL". The second instance shows "HelloWorldModel says:
Hello World!
This is instance: da565b8a-f210-40db-9579-7446210153de
Resource type is: trainingproject/components/content/helloworld". The third instance shows "Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet." At the bottom of each instance, there is a note: "Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros".

Exercise 2 – Create a Sling Model

Overview

In this exercise, you will create a Sling Model as a Java class and map it into a resource that will consist of a set of properties in a JCR node in the repository. The Sling servlet performs the mapping, which is used to return the values of the properties. In fact, you will use this class Model as a wrapper to format the class fields, and output them in the browser.

Steps

1. Create a Sling Model named **StockModel**.
 - a. In Eclipse, navigate to *training.core > src/main/java*
 - b. Right-click **com.adobe.training.models** and choose *New > Class*
 - Name: **StockModel**
 - Accept all other default settings and click **Finish**
2. Modify the code.
 - a. Double-click **StockModel** to open it in editor.
 - b. Replace the existing code with the contents from Exercise_Files under */Exercise_Files/08_Deep_Dive_Sling/StockModel.java*

```
1. package com.adobe.training.core.models;
2.
3. import javax.inject.Inject;
4. import javax.jcr.Node;
5. import javax.jcr.RepositoryException;
6.
7. import org.apache.sling.api.resource.Resource;
8. import org.apache.sling.models.annotations.Model;
9. import org.apache.sling.models.annotations.injectorspecific.Self;
10.
11. /**
12. * This model represents a Yahoo stock data structure created from the StockDataImporter:
13. * /content
14. * + ADBE [cq:Page]
15. * + lastTrade [nt:unstructured]
16. * - lastTrade = <value>
17. * - requestDate = <value>
18. * - requestTime = <value>
19. * - ..
```

```
20. *
21. * @author Kevin Nennig (nennig@adobe.com)
22. */
23. @Model(adaptables=Resource.class)
24. public class StockModel{
25.
26.     @Inject @Self
27.     private Node stock;
28.
29.     @Inject
30.     private Node lastTrade;
31.
32.     public String getStockSymbol() throws RepositoryException {
33.         return stock.getName();
34.     }
35.
36.     public double getLastTrade() throws Exception{
37.         return Double.parseDouble(lastTrade.getProperty("lastTrade").getString());
38.     }
39.     public String getRequestDate() throws Exception{
40.         return lastTrade.getProperty("requestDate").getString();
41.     }
42. }
43. public String getRequestTime() throws Exception{
44.     return lastTrade.getProperty("requestTime").getString();
45. }
46. }
47. public String getTimestamp() throws Exception{
48.     return getRequestDate() + " " + getRequestTime();
49. }
50. public double getUpDown() throws Exception{
51.     return Double.parseDouble(lastTrade.getProperty("upDown").getString());
52. }
53. public double getOpenPrice() throws Exception{
54.     return Double.parseDouble(lastTrade.getProperty("openPrice").getString());
55. }
56. public double getRangeHigh() throws Exception{
57.     return Double.parseDouble(lastTrade.getProperty("rangeHigh").getString());
58. }
59. public double getRangeLow() throws Exception{
60.     return Double.parseDouble(lastTrade.getProperty("rangeLow").getString());
61. }
62. public int getVolume() throws Exception{
63.     return Integer.parseInt(lastTrade.getProperty("volume").getString());
64. }
65. }
```

- a. Examine the code.

**NOTE:**

StockModel represents a Yahoo stock data structure created from the StockDataImporter. Notice the annotation @Model used to make a Sling Resource adaptable to our model, and the annotations @Inject used to inject resource properties into the data members of our class.

- b. Save the changes.

3. Create a servlet named **SlingModelServlet**.



NOTE: This StockModel will work with the StockDataImporter you will create later in this course. Since this data importer is not created yet, we will use a servlet and dummy data to simulate the input and output for the StockModel.

- a. In the subpackage **com.adobe.training.core.servlets**, create the class **SlingModelServlet** with following code:

```
1. package com.adobe.training.core.servlets;
2.
3. import java.io.IOException;
4.
5. import javax.servlet.ServletException;
6.
7. import org.apache.felix.scr.annotations.sling.SlingServlet;
8. import org.apache.sling.api.SlingHttpServletRequest;
9. import org.apache.sling.api.SlingHttpServletResponse;
10. import org.apache.sling.api.resource.Resource;
11. import org.apache.sling.api.resource.ResourceResolver;
12. import org.apache.sling.api.resource.ValueMap;
13. import org.apache.sling.api.servlets.SlingAllMethodsServlet;
14. import org.slf4j.Logger;
15. import org.slf4j.LoggerFactory;
16.
17. import com.adobe.training.core.models.StockModel;
18.
19. /**
20. * Example URI http://localhost:4502/content/trainingproject/en.model.html/content/ADBE
21. *
22. * To use this servlet, a content structure must be created:
```

```

23. * /content
24. * + ADBE [cq:Page]
25. * + lastTrade [nt:unstructured]
26. * - lastTrade = "100"
27. * - request Date = "11/13/2016"
28. * - requestTime = "4:00pm"
29. *
30. * @author Kevin Nennig (nennig@adobe.com)
31. */
32.
33. @SlingServlet(resourceTypes = "trainingproject/components/structure/page", methods="GET", selectors="model")
34. public class SlingModelServlet extends SlingAllMethodsServlet{
35.     Logger logger = LoggerFactory.getLogger(this.getClass());
36.
37.     private static final long serialVersionUID = 1L;
38.
39.     ResourceResolver resourceResolver;
40.
41.     public void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException{
42.         response.setContentType("text/html");
43.         try {
44.             // Get the resource (a node in the JCR) using ResourceResolver from the request
45.             resourceResolver = request.getResourceResolver();
46.
47.             //Specify the node of interest in the suffix on the request
48.             String nodePath = request.getRequestPathInfo().getSuffix();
49.             if(nodePath != null){
50.                 Resource resource = resourceResolver.getResource(nodePath);
51.
52.                 // Adapt resource properties to variables using ValueMap, and log their values
53.                 Resource parent = resource.getChild("lastTrade");
54.                 ValueMap valueMap=parent.adaptTo(ValueMap.class);
55.                 response.getOutputStream().println("<h3>");
56.                 response.getOutputStream().println("lastTrade node with ValueMap is");
57.                 response.getOutputStream().println("</h3><br />");
58.                 response.getOutputStream().println("(Last Trade ) "+valueMap.get("lastTrade").toString() + " " + valueMap.get("requestTime").toString());
59.
60.                 //Adapt the resource to our model
61.                 StockModel stockModel = resource.adaptTo(StockModel.class);
62.                 response.getOutputStream().println("<br /><h3>");
63.                 response.getOutputStream().println("lastTrade node with StockModel is");
64.                 response.getOutputStream().println("</h3><br />");
```

```

65.         response.getOutputStream().println("(Last Trade) " + stockModel.getLastTrade() +
   " (Requested Time) " + stockModel.getTimestamp());
66.     }
67.     else {
68.         response.getWriter().println("Can't get the last trade node, enter a suffix in the URI"
   );
69.     }
70. } catch (Exception e) {
71.     response.getWriter().println("Can't read last trade node. make sure the suffix path exi
sts!");
72.     logger.error(e.getMessage());
73. }
74. response.getWriter().close();
75. }
76.
77. }
3.

```



NOTE: This class extends the Java class named `org.apache.sling.api.servlets.SlingAllMethodsServlet`, to define an AEM Sling Servlet. We'll use this servlet to inject our resource properties into our model.

In this servlet, we get the node resource using the `ResourceResolver.getResource()` method. `ResourceResolver` is created from the request with the `getResourceResolver()` method. You can see the values of the node properties are adapted to a `ValueMap` to be manipulated within the servlet. This step is optional; it only demonstrates how you can extract each injected field in the servlet. The resource is adapted to our model with the `resource.adaptTo(MyModel.class)` call.

-
- b. Save the changes.

4. Deploy the project into AEM.

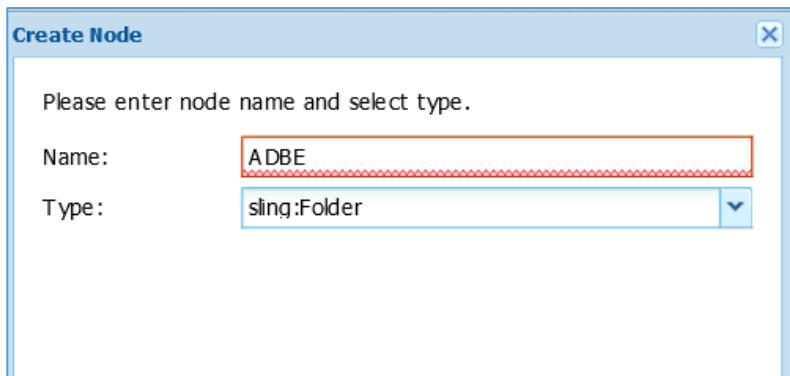
- In Project Explorer, right-click **training.core** and choose **Run AS > Maven install**
- Verify the bundle installed successfully.



NOTE: Along with the servlet, you must some dummy data for the Sling Model to represent. Create a node structure that represents the data set that will be created by the data importer.

5. Within CRXDE Lite, create a node structure that simulates the imported data set.

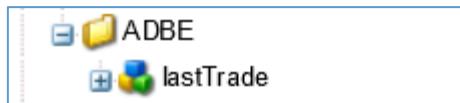
- Within CRXDE lite, create a node structure that matches the path defined by the **nodePath** property.
- Right-click **/content** and choose **Create > Create Node**.
 - Name: **ADBE**
 - Type: **sling:Folder**



- Click **OK** and save the changes

- Create a child node under **ADBE**.

- Name: **lastTrade**
- Type: **nt:unstructured**



d. In the newly created node, create three properties as follows:

- Name: **lastTrade** of type **String** and Value: **100**
- Name: **requestDate** of type **String**,Value: **11/13/2016**
- Name: **requestTime** of type **String**,Value: **4:00 pm**

Properties			Access Control	Replication	Console	Build Info
Name	Type	Value				
1 jcr:primaryType	Name	nt:unstructured				
2 lastTrade	String	100				
3 requestDate	String	11/13/2016				
4 requestTime	String	4:00pm				

e. Click Save All

6. Test the servlet.

- a. Finally, open an Adobe Experience Manager page in your browser whose sling:resourceType matches the one used by our servlet
- b. Insert a selector in it named **model** to execute the servlet
- c. Add a suffix that is the location of the data set we created

For example: <http://localhost:4502/content/trainingproject/en.model.html/content/ADBE>

A message, returned by the model, is displayed in the browser :

lastTrade node with ValueMap is
(Last Trade) 100 (Requested Time) 11/13/2016 4:00pm

lastTrade node with StockModel is
(Last Trade) 100.0 (Requested Time) 11/13/2016 4:00pm

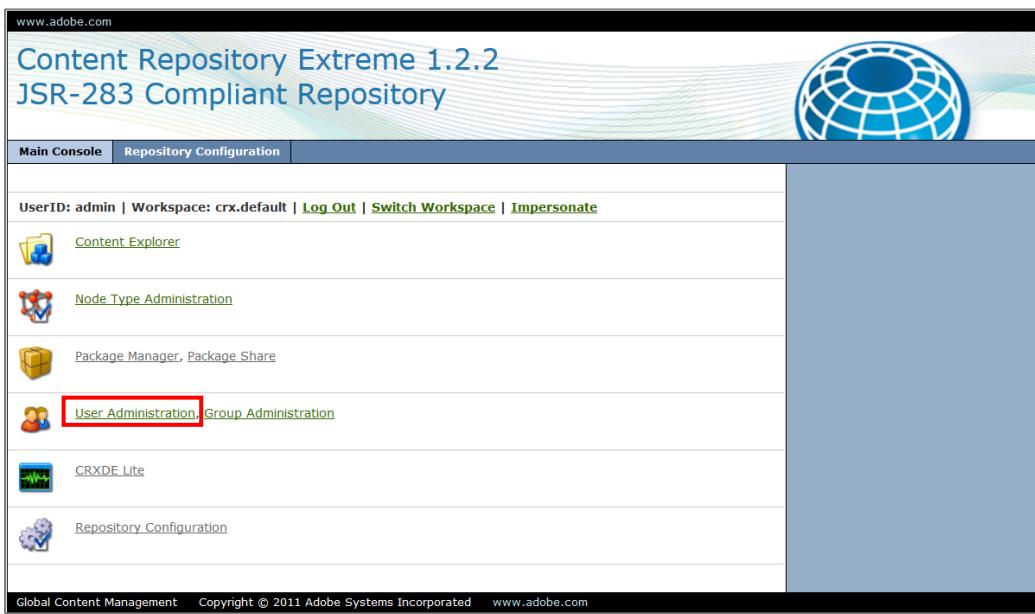
Exercise 3 – Create a System User

Overview

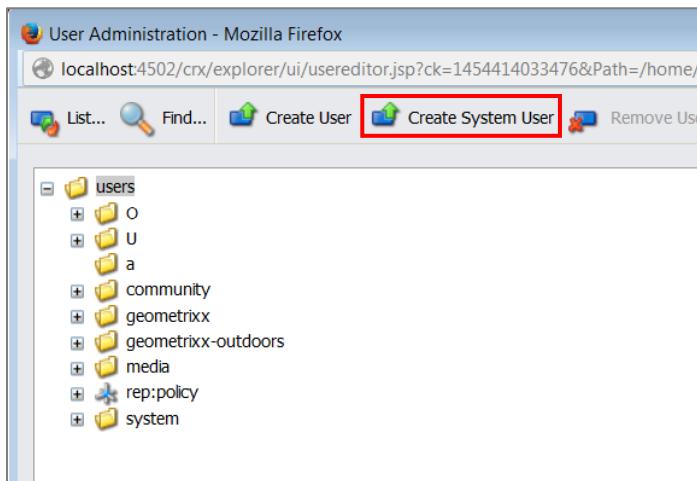
In this exercise, we will create a system user, access rights to the user, and test executing tasks as this user.

Steps

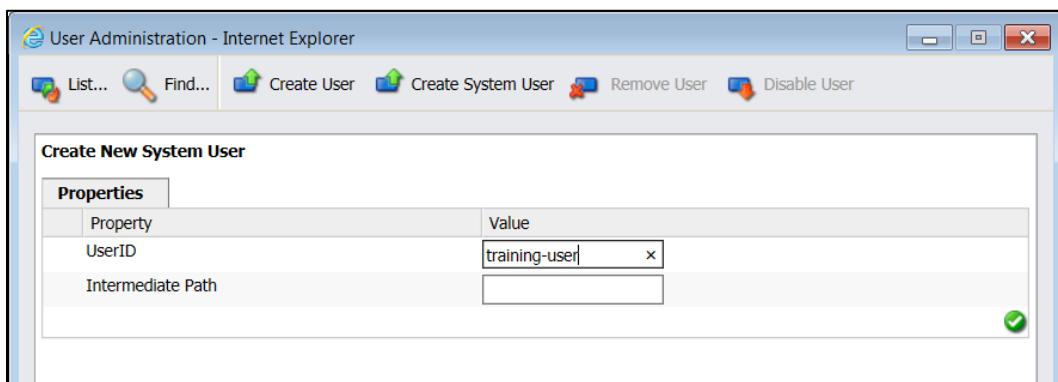
1. Create a service user
 - a. Navigate to: <http://localhost:4502/crx/explorer>
 - b. Login as **admin/admin**
 - c. Click User Administration



- d. In the popup, click **Create System User**.



- e. Enter the UserID: **training-user**



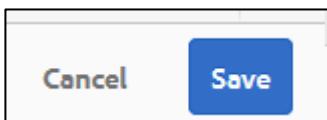
- Click the green checkmark in the lower-right corner of the dialog box.
 - Click **Close**.
2. Navigate to <http://localhost:4502/libs/granite/security/content/useradmin.html>
- a. Scroll-down and find the training-user at the bottom
 - b. Click training-user
 - c. Under Add Users to Groups:
 - d. Type admin and select the administrators group from the list

Edit User Settings

ID *	training-user	Email	
First Name		Last Name	
Phone Number		Job Title	
Street		Mobile	
City		Postal Code	
Country	Select	State	
Title		Gender	Select
About			

Account settings	Photo
Status Change Password Create KeyStore Create TrustStore	 <input type="button" value="Upload Image"/> <small>Accepted file types: .jpg, .png, .tif, and .gif. Preferred size 240x240 px.</small>
Add User to Groups	
<input type="button" value="Select group"/> <input checked="" type="button" value="X administrators"/>	

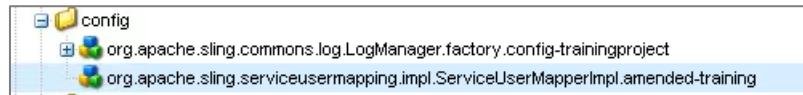
- Click the **Save** button on the top right corner



NOTE: For training purposes, you are given full rights access to the training-user. Typically, when creating a service user, they should only have permissions associated with the tasks it will be executing.

3. Configure the Service User Mapping service:

- Navigate to CRXDE Lite: <http://localhost:4502/crx/de>
- Under /apps/trainingproject/config, create a node
 - Name: org.apache.sling.serviceusermapping.impl.ServiceUserMapperImpl.amended-training
 - Type: sling:OsgiConfig



- Click OK and add the following two properties to the node:
 - **Name=service.ranking, type=Long, value=0**
 - **Name=user.mapping, type=String, value=com.adobe.training.core:training=training-user**
- The user.mapping allows us to map a bundle with a subservice name to a service user as shown in the screenshot.
 - Bundle: **com.adobe.training.core**
 - Subservice name: **training**
 - Service User: **training-user**

Properties		Access Control	Replication	Consistency
Name	Type	Value		
1 jcr:created	Date	2016-01-27T13:21:48.316+05:30		
2 jcr:createdBy	String	admin		
3 jcr:primaryType	Name	sling:OsgiConfig		
4 service.ranking	Long	0		
5 user.mapping	String	com.adobe.training.core:training=training		

- Save your changes.



NOTE: If different permissions are needed for different components in a bundle for security reasons, you must create new config nodes with new Subservices and different Service Users.

- Back in Eclipse, select training.ui.apps, right-click and select AEM > Import from server.
- Accept the default and click **Finish**.

Exercise 4 – Create a Job Consumer for a Topic

Overview

In this exercise, you will create a job consumer for a topic.

Steps

1. Create a class named **ReplicationLogger**.
 - a. In Eclipse, navigate to *training.core > src/main/java* and right-click on **com.adobe.training.core** and select *New > Class*
 - b. Create a class named ReplicationLogger
 - c. Paste the following code (from the Exercise_Files):



NOTE: The code is provided as part of the Exercise_Files under */Exercise_Files/08_Deep_Dive_Sling/*. Copy and paste the code from the exercise files to Eclipse. Do not copy from this exercise book. The following is for illustration purposes only.

```
1. package com.adobe.training.core;
2.
3. import org.apache.felix.scr.annotations.Component;
4. import org.apache.felix.scr.annotations.Property;
5. import org.apache.felix.scr.annotations.Reference;
6. import org.apache.felix.scr.annotations.Service;
7. import org.apache.sling.api.resource.LoginException;
8. import org.apache.sling.api.resource.ResourceResolver;
9. import org.apache.sling.api.resource.ResourceResolverFactory;
10. import org.apache.sling.event.jobs.Job;
11.
12. import org.apache.sling.event.jobs.consumer.JobConsumer;
13.
14. import org.slf4j.Logger;
15. import org.slf4j.LoggerFactory;
16. import com.day.cq.wcm.api.Page;
17. import com.day.cq.wcm.api.PageManager;
18.
19. import java.util.HashMap;
20. import java.util.Map;
```

```

21.
22. @Component(immediate = true)
23. @Service(value={JobConsumer.class})
24. @Property(name=JobConsumer.PROPERTY_TOPICS, value = "com/adobe/training/core/replicatio
   njob")
25.
26. public class ReplicationLogger implements JobConsumer {
27.     private final Logger logger = LoggerFactory.getLogger(getClass());
28.
29.     @Reference
30.     private ResourceResolverFactory resourceResolverFactory;
31.
32.     @Override
33.     public JobResult process(final Job job) {
34.
35.         final String pagePath = job.getProperty("PAGE_PATH").toString();
36.
37.         ResourceResolver resourceResolver = null;
38.         try {
39.             Map<String, Object> serviceParams = new HashMap<String, Object>();
40.             serviceParams.put(ResourceResolverFactory.SUBSERVICE, "training");
41.             resourceResolver = resourceResolverFactory.getServiceResourceResolver(serviceParams);
42.         } catch (LoginException e) {
43.             e.printStackTrace();
44.         }
45.
46.         // Create a Page object to log its title
47.         final PageManager pm = resourceResolver.adaptTo(PageManager.class);
48.         final Page page = pm.getContainingPage(pagePath);
49.         if(page != null) {
50.             logger.info("++++++ ACTIVATION OF PAGE : {}", page.getTitle());
51.         }
52.         return JobConsumerJobResult.OK;
53.     }
54. }
```

- d. Save all your files.
2. Perform *Run As > Maven install* on training.core.

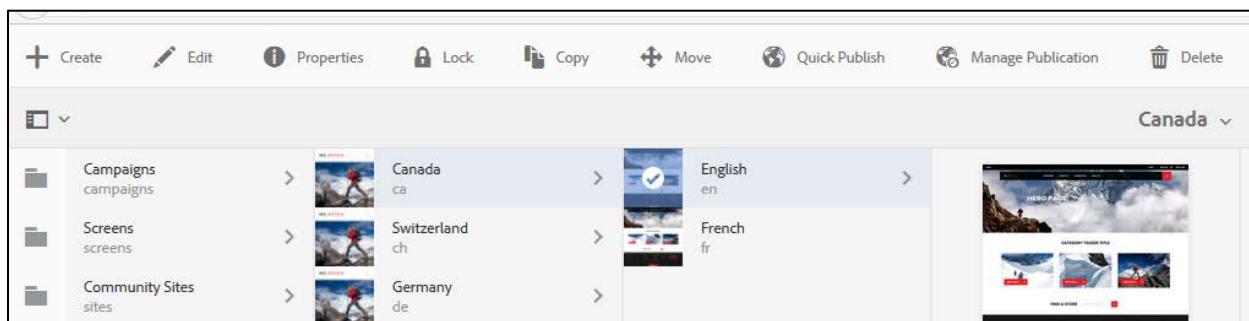
3. Notice the **ReplicationAction.EVENT_TOPIC** property, which is being listened for, and the **@Component(immediate = true)** statement, which is needed to ensure the component **com.adobe.training.core.ReplicationLogger** is active immediately.

You can verify that the component is available in Adobe Experience Manager at:

<http://localhost:4502/system/console/components> and search for the component by its name:

com.adobe.training.core.ReplicationLogger	
Bundle	com.adobe.training.core (471)
Implementation Class	com.adobe.training.core.ReplicationLogger
Default State	enabled
Activation	immediate
Configuration Policy	optional
Service Type	singleton
Services	org.apache.sling.event.jobs.consumer.JobConsumer
Reference resourceResolverFactory	[{"Satisfied","Service Name: org.apache.sling.api.resource.ResourceResolverFactory","Cardinality: 1..1","Policy: Service ID 894 (Apache Sling Resource Resolver Factory)"]
Properties	component.id = 2693 component.name = com.adobe.training.core.ReplicationLogger job.topics = com/adobe/training/core/replicationjob service.pid = com.adobe.training.core.ReplicationLogger service.vendor = Adobe

4. To activate (publish) a page, select the page, and select **Quick Publish**.



5. Activate a page in the Site Admin.

- Go to your desktop and inspect the **/crx-quickstart/logs/project-trainingproject.log** file on your C:/ drive.

- b. You should see logged messages giving the titles of pages as they are activated.

```
[com.adobe.training.core.servlets.TitleSlingServlet,6927, [javax.servlet.Servlet]] ServiceEvent REGISTERED
27.03.2017 10:08:53.277 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service
[com.adobe.training.core.ReplicationLogger,6929, [org.apache.sling.event.jobs.consumer.JobConsumer]]
ServiceEvent REGISTERED
27.03.2017 10:08:53.282 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service
[com.adobe.training.core.schedulers.SimpleScheduledTask,6937, [java.lang.Runnable]] ServiceEvent REGISTERED
27.03.2017 10:08:53.285 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service
[com.adobe.training.core.servlets.SlingModelServlet,6938, [javax.servlet.Servlet]] ServiceEvent REGISTERED
27.03.2017 10:08:53.321 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service
[com.adobe.training.core.listeners.ReplicationListener,6940, [org.osgi.service.event.EventHandler]]
ServiceEvent REGISTERED
27.03.2017 10:08:53.323 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service
[com.adobe.training.core.listeners.SimpleResourceListener,6941, [org.osgi.service.event.EventHandler]]
ServiceEvent REGISTERED
27.03.2017 10:08:53.944 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent STARTED
27.03.2017 10:09:03.503 *INFO*
[sling-threadpool-6744ac64-4be0-47e1-9b21-8a58c9d24563-(apache-sling-job-thread-pool)-19-<main
queue> (com/adobe/training/core/replicationjob)] com.adobe.training.core.ReplicationLogger ++++++
ACTIVATION OF PAGE : English
```

The Sling Jobs page in the Felix console shows statistics about our job, that can be found by topic:

<http://localhost:4502/system/console/slingevent>

- c. Search for replicationjob:

Topic Statistics: com/adobe/training/core/replicationjob	
Last Activated	10:09:03:532 2017-Mar-27
Last Finished	10:09:03:544 2017-Mar-27
Finished Jobs	2
Failed Jobs	0
Cancelled Jobs	0
Processed Jobs	2
Average Processing Time	4 ms
Average Waiting Time	4054 min 25 secs

Exercise 5 – Clean up Nodes with a Sling Scheduler

Overview

In this exercise, you will write a job that periodically deletes temporary nodes in the repository. The configuration will be stored in a configuration node, which you can see and edit in the Web Console.

Steps

1. Schedule a job.
 - a. Open Eclipse and write a new class named **CleanupScheduledTask** under *training.core > src/main/java > com.adobe.training.core.schedulers* package.
 - b. Write the following code in **CleanupScheduledTask**



NOTE: The code is provided as part of the Exercise_Files under /Exercise_Files/08_Deep_Dive_Sling/. Copy and paste the code from the exercise files to Eclipse. Do not copy from this exercise book. The following is for illustration purposes only.

```
2. package com.adobe.training.core.schedulers;
3.
4.
5. import java.util.Dictionary;
6. import javax.jcr.RepositoryException;
7. import javax.jcr.Session;
8. import org.apache.felix.scr.annotations.Component;
9. import org.apache.felix.scr.annotations.Property;
10. import org.apache.felix.scr.annotations.Reference;
11. import org.apache.felix.scr.annotations.Service;
12. import org.apache.sling.commons.osgi.PropertiesUtil;
13. import org.apache.sling.jcr.api.SlingRepository;
14. import org.osgi.service.component.ComponentContext;
15. import org.slf4j.Logger;
16. import org.slf4j.LoggerFactory;
17.
18. @Component(immediate = true, metatype = true, label = "Training Cleanup Service")
19. @Service(value = Runnable.class)
20. @Property(name = "scheduler.expression", value = "*/20 * * * ?") // Every 20
```

```
21.                                     // seconds
22. public class CleanupScheduledTask implements Runnable {
23.     private final Logger logger = LoggerFactory.getLogger(getClass());
24.
25.     @Reference
26.     private SlingRepository repository;
27.     @Property(label = "Path", description = "Delete this path", value = "/mypathtraining")
28.     public static final String CLEANUP_PATH = "cleanupPath";
29.     private String cleanupPath;
30.
31.     protected void activate(ComponentContext componentContext) {
32.         configure(componentContext.getProperties());
33.     }
34.
35.     protected void configure(Dictionary<?, ?> properties) {
36.         this.cleanupPath = PropertiesUtil.toString(properties.get(CLEANUP_PATH), null);
37.         logger.info("!!!configure: cleanupPath='{}'", this.cleanupPath);
38.     }
39.
40.     @Override
41.     public void run() {
42.         logger.info("!!!running now");
43.         Session session = null;
44.         try {
45.             session = repository.loginService("training", null);
46.             if (session.itemExists(cleanupPath)) {
47.                 session.removeItem(cleanupPath);
48.                 logger.info("!!!node deleted");
49.                 session.save();
50.             }
51.         } catch (RepositoryException e) {
52.             logger.error("!!!exception during cleanup", e);
53.         } finally {
54.             if (session != null) {
55.                 session.logout();
56.             }
57.         }
58.     }
59. }
```

NOTE:

- metatype = true enables configuration in the Web Console.
 - Configure will be called when the configuration changes. This is used to obtain the new value for the cleanup path when it is changed.
-

2. Build the project by selecting the **training.core** project and selecting *Run As > Maven install*
3. Refresh your browser (reload the page) where you are working with your Adobe Experience Manager instance (<http://localhost:4502>).
4. Inspect the log output (project-trainingproject.log):

```
25.11.2015 10:54:31.205 *INFO* [qtp1977343400-303] org.apache.sling.auth.core.impl.SlingAuthenticator get
25.11.2015 10:54:31.209 *INFO* [qtp1977343400-281] org.apache.sling.auth.core.impl.SlingAuthenticator get
25.11.2015 10:54:35.000 *INFO* [pool-7-thread-1] com.adobe.training.core.CleanupServiceImpl running now
25.11.2015 10:54:40.000 *INFO* [pool-7-thread-2] com.adobe.training.core.CleanupServiceImpl running now
25.11.2015 10:54:45.000 *INFO* [pool-7-thread-2] com.adobe.training.core.CleanupServiceImpl running now
25.11.2015 10:54:49.485 *INFO* [pool-6-thread-13] com.adobe.training.core.TitlePropertyListener *****

```

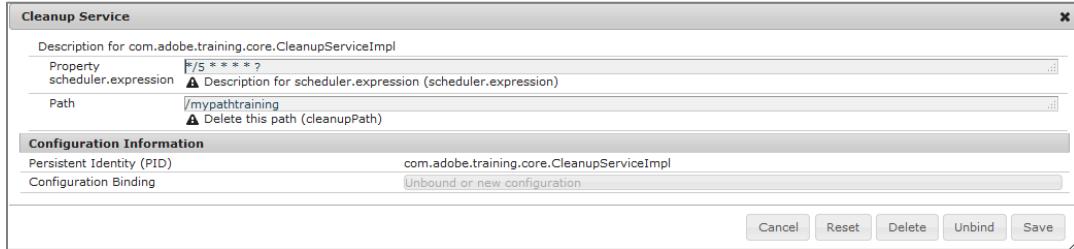
5. Now you know the scheduler is running. However, you need to test to see if it will delete, or cleanup **/mypathtraining** nodes.
 - a. Save the changes, wait 20 seconds, refresh CRXDE Lite, and then see that is is deleted by the cleanup service.
 - b. You can verify this in the custom log file project-trainingproject.log

```
25.11.2015 11:21:20.012 *INFO* [oak-repository-executor-1] com.adobe.granite.repository.Service [4554, {org.apache.jackrabbit.oak.spi.jmx.SessionMBean}] ServiceEvent REGISTERED
25.11.2015 11:21:23.885 *INFO* [pool-6-thread-11] org.apache.jackrabbit.oak.plugins.index.lucene.IndexCopier Error occurred while deleting following files from the local index directory [MMapDir]
25.11.2015 11:21:23.000 *INFO* [pool-7-thread-4] com.adobe.training.core.CleanupServiceImpl running now
25.11.2015 11:21:23.000 *INFO* [pool-7-thread-4] com.adobe.training.core.CleanupServiceImpl running now
25.11.2015 11:21:23.001 *INFO* [pool-7-thread-4] com.adobe.training.core.CleanupServiceImpl task SimpleScheduledTask is now running. myParameter="null"
25.11.2015 11:21:30.000 *INFO* [pool-7-thread-3] com.adobe.training.core.CleanupServiceImpl running now
25.11.2015 11:21:42.912 *INFO* [pool-6-thread-19] com.adobe.training.core.TitlePropertyListener *****
25.11.2015 11:21:42.912 *INFO* [pool-6-thread-19] com.adobe.training.core.TitlePropertyListener *****new property event: /mypathtraining/crpPrimaryType
25.11.2015 11:21:42.917 *INFO* [pool-6-thread-19] com.adobe.training.core.TitlePropertyListener *****new property event: /mypathtraining/jcr:createdBy
25.11.2015 11:21:42.917 *INFO* [pool-6-thread-19] com.adobe.training.core.TitlePropertyListener *****new property event: /mypathtraining/jcr:created
25.11.2015 11:21:42.919 *DEBUG* [pool-6-thread-18] com.adobe.training.core.listeners.SimpleResourceListener Resource event: org/apache/sling/api/resource/Resource@ADDDED at: /mypathtraining
25.11.2015 11:21:44.692 *INFO* [pool-6-thread-17] org.apache.jackrabbit.oak.plugins.index.lucene.IndexCopier Error occurred while deleting following files from the local index directory [MMapDir]
25.11.2015 11:21:45.000 *INFO* [pool-7-thread-1] com.adobe.training.core.CleanupServiceImpl task SimpleScheduledTask is now deleted
25.11.2015 11:21:45.003 *INFO* [pool-7-thread-1] com.adobe.training.core.CleanupServiceImpl node deleted
25.11.2015 11:21:46.528 *DEBUG* [0|0|0|0:0|0|0|1] [14484307085271 GET /like/oq/dict-en.json HTTP/1.1] com.adobe.training.core.filter.LoggingFilter request for /like/oq/dict-en.json, with session 14484307085271, client IP 127.0.0.1, user agent Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2485.149 Safari/537.36
25.11.2015 11:21:49.308 *INFO* [pool-6-thread-21] com.adobe.training.core.listeners.SimpleResourceListener Resource event: org/apache/sling/api/resource/Resource@CHANGED at: /xyz/discovery/img1
25.11.2015 11:21:49.630 *INFO* [pool-7-thread-1] com.day.cq.replication.Agent.publish reverse sending GET request to http://localhost:4503/bin/receive?slng=authRequestLogin=1

```

<pre>3751 + com.adobe.training.core.CleanupServiceImpl Bundle com.adobe.training.core (448) Implementation Class com.adobe.training.core.CleanupServiceImpl Default State active Activation immediate Configuration Policy optional Service Type singleton Services java.lang.Runnable Reference repository Properties ["BundleId","Service Name: org.apache.sling.jcr.api.SlingRepository","Cardinality: 1..1","Policy: static","Policy Option: reluctant","Bound Service ID 345 (com.adobe.granite.repository.impl.SlingRepositoryManager)"] component.id = 3751 component.name = com.adobe.training.core.CleanupServiceImpl component.version = 1.0.0 service.pid = com.adobe.training.core.CleanupServiceImpl service.vendor = Adobe </pre>	active 
---	--

6. In the Web Console, go to *OSGi > Configuration* (<http://localhost:4502/system/console/configMgr>) and search for: **CleanupScheduledTask**
- Configure a different path (for example, `/mynewpathtraining`), and click **Save**



- Now, in CRXDE Lite, create a node with the new path name.
 - Refresh the browser and confirm it is deleted.
7. Change job scheduler configuration settings using repository.
- It is assumed the project (trainingproject) you created in Eclipse earlier is synched with the Adobe Experience Manager server.
8. Expand the **trainingproject** folder.
9. In the **config.author** folder, create a node of type **sling:OsgiConfig** with the name **com.adobe.training.core.schedulers.CleanupScheduledTask**, with the following properties:
- **Name:** cleanupPath, **Type:** String, **Value:** /mynewpath
 - **Name:** scheduler.expression, **Type:** String, **Value:** */5 * * * * ?

Name	Type	Value	Protected	Mandatory	Multiple	Auto Created
cleanupPath	String	/mynewpath	false	false	false	false
jcr:primaryType	Name	sling:OsgiConfig	true	true	false	true
scheduler.expression	String	*/5 * * * * ?	false	false	false	false

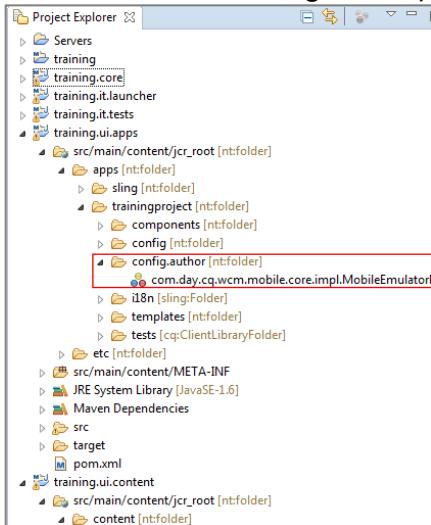
➤ Click Save All

10. Open the web console and confirm the updated configuration on:

<http://localhost:4502/system/console/configMgr/com.adobe.training.core.schedulers.CleanupScheduledTask>

Adobe Experience Manager Web Console	
Components	
Id	# Name
2280	+ com.adobe.training.core.CleanupServiceImpl Binding: com.adobe.training.core (448) Implementation Class: com.adobe.training.core.CleanupServiceImpl Default State: enabled Activation: immediate Configuration Policy: optional Service Type: singleton Services: Reference repository: com.adobe.granite.repository.impl.BingRepositoryManager Properties:
	clearpath = "/myviewpath" component.id = "2280" component.pid = "com.adobe.training.core.CleanupServiceImpl" scheduler.expression = "#715 * * * * ?" service.pid = "com.adobe.training.core.CleanupServiceImpl" service.vendor = "Adobe"

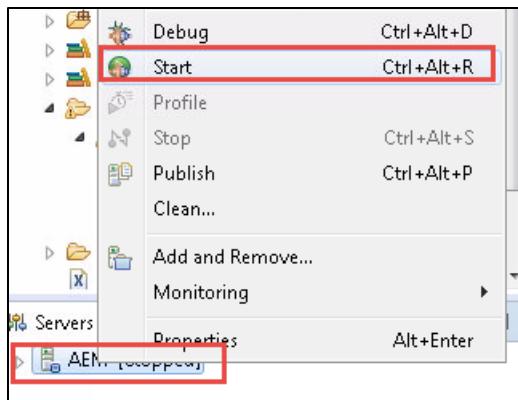
- To sync the **config.author** configuration to your workspace, you need to import the settings from the server. Before checking out, Eclipse looks as follows:



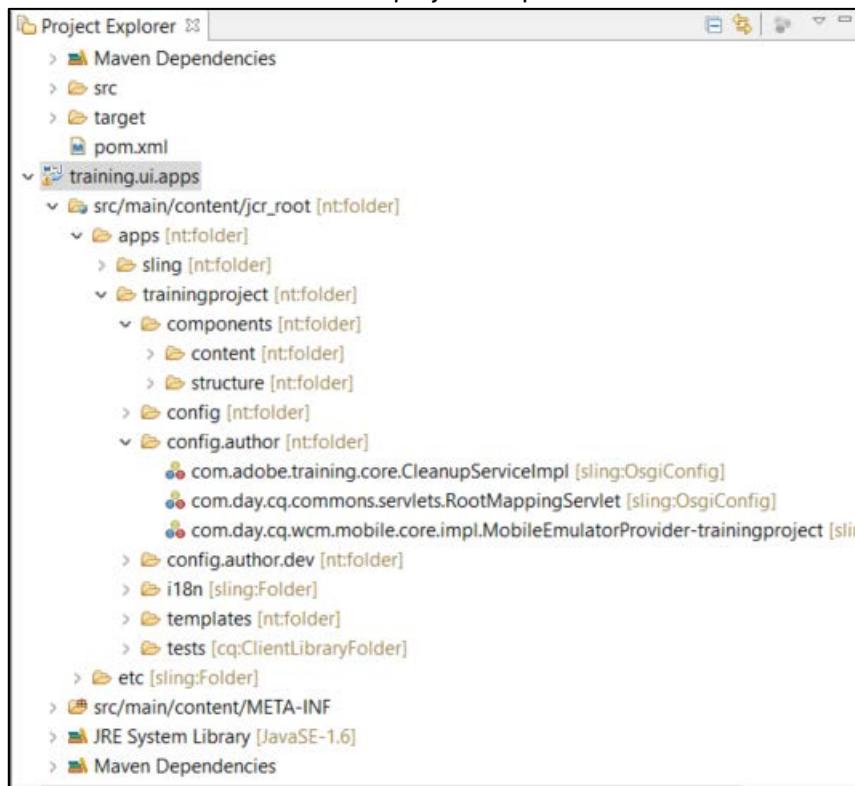
- a. In Eclipse, right-click training.ui.apps, select AEM > Import from server...



NOTE: You may need to ensure your server in Eclipse has started. If it is not, start it.



- b. Click **Finish**. You will see that the project is updated with the CRXDE changes done above.



Review:

In this lab, you did the following:

- Worked with servlets
- Created a System User
- A servlet that displays the title and the JCR properties of the page.
- A class using Sling Models that uses the injected values from another resource.

9 JCR DEEP DIVE

Overview

This module deep dives into the Java Content Repository (JCR) model, with emphasis on David's model. David's model is a methodology for modeling data in a content repository. It is based on the principle that developers have been modeling data for a long time, but have not been modeling data in a content repository space for as long. You will learn how data is stored in Adobe Experience Manager, as well as the structure of the JCR.

Objectives

By the end of this module, you will be able to:

- Describe the JCR Model
- Define the Structure of JCR
- Understand JCR Observation
- Examine Oak indexing and Queries

JCR Model

The JCR model is represented as a tree of nodes and properties. Nodes are addressed as a path similar to the file system and used to organize the content in hierarchical form; properties store the actual data, either as simple types (such as string, Boolean, etc.) or as binary streams for storing files of arbitrary size.

How is JCR different from a relational database (RDB)?

A JCR repository:

- is hierarchical—you can organize your content according to your requirements, with related information stored together. This helps achieve better navigability.
- is flexible—the content can adapt and evolve to be either schema-less or completely restrictive.
- uses a standard Java API.
- abstracts where the information is really stored.
- supports queries and full-text search.

JCR Features:

- Query via SQL, JQOM, and XPath
- Export/import (XML)
- Referential integrity
- Authentication
- Access control
- Versioning
- Observation
- Locking and transactions (JTA)

Understanding David's Model

David's model has the following set of guidelines on how to model content in a repository.

- Data First. Structure Later. Maybe.
- Drive the content hierarchy, don't let it happen.
- Workspaces are for clone(), merge() and update().
- Beware of Same Name Siblings.
- References considered harmful.
- Files are Files are Files.
- IDs are evil.

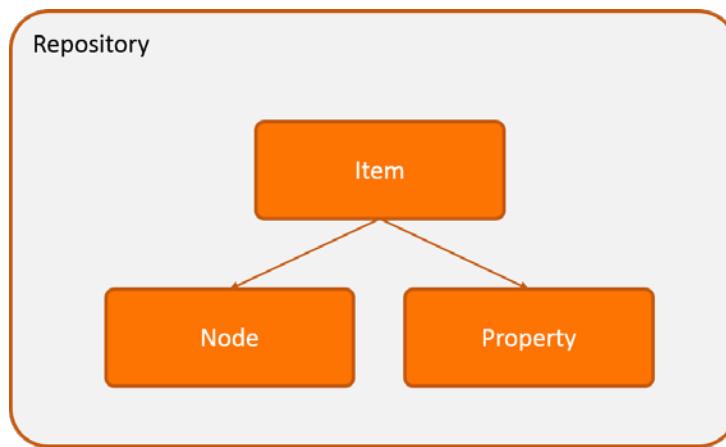
Content Services of JCR

JCR provides the following services, which makes Adobe Experience Manager a robust environment.

- Author-based versioning
- Full-text searching
- Fine-grained access control
- Content categorization
- Content event monitoring

Structure of the JCR

- A JCR repository is comprised of items.
- An item is either a node or a property. A node can have zero or more child items. A property cannot have child items but can hold zero or more values.
- The nodes form the structure of the stored data while the actual content is stored in the values of the properties.
- An additional advantage of the JCR is the 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).



The data in a JCR consists of a tree of nodes with associated properties.

Nodes

- Nodes may optionally have one or more types associated with them, which dictate the kinds of properties, number and type of child nodes, and certain behavioral characteristics of the nodes.
- Nodes may point to other nodes via a special reference type property.
- Nodes may also be of the version-able type. This makes the repository track a document's history and store copies of each version of the document.

- Properties
- Properties are either single or multi-valued.
- Each value has one of the 12 possible types. These types include familiar data storage types such as strings, numbers, Booleans, binaries and dates, as well as types that hold pointers to other nodes.

Node Types

Node types are used to enforce structural restrictions on the nodes and properties in a workspace by defining for each node its required and permitted child nodes and properties.

- Primary node types are typically used to define the core characteristics of a node (indicated by the property `jcr:primaryType`).
- Mixin node types are used to add additional characteristics often related to specific repository functions or to metadata.

Node Type Definitions

- Node types are stored in the form of node type definitions.
- They 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 Inheritance

A node may have one or more super types. Seen from the super type, the inheriting node type is a subtype. Super types are discoverable using the JCR API.

A subtype inherits:

- property definitions
- child node definitions
- other attributes, such as 'isMixin'

Event Modelling

A persisted change to a workspace is represented by a set of one or more events. Each event reports a single change to the structure of the persistent workspace in terms of an item added, changed, moved or removed.

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 sub-item in the structure.

The Event Object

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

Types of Events

There are various types of events. The type of event is identified and retrieved through the method `int Event.getType()`.

Types of events are:

- Node added
- Node moved
- Node removed
- Property added
- Property removed
- Property changed
- Persist

Event Information

Each event is associated with the following information:

- Event path – retrieved through `String Event.getPath()`
- Identifier – retrieved through `StringEvent.getIdentifier()`
- Information map – retrieved through `java.util.Map Event.getInfo()`

If the event is `NODE_ADDED` or `NODE_REMOVED`:

- `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()` returns an empty Map object.

If the event is `NODE_MOVED`:

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

If the event is `PROPERTY_ADDED`, `PROPERTY_CHANGED`, or `PROPERTY_REMOVED`:

- `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()` returns an empty Map object.

If the event is `PERSIST`,

- `Event.getPath()` returns null.
- `Event.getIdentifier()` returns null.
- `Event.getInfo()` returns an empty Map.

Asynchronous Observation

An application connects with the asynchronous observation mechanism by registering an event listener with the workspace. 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. In this type of observation, execution continues normally on the thread that performed the operation.

Understanding Asynchronous Observation

The following sections help you understand the functionalities of asynchronous observations.

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 the following code:

```
1. void ObservationManager.addEventListener(EventListener listener,
2. int eventTypes,
3. String absPath,
4. boolean isDeep,
5. String[] uuid,
6. String[] nodeTypeName,
7. 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.

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 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: `EventIterator.nextEvent()`.

Listing Event Listeners

A set of `EventListener` objects are retrieved using the `EventListenerIterator`.

The `EventListenerIterator` class inherits the methods of `RangeIterator` and adds an `EventListener`-specific next method:

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

Removing Event Listeners

You can remove the event listener using:

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

User Data

You can set the user data using:

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



Perform Task –[Exercise 1 – Create an Observation Listener](#)**from Lab H.**

Query Index

Oak does not index content by default as Jackrabbit 2 does. You must create custom indexes when necessary, much like in traditional RDBs. As a finished product, AEM ships with the most common indexers for Apache Oak to make indexing an easier task. If there is no index for a specific query, the repository will be traversed. That is, the query will still work but will probably be very slow.

If Oak encounters a query without an index, a WARN level log message displays:

```
*WARN* Traversed 1000 nodes with filter Filter(query=select...) consider creating an index or changing the query
```

If this is the case, you might need to create an index, or you might need to change the condition of the query to take advantage of an existing index.

If a query reads more than 10000 nodes in memory, then the query is cancelled with an `UnsupportedOperationException` saying that, “The query read more than 10000 nodes in memory. To avoid running out of memory, processing was stopped.” As a workaround, you can change this limit using the system property “`oak.queryLimitInMemory`.”

Query Indices are defined under the `oak:index` node.

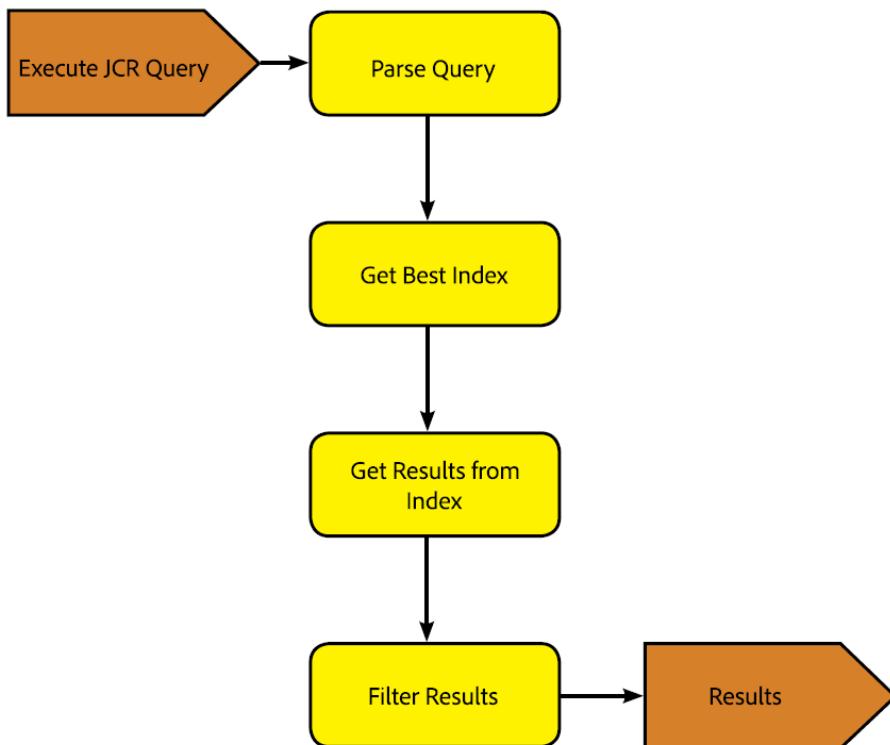
Supported Query Languages

The Oak query engine supports the following languages:

- XPath
- SQL
- SQL-2
- JQOM

Oak Query Implementation Overview

The new Apache Oak based backend allows different indexers to be plugged into the repository. The standard indexer is the Property Index, for which the index definition is stored in the repository itself. External full text indexers can also be used with Adobe Experience Manager. Implementations for Apache Lucene and Solr are available by default. The Traversal Index indexer is the one used if no other indexer is available. This means the content is not indexed and all content nodes are traversed to find matches to the query. If multiple indexers are available for a query, each available indexer estimates the cost of executing the query. Oak then chooses the indexer with the lowest estimated cost.



The above diagram is a high-level representation of the query execution mechanism of Apache Oak.

First, the query is parsed into an Abstract Syntax Tree then the query is checked and transformed into SQL-2, which is the native language for Oak queries.

After the query is checked and transformed into SQL-2, each index is consulted to estimate the cost for the query. Once completed, the results from the most cost-effective index are retrieved. Finally, the results are filtered, both to ensure the current user has read access to the result and the result matches the complete query.

- Query Processing on Cost Calculations

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

By default, the following indexes are available:

- Property index for each indexed property
- Full-text index, which is based on Apache Lucene/Solr
- Node type index, which is based on an property index for the properties `jcr:primaryType` and `jcr:mixins`
- Traversal index that iterates over a subtree

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

Native Queries

To take advantage of features available in full-text index implementations such as Apache Lucene and Apache Lucene Solr, so-called native constraints are supported. These constraints are passed directly to the full-text index. This is supported for both XPath and SQL-2. For XPath queries, the name of the function is `rep:native`, and for SQL-2, it is `native`.

The first parameter is the index type and currently supported parameters are Solr and Lucene. The second parameter is the native search query expression.

For SQL-2, the selector name (if needed) is the first parameter, just before the language. If full-text implementation is not available, the queries will fail.

Configuring the Indexes

Indexes are configured as nodes in the repository under the `oak:index` node. The type of the index node must be `oak:QueryIndexDefinition`. Several configuration options are available for each indexer as node properties. For more information, see the following configuration details for Property Index, Lucene Index, and Solr Index.

The Property Index

The Property Index is generally useful for queries with property constraints but are not full-text. You can configure it using the following procedure:

Open CRXDE Lite by going to <http://localhost:4502/crx/de/index.jsp>

Create a new node under `oak:index`.

Name the node `PropertyIndex`, and set the node type to `oak:QueryIndexDefinition`.

Click **OK** then set the following properties for the new node:

Name	Type	Value
<code>type</code>	String	<code>property</code>
<code>propertyNames</code>	Name	<code>jcr:uuid</code>

Click **Save All** to save the changes – the `PropertyIndex` node will have three properties on its tab now.

This particular example will index the `jcr:uuid` property, whose job is to expose the Universally Unique Identifier (UUID) of the node it is attached to.

The Lucene Full-text Index

A full-text indexer based on Apache Lucene is available in Adobe Experience Manager. If a full-text index is configured, then all queries with a full-text condition use the full-text index, no matter if there are other conditions that are indexed, and no matter if there is a path restriction.

If no full-text index is configured, then queries with full-text conditions may not work as expected. The query engine has a basic verification in place for full-text conditions, but it does not support all features that Lucene does and it will traverse all nodes if there are no indexed constraints.

As the index is updated through an asynchronous background thread, some full-text searches will be unavailable for a small window of time until the background processes are finished.

You can configure a Lucene full-text index using the following procedure:

Open CRXDE Lite and create a new node under `oak:index`.

Name the node `LuceneIndex` and set the node type to `oak:QueryIndexDefinition`.

Add the following properties to the node:

Name	Type	Value
<code>type</code>	String	<code>lucene</code>

async	String	async
-------	--------	-------

Save the changes.

The Solr Index

You can use the Solr index for full-text search, as well as search by path, property restrictions, and primary type restrictions.

- It can be used for any type of JCR query.
- Its integration with Adobe Experience Manager occurs at the repository level.
- It can be configured to work as an embedded server with the Adobe Experience Manager instance, or as a remote server.

To configure Adobe Experience Manager with an embedded Solr server:

1. Navigate to the Web Console at: <http://localhost:4502/system/console/configMgr>
2. Search for Oak Solr server provider.
3. Click the **Edit** icon, and in the resulting dialog box, select the server type as **Embedded Solr** from the drop-down list.
4. Click **Save**.
5. Search for Oak Solr embedded server configuration.
6. Click the **Edit** icon, and update the configuration. The Solr home directory configuration will look for a folder with the same name in the Adobe Experience Manager installation folder.
7. Open CRXDE Lite, and log in as **Admin**.
8. Under **oak:index**, create a node called **solrIndex**, of type **oak:QueryIndexDefinition** with the following three properties (enter the information for each field in each line below by clicking **Add** to create a new row on the Properties tab for the corresponding lines and values below):
 - Name: **type**, Type: String, Value: **solr**
 - Name: **async**, Type: String, Value: **async**
 - Name: **reindex**, Type: Boolean, Value: **true**
9. Click **Save All** in the upper-left corner.

Temporarily Disabling an Index

To temporarily disable an index (for example, for testing), set the index type to disabled. While the index type is not set, the index is not updated, so if you enable it again, it might not be correct. This is especially important for synchronous indexes.

Indexing Tools

The Adobe Consulting Service Commons toolkit introduces a set of indexing tools in an effort to simplify management and maintenance of indexes in the underlining data storage layer of Adobe Experience Manager, Apache Oak.

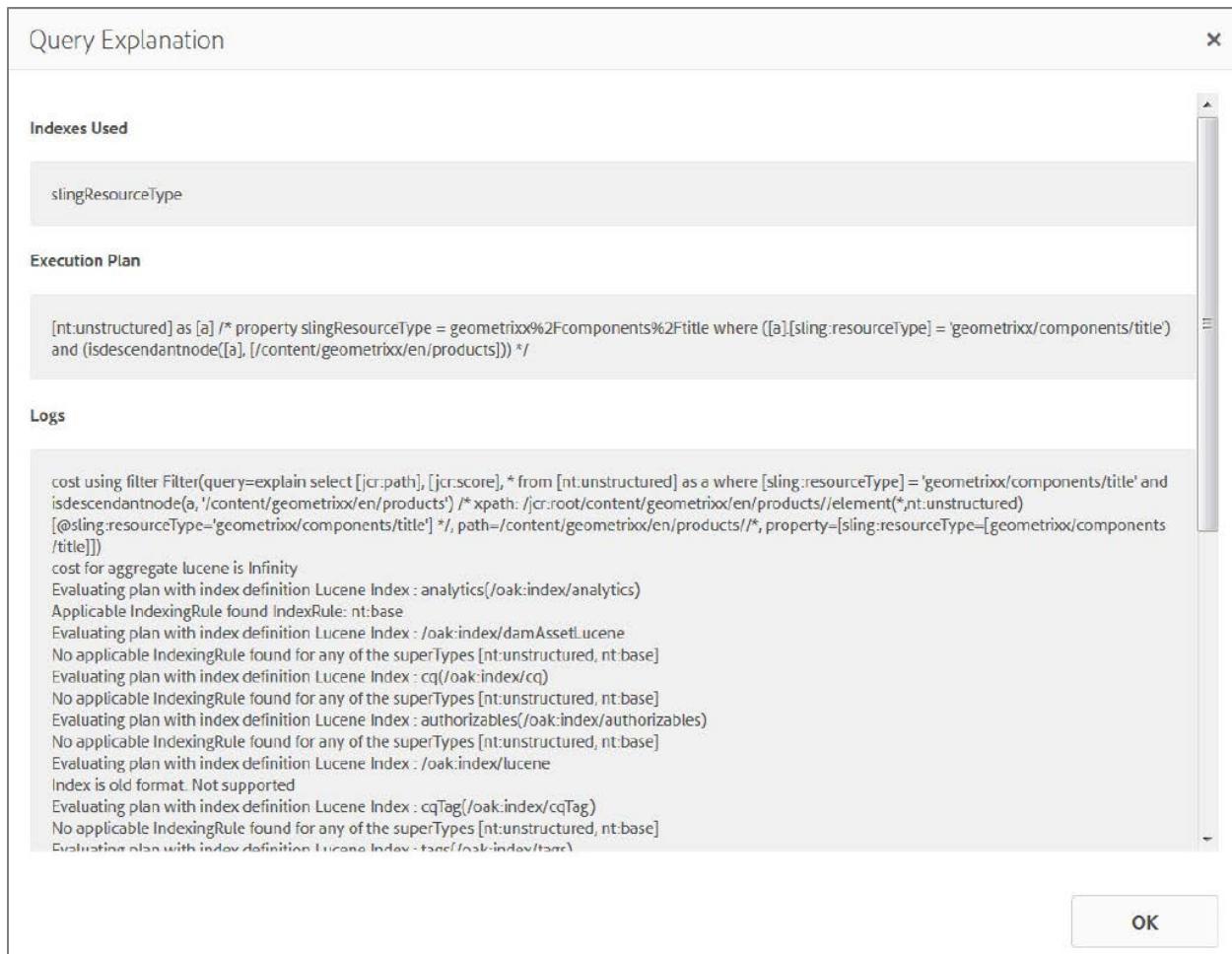
Explain Query Tool

This is a tool designed to help administrators understand how queries are executed. Oak attempts to figure out the best way to execute any given query, based on the Oak indexes defined in the repository under the oak:index node. Depending on the query, different indexes may be chosen by Oak. Understanding how Oak is executing a query is the first step to optimizing the query.

Features:

- Supports the Xpath, JCR-SQL, and JCR-SQL2 query languages
- Reports the actual execution time of the provided query
- Detects slow queries and warns about queries that could be potentially slow
- Reports the Oak index used to execute the query
- Displays the actual Oak Query engine explanation
- Provides click-to-load list of Slow and Popular queries

The screenshot shows the 'Query Performance' page of the Adobe Experience Manager Commons toolkit. At the top, there are tabs for 'SLOW QUERIES', 'POPULAR QUERIES', and 'EXPLAIN QUERY', with 'EXPLAIN QUERY' being the active tab. Below the tabs, there are fields for 'Language *' (set to 'xPath') and 'Query *' containing the XPath query '/jcr:root/content/geometrixx/en/products//element(*,nt:unstructured)[@sling:resourceType='geometrixx/components/title']'. There are also two checkboxes: 'Include Execution Time' and 'Include Node Count', both of which are checked. At the bottom left is a large 'Explain' button.



The first entry in the Query Explanation section is the actual explanation. The explanation will show the type of index that was used to execute the query.

The second entry is the query plan. Selecting the **Include execution time** checkbox before running the query will also show the amount of time the query was executed in, allowing for more information that can be used for optimizing the indexes for your application or deployment.

The tool will also build a list of popular and slow queries that can be automatically loaded in the UI by selecting their name in the predefined list.

Oak Index Manager

The Oak Index Manager is a simple Web UI created to facilitate index management such as maintaining indexes, viewing their status, or triggering re-indexes. You can use the UI to filter indexes in the table by typing in the filter criteria in the search box in the upper-left corner of the screen.

You can trigger a single reindex by clicking the icon in the Reindex column for the respective index. You can trigger a bulk reindex by selecting multiple indexes and clicking the Bulk Reindex button.

With Adobe Experience Manager, both the Explain Query and Oak Index Manager tools are available by going to **Tools > Operations > Dashboard > Diagnosis..**

Query SYNTAX

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

`Join` transforms 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. For example, nodes that are children of /pictures
- Name of the node.
- Value of a property. For example, nodes whose jcr:created property is after 2007-03-14T00:00:00.000Z.
- Length of a property. For example, nodes whose jcr:data property is longer than 100 KB.
- Existence of a property. For example, nodes with a jcr:description property.
- Full-text search. For example, nodes that have a property containing the phrase "beautiful sunset."

Search Basics

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

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

- Get the QueryManager for the Session/Workspace:

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

- Create the query statement:

- Execute the query:

```
QueryResult res = q.execute();
```

- Iterate through the results:

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

Query Examples—SQL2

- Find all nt:folder nodes.

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

- 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. Sort the 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

Java Query Object Model (JQOM) is a mapping of AQM to Java API, and 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:

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

JQOM Examples

- Find all nt:folder nodes.

```
1. QueryObjectModelFactory qf = qm.getQOMFactory();
2. Source source = qf.selector("content:folder", "contentfolder");
3. QueryObjectModel query = qf.createQuery(source, null,
4. null,null);
```

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

```

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

```

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

```

1. QueryObjectModelFactory qf = qm.getQOMFactory();
2. Source source = qf.join(qf.selector("nt:file", "file"), qf.
3. selector("rep:User", "user"), QueryObjectModelFactory.JCR _ 
4. JOIN _ TYPE _ INNER,qf.equiJoinCondition("file","jcr:createdBy",
5. "user","rep:principalName"));
6. Constraint pathConstraint = qf.and(qf.descendantNode("file", "/
7. var"), qf.not(qf.descendantNode("file", "/var/classes")));
8. Ordering orderings[] = {qfascending(qf.propertyValue("file",
9. "jcr:createdBy")), qf.ascending(qf.propertyValue("file",
10. "jcr:created")) };
11. QueryObjectModel query = qf.createQuery(source,
12. 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. to the total number of nodes

Which JCR search methodologies are fast?

- Constraints on the path

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 on the Tools menu from the top toolbar in CRXDE Lite.

The screenshot shows the CRXDE Lite interface. On the left is a tree view of the AEM repository structure under the path /apps/geometrix/config/com.day.cq.mcm.impl.MCMConfiguration-geometrixxconfig. On the right, there is a 'Query' editor window. The 'Type' dropdown is set to 'SQL2' and the 'Path' is '/content'. The 'Text' field contains the search term 'geometrix'. The 'Query' field contains the SQL-like query: 'SELECT * FROM [nt:base] AS s WHERE ISDESCENDANTNODE(/content) and CONTAINS(s.* , 'geometrix')'. Below the query are 'Generate', 'Execute', and 'Show result' buttons. The 'Show result' checkbox is checked. The results table shows 10 rows of paths, all starting with '/content/geometrix'. The last row is highlighted. At the bottom, it says 'Execution Info: 14 results (115msec)'.

Path
1 /content/geometrix/en/services/banking/jcr:content/par/text
2 /content/geometrix/en/services/banking/jcr:content
3 /content/geometrix/en/services/banking
4 /content/geometrixx-media/en/community/jcr:content/grid-4-par/welcome_message
5 /content/geometrixx-media/en/community/jcr:content
6 /content/geometrixx-media/en/community
7 /content/dam/geometrix/documents/GeoMetrix_Banking.indd/jcr:content/renditions/page/jcr:content/par/text_u1b0
8 /content/dam/geometrix/documents/GeoMetrix_Banking.indd/jcr:content/renditions/page/jcr:content
9 /content/dam/geometrix/documents/GeoMetrix_Banking.indd/jcr:content/renditions/page
10 /content/dam/geometrix/documents/GeoMetrix_Banking.html/jcr:content/renditions/GeoMetrix_Banking.html/jcr:content



Perform Task –[Exercise 2 – Write Queries for a Search Servlet from Lab H.](#)

LAB H – IMPLEMENT THE JCR API

Scenario

In your project, you must keep a track of some changes/updates made to the repository, without requiring any task to be performed. You need to only display a notification of the change on your page. For example, if there is a change made to the title of the page, then the notification about that change would be visible on the page.

Challenge

You must identify which properties need to be tracked for changes. When a change is made to the same property by multiple users, it is difficult to track any change beyond the first one.

Objectives

- Create an Observation Listener
- Write Queries for a Search Servlet

Prerequisites

- AEM installed on your machine:
 - Running Adobe Experience Manager Author instance
 - An Eclipse project from the AEM Archetype

Directions

Complete the exercises that follow

Exercise 1 – Create an Observation Listener

Overview

In this lab exercise, you will create an observation listener for a change to a title.

Steps

1. Create an Observation Listener.
 - a. Open Eclipse and navigate to: **training.core > src/main/java > com.adobe.training.core.listeners**
 - b. Create a new class: **TitlePropertyListener**
 - c. Paste the code from /Exercise_Files/09_Deep_Dive_JCR.



Note: The code is provided as part of the Exercise_Files under /Exercise_Files/09_Deep_Dive_JCR/. Copy and paste the code from the exercise files to Eclipse. Do not copy from this exercise book. The following is for illustration purposes only.

```
1. package com.adobe.training.core.listeners;
2.
3.
4. import javax.jcr.Property;
5. import javax.jcr.RepositoryException;
6. import javax.jcr.Session;
7. import javax.jcr.observation.Event;
8. import javax.jcr.observation.EventIterator;
9. import javax.jcr.observation.EventListener;
10. import javax.jcr.observation.ObservationManager;
11. import org.apache.felix.scr.annotations.Component;
12. import org.apache.felix.scr.annotations.Reference;
13. import org.apache.sling.jcr.api.SlingRepository;
14. import org.osgi.service.component.ComponentContext;
15. import org.slf4j.Logger;
16. import org.slf4j.LoggerFactory;
17.
```

```
18. @Component
19. public class TitlePropertyListener implements EventListener {
20.     private final Logger logger = LoggerFactory.getLogger(getClass());
21.
22.     @Reference
23.     private SlingRepository repository;
24.
25.     private Session session;
26.     private ObservationManager observationManager;
27.
28.     protected void activate(ComponentContext context) throws Exception {
29.         session = repository.loginService("training",null);
30.         observationManager = session.getWorkspace().getObservationManager();
31.
32.         observationManager.addEventListener(this, Event.PROPERTY_ADDED | Event.PRO
33.             PERTY_CHANGED, "/content/trainingproject/fr", true, null,
34.             new String[]{"cq:PageContent","nt:unstructured"}, true);
35.         logger.info("*****added JCR event listener");
36.     }
37.     protected void deactivate(ComponentContext componentContext) {
38.         try {
39.             if (observationManager != null) {
40.                 observationManager.removeEventListener(this);
41.                 logger.info("*****removed JCR event listener");
42.             }
43.         catch (RepositoryException re) {
44.             logger.error("*****error removing the JCR event listener ", re);
45.         }
46.         finally {
47.             if (session != null) {
48.                 session.logout();
49.                 session = null;
50.             }
51.         }
52.     }
53.     public void onEvent(EventIterator it) {
54.         while (it.hasNext()) {
55.             Event event = it.nextEvent();
56.             try {
57.                 Property changedProperty = session.getProperty(event.getPath());
58.                 if (changedProperty.getName().equalsIgnoreCase("jcr:title")
59.                     && !changedProperty.getString().endsWith("!")) {
60.                     changedProperty.setValue(changedProperty.getString() + "!");
61.                     logger.info("*****Property updated: {}", event.getPath());
62.                     session.save();
63.                 }
64.             }
65.         }
66.     }
67. }
```

```

63.        }
64.    }
65.    catch (Exception e) {
66.        logger.error(e.getMessage(), e);
67.    }
68. }
69.
70. }

```



NOTE: The method call `addEventListener`: Event types are bitwise OR'ed. The last parameter (`noLocal`) prevents changes made by this session to be sent to the listener, which would result in an endless loop in this case. We also specify a specific path `/content/trainingproject/fr` that this listener will attach to. Any property changes outside this path will not result in this listener being called.

The `onEvent` method checks changed/added `jcr:title` properties and appends a "!" if the property does not already end with a "!".

2. Build the project by selecting **training.core** and click **Run As > Maven install**.
3. Verify the component is installed at: <http://localhost:4502/system/console/components>

Id		Name
3000	com.adobe.training.core.listeners.TitlePropertyListener	
	Bundle	com.adobe.training.core (521)
	Implementation Class	com.adobe.training.core.listeners.TitlePropertyListener
	Default State	enabled
	Activation	immediate
	Configuration Policy	optional
	Services	
	PID	com.adobe.training.core.listeners.TitlePropertyListener
	Reference repository	Satisfied Service Name: org.apache.sling.jcr.api.SlingRepository Cardinality: 1..1 Policy: static Policy Option: reluctant Bound Service ID 1591 (com.adobe.granite.repository.impl.SlingRepositoryManager)
	Properties	component.id = 3000 component.name = com.adobe.training.core.listeners.TitlePropertyListener service.pid = com.adobe.training.core.listeners.TitlePropertyListener service.vendor = Adobe

- a. Change property `jcr:title` in CRXDE Lite. Refresh CRXDE Lite to see if "!" is added by **com.adobe.training.core.TitlePropertyListener** component.
- b. To kick off our listener, you need to change the `jcr:title` property under `/content/trainingproject/fr` to the French page in our TrainingProject Site, <http://localhost:4502/editor.html/content/trainingproject/fr.html>.

- c. Go to the **Page information > Open Properties**.

The screenshot shows the 'Page Properties' dialog box for a page named 'MyPage'. The dialog has tabs at the top: BASIC (selected), ADVANCED, THUMBNAIL, CLOUD SERVICES, PERSONALIZATION, and PERMISSIONS. The title bar includes 'Français', 'Cancel', 'Save & Close', and a dropdown menu. The 'Title and Tags' section contains fields for 'Name' (empty), 'Title *' (containing 'MyPage'), 'Tags' (empty), and a checkbox for 'Hide in Navigation' (unchecked). The background of the dialog shows a preview of the page content.

- d. Change the Title, and click **Save & Close**.

The page will reload and you should see the Page Title have an appended "!" on it. (It should be in the gray editing bar at the top of the page.)

The screenshot shows the AEM page preview for 'MYPAGE'. The page content includes the text 'SLING SERVLET INJECTED THIS TITLE' repeated three times. Below the content, there is developer information: 'Created by Scott Reynolds. Hobbies include: [Hiking]. Preferred programming language in AEM is: HTL'. At the bottom, there is a snippet of Java code from 'HelloWorldModel' and a paragraph of placeholder text ('Lorem ipsum' etc.). The top navigation bar shows the page title 'MYPAGE!', the language 'English', and the page name 'MyPage!'.

4. Because this is a JCR listener, you can also go into CRXDE Lite and modify a jcr:title property under **/content/trainingproject/fr**. Feel free to try this on your own.

NOTE: To see the new string, you must refresh CRXDE Lite after saving your changes.

Properties		Access Control	Replication	Console	Build Info
	Name ▲	Type	Value		
7	jcr:lastModified	Date	2017-03-01T16:24:53.121-08:00		
8	jcr:lastModifiedBy	String	admin		
9	jcr:primaryType	Name	nft:unstructured		
10	jcr:title	String	MyNewPage		
11	pageTitle	String	Nouveau Project		
12	sling:resourceType	String	trainingproject/components/structure/page		

5. Verify the new title was added:

The screenshot shows the AEM authoring interface. At the top, there are icons for document, preview, and properties. The title bar says 'MYNEWPAGE!'. On the right, there are 'Edit' and 'Preview' buttons, and language selection 'English MyNewPage!'. The main content area displays the page's body. It features a red 'Adobe' logo. Below it, the page has two subtitles: 'SLING SERVLET INJECTED THIS TITLE' in green, and another 'SLING SERVLET INJECTED THIS TITLE' in blue. At the bottom of the page content, there is a block of developer information and logs:

```

HelloWorldModel says:
Hello World!
This is instance: 9dbef65d-5431-44e4-a96a-74c5d4ffcd5e
Resource type is: trainingproject/components/content/helloworld

```

Review

You created an observation listener that looks for any changes made to the jcr:title property of a page, and displayed the notification accordingly.

Exercise 2 – Write Queries for a Search Servlet

Overview:

In this exercise, you will create and observe two different APIs to search the JCR. You will use SQL and JQOM to search for a full text phrase in a website and then output the pages that contain that phrase. This will also be a use case to understand.

Steps:

1. Create a Servlet for Search
 - a. Open Eclipse and open the **training** project.
 - b. Create a servlet named **SearchServlet** under **com.adobe.training.core.servlets** with the following code:

```
1. package com.adobe.training.core.servlets;
2.
3. import java.io.IOException;
4. import java.util.ArrayList;
5. import java.util.Arrays;
6.
7. import javax.jcr.Node;
8. import javax.jcr.Nodelterator;
9. import javax.jcr.RepositoryException;
10. import javax.jcr.ValueFactory;
11. import javax.jcr.query.Query;
12. import javax.jcr.query.QueryManager;
13. import javax.jcr.query.qom.Constraint;
14. import javax.jcr.query.qom.QueryObjectModel;
15. import javax.jcr.query.qom.QueryObjectModelFactory;
16. import javax.jcr.query.qom.Selector;
17. import javax.servlet.ServletException;
18.
19. import org.apache.felix.scr.annotations.sling.SlingServlet;
20. import org.apache.sling.api.SlingHttpServletRequest;
21. import org.apache.sling.api.SlingHttpServletResponse;
22. import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
23. import org.apache.sling.commons.json.JSONArray;
24. import org.apache.sling.commons.json.JSONObject;
```

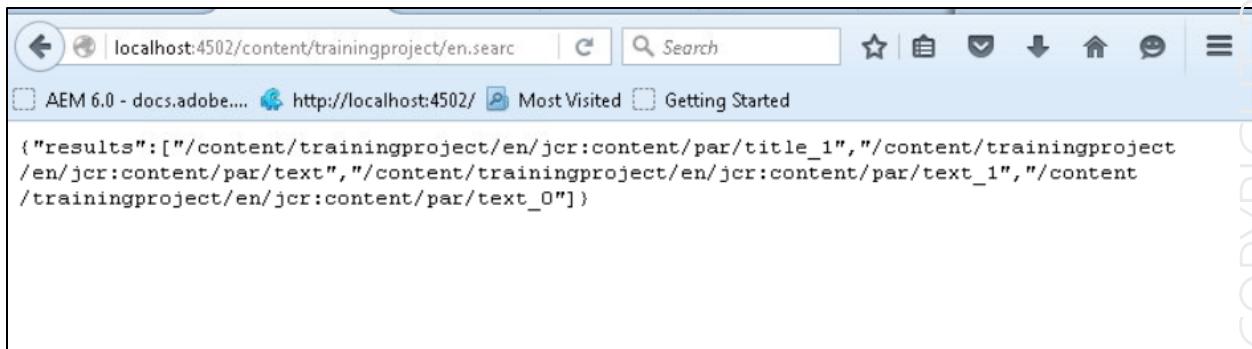
```
25.
26. import com.day.cq.wcm.api.PageManager;
27.
28. /**
29. * Example URI: http://localhost:4502/content/trainingproject/en.search.sql.html?q=ipsum&wc
30. mmode=disabled
31. *
32. * Example URI: http://localhost:4502/content/trainingproject/en.search.jqom.html?q=Lorem&
33. wcmmode=disabled
34. *
35. @SlingServlet(resourceTypes = "trainingproject/components/structure/page", selectors="search
36. ")
36. public class SearchServlet extends SlingSafeMethodsServlet {
37.
38.     private static final long serialVersionUID = 3169795937693969416L;
39.
40.     @Override
41.     public final void doGet(final SlingHttpServletRequest request, final SlingHttpServletResponse
42. response)
43.             throws ServletException, IOException {
44.         response.setHeader("Content-Type", "application/json");
45.         JSONObject jsonObject = new JSONObject();
46.         JSONArray resultArray = new JSONArray();
47.
48.         try {
49.             //current node that is requested
50.             Node currentNode = request.getResource().adaptTo(Node.class);
51.
52.             //cq:page node containing the requested node
53.             PageManager pageManager = request.getResource().getResourceResolver().adaptTo(Pa
54. geManager.class);
55.             Node queryRoot = pageManager.getContainingPage(currentNode.getPath()).adaptTo(N
56. ode.class);
57.
58.             String queryTerm = request.getParameter("q");
59.             if (queryTerm != null) {
60.                 //get the selectors from the URI
61.                 String[] selectors = request.getRequestPathInfo().getSelectors();
62.                 ArrayList<String> language = new ArrayList<String>(Arrays.asList(selectors));
63.
64.                 //Search with JQOM or SQL depending on the selector
65.                 NodeIterator searchResults = null;
66.                 if(language.contains("jqom")) {
67.                     searchResults = performSearchWithJQOM(queryRoot, queryTerm);
```

```
65. } else if(language.contains("sql")) {
66.     searchResults = performSearchWithSQL(queryRoot, queryTerm);
67. }
68.
69. //Add the search results into the json array
70. if(searchResults != null) {
71.     while (searchResults.hasNext()) resultArray.put(searchResults.nextNode().getPath())
72. ;
73.     }
74. jsonObject.put("results", resultArray);
75. }
76. catch (Exception e) {
77.     e.printStackTrace();
78. }
79. response.getWriter().print(jsonObject.toString());
80. response.getWriter().close();
81.
82. }
83.
84. private NodeIterator performSearchWithJQOM(Node queryRoot, String queryTerm) throws RepositoryException {
85.
86.     // JQOM infrastructure
87.     QueryObjectModelFactory qf = queryRoot.getSession().getWorkspace().getQueryManager()
88.         .getQOMFactory();
89.     ValueFactory vf = queryRoot.getSession().getValueFactory();
90.
91.     final String SELECTOR_NAME = "all results";
92.     final String SELECTOR_NT_UNSTRUCTURED = "nt:unstructured";
93.     // select all unstructured nodes
94.     Selector selector = qf.selector(SELECTOR_NT_UNSTRUCTURED, SELECTOR_NAME);
95.
96.     // full text constraint
97.     Constraint constraint = qf.fullTextSearch(SELECTOR_NAME, null, qf.literal(vf.createValue(q
98.         ueueTerm)));
99.     // path constraint
100.    constraint = qf.and(constraint, qf.descendantNode(SELECTOR_NAME, queryRoot.getPath()
101.        ));
102.
103.    // execute the query without explicit order and columns
104.    QueryObjectModel query = qf.createQuery(selector, constraint, null, null);
105.    return query.execute().getNodes();
```

```

106. private NodeIterator performSearchWithSQL(Node queryRoot, String queryTerm) throws RepositoryException {
107.     QueryManager qm = queryRoot.getSession().getWorkspace().getQueryManager();
108.     Query query = qm.createQuery("SELECT * FROM [nt:unstructured] AS node WHERE ISDESCENDANTNODE(["
109.         + queryRoot.getPath() + "]) AND CONTAINS(node.*,'" + queryTerm + "')", Query.JCR_SQL2);
110.     return query.execute().getNodes();
111. }
112.
113. }
```

2. Build the project by selecting **training.core** and click **Run As > Maven install**.
3. To test searching with SQL, you need to use the sql selector. For instance:
<http://localhost:4502/content/trainingproject/en.search.sql.html?q=ipsum&wcmmode=disabled>



4. To test searching with JQOM, you need to use the jqom selector:

For instance:

<http://localhost:4502/content/trainingproject/en.search.jqom.html?q=Lorem&wcmmode=disabled>

Review:

In this exercise, you implemented a Sling selector that performs a full text search, and returns the query results as JSON.

10 DEEP DIVE INTO AEM APIs

Overview

This module does a deep dive into the various APIs that allow automation of various tasks like importing data and displaying it on a website, creating custom workflow steps, and automating the creation of content like pages and assets. These APIs are at the highest layer of the AEM architecture stack and are most closely related to the various business tasks a typical content producer would work with.

Objectives

By the end of this module, you will be able to:

- Create custom Polling Importers
- Connect External Data to a AEM Component
- Create custom Workflow process steps
- Setup workflow launchers
- Programmatically create an AEM Page
- Create AEM Pages based upon a CSV file input

Polling Importers

The feed importer is a framework to repeatedly import content from external sources into your repository. The idea behind a feed importer is to poll a remote resource at a specified interval, parse it, and create nodes in the content repository that represent the content of the remote resource. In Adobe Experience Manager Sites (out-of-the-box), the feed importer is used for the following:

- In blogs to support the auto-blogging feature, which automatically creates blog posts from an external RSS or Atom feed.
- In calendars 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 the Feed Importer node opens a page that allows configuration of standard feed importers for RSS, Atom, Calendar, IMAP, and POP3.

To implement your own feed importer, use the interface: `com.day.cq.pollingimporter.Importer`



Perform Task – [Exercise 1 –Create a Polling Importer](#) from Lab I.

Content Components

Components are modular units which realize specific functionality to present your content on your website. They are developed as self-contained units within one folder of the repository and can also be developed to create customized components that extend the default functionality.

In real time scenario, you might have many requirements and might need to design a component based on the requirements. Let us customize a component with the following requirements:

- A component that will return stock information.
- A component that fits in with the website look and feel.
- Allows the author to specify
 - which stock is of interest
 - Validating the stock is a valid stock symbol
 - a summary about the stock feed
- Allows the author to choose to display
 - The date the request was made
 - The time the request was made
 - Whether the stock price is up or down
 - The stock's opening price
 - The high and low ranges

- The stock volume
- A component that allows the user to download stock history
- A component that can be dragged into a paragraph system

Note: Due to environment constraints, our component will use canned data.

After the component has been developed, you add it to the paragraph system, which enables authors to select and use the component when editing a page. The component stores its content in a paragraph on that page.



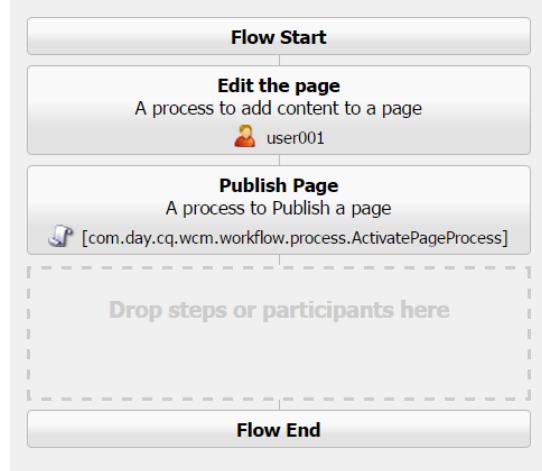
Perform Task –[Exercise 2 – Connect External Data to an AEM Component](#) from Lab I.

AEM Workflows

In Adobe Experience Manager, workflows are used to automate processes and activities. They are a set of steps performed in a specific order. Each step performs a distinct activity such as activating a page or sending an email message. Workflows can interact with assets in the repository, user accounts, and services. You can pass data from one step to another, invoke workflows automatically, and create custom workflows.

An example of a workflow in Adobe Experience Manager is publishing a page. An editor needs to review a page then site administrator needs to activate it before it can be published. In Adobe Experience Manager, you can create a workflow that automates this process and notifies each of the participants when it is time to perform their assigned tasks. These workflows are specific to your business organization.

The following is a sample workflow used to edit and publish a page in Adobe Experience Manager.



Understanding the Workflow Objects

A workflow is associated with the following objects:

- Model: A model consists of nodes and transitions. The transitions connect the nodes and define the flow. The model always has a start node and an end node. Workflow models are versioned. Running workflow instances keep the initial workflow model version that is set when the workflow is started.
- Steps: Workflow models consist of a series of steps of various types, which can be extended with scripts to provide the functionality and control you require.
- Transition: A transition defines the link between two consecutive steps. You can apply rules to a transition.
- WorkItem: This is an object that represents a task or action in the workflow system at runtime. A workflow instance can have more than one WorkItems at the same time.
- Payload: It is an entity upon which a workflow instance acts. For example, a page in Adobe Experience Manager could be passed from step-to-step as a payload, which could be either as a JCR node, a JCR path, or an identifier.
- Lifecycle: The lifecycle of a workflow begins when the workflow is started, and ends when the end node is processed. You can apply the following actions on a workflow: terminate, suspend, resume, and restart. Completed and terminated instances are archived.
- Inbox: Logged-in users have their own workflow inbox in which the assigned WorkItems are accessible. The WorkItems are assigned either to a user itself or to the group to which the user belongs.

EXECUTING AN EXISTING WORKFLOW

Adobe Experience Manager comes with a set of predefined workflows that perform common actions. You can customize those workflows if the built-in steps do not include all the necessary tasks. You can execute a workflow from any of the following places:

- Context menu
- Workflow console
- Site Admin (classic UI)
- Sidekick (classic UI)

Depending on the type of workflow, the payload goes through a series of steps until its completion. Workflow launchers let you automatically invoke a workflow based on certain conditions, such as page modification. You can set the Workflow Launcher configuration to start a workflow when a specified node selection or nodes of a specified type are created, modified, or deleted.



Perform Task –[Exercise 3 – Execute a Workflow](#) from Lab I.

Defining Workflow Models

Based on your business requirements, there are two actions that you can perform with respect to workflows: modifying an existing workflow, or creating a new one. You can use the Workflow Console to manage workflow models and launchers. A workflow model includes a Flow Start and a Flow End step. These steps indicate the beginning and end of the workflow. All other steps are contained within these two steps.

There are many steps available for a workflow. Two of the most common steps used in workflows are:

- Participant step: Requires manual intervention by a person to advance the workflow.
- Process step: Automatic actions that are executed by the system if specific conditions are met.

Every new model created includes a sample participant step, which you can either edit or remove. You can add and configure additional steps as required.

THE WORKFLOW CONSOLE

The Workflow Console is a centralized location for managing workflows.

The screenshot shows the Adobe Marketing Cloud Workflow Console interface. The left sidebar has categories: Workflow, Models, Instances, Launchers, Archive, and Failures. The main area is titled 'Models' and shows a list of workflow models. Each model entry includes a checkbox, the model name, a description, and its version number. The models listed are: SampleModel (No Description, Version 1.2), DAM Update Asset (This workflow manages the update of assets, Version 1.0), DAM Create Language Copy (This workflow creates language copies for assets, Version 1.0), DAM Create and Translate Language Copy (This workflow creates and translates language copies for assets, Version 1.0), WCM: Prepare Translation Project (Workflow to Prepare Translation Project, Version 1.0), WCM: Update Language Copy (Workflow to update an existing language copy using a launch, Version 1.0), and Request to complete Move operation (No Description, Version 1.0). There is also a 'Card View' button at the top right of the list.

	Title	Description	Version Transfer
<input type="checkbox"/>	SampleModel	No Description	1.2
<input type="checkbox"/>	DAM Update Asset	This workflow manages the update of assets	1.0
<input type="checkbox"/>	DAM Create Language Copy	This workflow creates language copies for assets	1.0
<input type="checkbox"/>	DAM Create and Translate Language Copy	This workflow creates and translates language copies for assets	1.0
<input type="checkbox"/>	WCM: Prepare Translation Project	Workflow to Prepare Translation Project	1.0
<input type="checkbox"/>	WCM: Update Language Copy	Workflow to update an existing language copy using a launch	1.0
<input type="checkbox"/>	Request to complete Move operation	No Description	1.0

UNDERSTANDING WORKFLOW STEPS

Before we start creating workflows, let us look at what a workflow step is. As mentioned earlier, a workflow consists of one or more steps. Each step can contain any number of actions and associated conditions. For example, a step in a publish workflow may involve approval from an editor. Some steps may require manual intervention, while others may be automatic.

In Adobe Experience Manager, there are a number of steps available for workflows such as Participant, Process, Create Task, Delete Node, Dialog Participant, Dynamic Participant, Form Participant, and many more. Two of the most commonly used workflow steps are Participant and Process.

PARTICIPANT STEP

Enables you to assign a step to a user or a group of users. If the workflow is assigned to just one user, then that user needs to complete that task before the workflow can proceed to the next step. If the workflow is assigned to a group of users, then all those users need to complete the step.

You can notify participants of their required action through email. Also, if configured, the participants will receive an email notification when the workflow is completed, or if the workflow is terminated.

You can configure timeouts and timeout handlers for this step. Timeout is the period after which the step will be timed out. You can select between Off, Immediate, and, if you want to specify specific blocks of time: 1h, 6h, 12h, and 24h. The timeout handler controls the workflow when the step times out.

PROCESS STEP

- Executes an ECMA script or calls an OSGi service to automate the process.
- Offers built-in processes that you can use:
 - Workflow control processes: Control the behavior of the workflow, and do not perform any action on content.
 - Basic processes: Deleting a node, or logging a debug message.
 - WCM processes: WCM-related tasks such as activating a page, or confirming registration.
 - Versioning processes: Version-related tasks such as creating versions of the payload.
 - DAM processes: DAM-related tasks such as creating thumbnails, creating sub-assets, and extracting metadata.
 - Collaboration Processes: Related to the collaboration features of Adobe Experience Manager, such as that of social communities.

Developing Custom Steps

You can extend workflow steps with scripts to provide more functionality and control. You can create customized process steps using the following methods:

- Java class bundles: Create a bundle with the Java class, and then deploy the bundle into the OSGi container using the Web Console.
- ECMA scripts: Scripts are located in the JCR repository under etc/workflows, and they are executed from there. To use a custom script, create a new script with the extension .ecma under the same folder. The script will then show up in the process list for a process step.

An exercise later in the module shows how you can create a custom process step using Java, and how to include that step in a customized workflow.

Creating a Workflow

Now that you have learned about the basic workflow steps, and how to create custom process steps, you can now proceed to the creation of the actual workflow.

To create a workflow:

1. Open the Workflow Console, and create a new model.
2. Double-click the newly created model, and modify the steps by clicking and dragging the required workflow steps from the sidekick to the workflow.
3. Edit the properties of the steps.
4. Save the workflow.

You can execute this workflow either manually, or by configuring a launcher. We talk about launcher later in the module.



Perform Task –[Exercise 4 – Create a Custom Workflow Step](#) from Lab I.

Workflow Launchers

The Workflow launcher enables you to invoke a workflow based on certain predefined conditions. These conditions are based on changes to the content located in the JCR. For example, when a page is modified, it can trigger a workflow.

You can configure the workflow launchers through the Workflow console (<http://localhost:4502/libs/cq/workflow/admin/console/content/launchers.html>).

Workflow Launchers							
	Description	Event Type	Node type	Condition	Workflow	Enabled	Exclude
<input type="checkbox"/>	Globbing /content/dam(/*)renditions/original	Parse Word documents (DOC) that have been added to the DAM	Node created	nt:file	jcr:content/jcr:mimeType==application/msword	DAM Parse Word Documents	true
<input type="checkbox"/>	/content/dam(/*)renditions/original	Parse Word documents (DOCX) that have been added to the DAM	Node created	nt:file	jcr:content/jcr:mimeType==application/vnd.openxmlformats-officedocument.wordprocessingml.document	DAM Parse Word Documents	true
<input type="checkbox"/>	/content/dam(/*)renditions/original		Node created	nt:file		DAM Update Asset	true
<input type="checkbox"/>	/var/lightbox		Node created	nt:file		DAM Update from Lightbox	true
<input type="checkbox"/>	/etc/commerce/products/geometrixx-outdoors	Prepare created products for SAINT export	Node created	nt:unstructured		SAINT Product Export Handler	false
<input type="checkbox"/>	/home/users(/*)mail(/*)sentitems(/*)	Replicating the pathholder node under SentItems on creation	Node created	nt:unstructured	messageBoxRelativePath==/mail/geometrixx/sentitems	/etc/workflow/models/Activate_Content/jcr:content/model	true
<input type="checkbox"/>	/home/users(/*)mail(/*)sentitems(/*)	Replicating the pathholder node under SentItems on creation	Node created	nt:unstructured	messageBoxRelativePath==/mail/geometrixx-media/sentitems	/etc/workflow/models/Activate_Content/jcr:content/model	true
<input type="checkbox"/>	/home/users(/*)mail(/*)inbox(/*)	Replicating the pathholder node under inbox node on creation	Node created	nt:unstructured	messageBoxRelativePath==/mail/geometrixx-media/inbox	/etc/workflow/models/Activate_Content/jcr:content/model	true

For example, to publish a page after it is modified, you would use the following values for the properties:

- **Event type:** `modified`: Type of event that triggers the workflow.
- **Node type:** `nt:unstructured`: Type of node that is affected by the workflow.
- **Path:** `/content/Geometrixx`: Property that indicates the path of the node.
- **Workflow:** `PublishPage`: Property that indicates the workflow to be executed when the event occurs.
- **Activate:** `Enable`: Property that controls whether the launcher should be activated.
- **Run mode (s):** `Author`: Property that indicates the type of server that the launcher applies to.

Difference in using Workflow Launchers and JCR Observations or Sling eventing:

Workflow Launcher	JCR Observation / Sling Eventing
Has a UI	Does not have a UI
Easy to start or stop	Requires the use of Web Console to stop the listener component
Workflow model can be changed dynamically	Requires programmatic or configuration changes
Involves more overhead, and is ideal to use if there are only moderate amounts of event expected	Involves less overhead, and can handle more frequent events
It is cluster-aware, and runs only on the cluster master	Sling eventing is not cluster-aware

Monitoring Performance of Workflows

You can monitor the performance of workflows through its reporting interface, at:

<http://localhost:4502/etc/reports/workflowreport.html>

This interface provides the following information:

- Details on number of workflows and its duration
- Links for various workflows, and a breakdown of its steps



Perform Task –[Exercise 5 – Use the Workflow Launcher to Monitor Polling events](#) from Lab I.

Creating AEM Pages and Assets Programmatically

The following sections are best practices used in data migration.

Creating Pages Dynamically

Adobe Experience Manager has some very powerful high-level APIs that you can use to create pages based on the underlying data structures. You can create paragraph or text nodes, tag a page, or activate a page.

A page consists of nodes, and properties along with a `sling:resourceType`. You can use the `com.day.cq.wcm.api.PageManager` class to point to the underlying resources in XML and automate the process of data migration.

Creating Assets Dynamically

Assets consist of nodes, and properties along with a `sling:resourceType`. You can use the `com.day.cq.dam.api.AssetManager` to create the asset, and the `com.day.cq.tagging.TagManager` APIs to tag the assets. Using these APIs, you can create assets, tag them, as well as add metadata to the assets.

You can use this process for migration by pointing to the existing assets and creating the asset in a Digital Asset Manager, with all the asset information that is required.

Identifying Cost Benefit

Based on the complexity and size of the migrating system, you can use any of the following three methods for migration:

- Manual migration (human entry)
Hire someone to enter the data into a format that is compatible with Adobe Experience Manager.
- Automated migration (using APIs)
Export data as XML from the legacy system, then write a script to convert the data into a JCR friendly XML, and finally import that package into JCR.
- Hybrid migration (combination of manual and automated)
Import the bulk of the data through XML, and hire someone to update the data manually; for example, for adding metadata to modernize the data.



Perform Task –[Exercise 6 – Programmatically create AEM Pages](#) from Lab I.

LAB I – IMPLEMENT AEM APIs

Scenario

You need to obtain the Adobe (ADBE) share price and display it on your page. Because the price is fairly stable, you need to reach out to Yahoo Stock at a predefined interval and import the stock information into AEM. The imported stock information can then be read by a stock component that needs the information without having to make costly requests to a 3rd party service on page load. This will be the imported stock information used by the Stockplex component.

You will also need to implement a custom workflow process and trigger it by a workflow launcher. The workflow process will kick off whenever the stock price changes and then alert the user in the form of a log message.

Lastly you must create an automated process for creating pages based upon a csv file that the business produces.

Challenge

Select the right importer, and make use of the share price as per your requirement. Create custom workflows and a process for automating page creation.

Objectives

- Create a Polling Importer
- Connect External Data to an AEM Component
- Execute a Workflow
- Create a Custom Workflow Step
- Programmatically Create AEM Pages

Prerequisites

- AEM installed on your machine:
 - Running Adobe Experience Manager Author instance
 - An Eclipse project from the AEM Archetype

Directions

Complete the exercises that follow

Exercise 1 –Create a Polling Importer

Overview:

In this exercise, you will create a polling importer to create a stock ticker for Adobe system share price and display it on your page.

Steps:

1. Create a Polling Importer.
 - a. In Eclipse, create new **StockDataImporter** class file under *training.core > src/main/java > com.adobe.training.core* with the following code:



Note: The code is provided as part of the Exercise_Files under /Exercise_Files/10_Deep_Dive_AEM_APIs/. Copy and paste the code from the exercise files to Eclipse. Do not copy from this exercise book. The following is for illustration purposes only.

```

1. package com.adobe.training.core;
2.
3. import java.io.BufferedReader;
4. import java.io.IOException;
5. import java.io.InputStreamReader;
6. import java.net.MalformedURLException;
7. import java.net.URL;
8.
9. import javax.jcr.Node;
10. import javax.jcr.RepositoryException;
11. import javax.jcr.Session;
12.
13. import org.apache.felix.scr.annotations.Component;
14. import org.apache.felix.scr.annotations.Property;
15. import org.apache.felix.scr.annotations.Reference;
16. import org.apache.felix.scr.annotations.Service;
17. import org.apache.sling.api.resource.Resource;
18. import org.apache.sling.jcr.api.SlingRepository;
19. import org.slf4j.Logger;
20. import org.slf4j.LoggerFactory;
21.
22. import com.day.cq.commons.jcr.JcrUtil;
23. import com.day.cq.polling.importer.ImportException;
24. import com.day.cq.polling.importer.Importer;
25.
26. /**
27. * This class imports from Yahoo Finance and creates the data structure example below
28. *

```

```

29. * /content
30. * + <STOCK_SYMBOL> [cq:Page]
31. * + lastTrade [nt:unstructured]
32. * - lastTrade = <value>
33. * - requestedDate = <value>
34. * - requestTime = <value>
35. * - upDown = <value>
36. * - openPrice = <value>
37. * - rangeHigh = <value>
38. * - rangeLow = <value>
39. * - volume = <value>
40. *
41. * @author Kevin Nennig (nennig@adobe.com)
42. *
43. */
44. @Service(value = Importer.class)
45. @Component
46. @Property(name = "importer.scheme", value = "stock", propertyPrivate = false)
47. public class StockDataImporter implements Importer {
48.     private final Logger logger = LoggerFactory.getLogger(getClass());
49.
50.     private final String SOURCE_URL = "http://download.finance.yahoo.com/d/quotes.csv
?f=s1d1t1c1ohgv&e=.csv&s=";
51.
52.
53.     private static final String LASTTRADE = "lastTrade";
54.     private static final String REQUESTDATE = "requestDate";
55.     private static final String REQUESTTIME = "requestTime";
56.     private static final String UPDOWN = "upDown";
57.     private static final String OPENPRICE = "openPrice";
58.     private static final String RANGEHIGH = "rangeHigh";
59.     private static final String RANGELOW = "rangeLow";
60.     private static final String VOLUME = "volume";
61.
62.     @Reference
63.     private SlingRepository repo;
64.
65.     @Override
66.     public void importData(final String scheme, final String dataSource, final Resource resource)
67.             throws ImportException {
68.         try {
69.             // dataSource will be interpreted as the stock symbol
70.             URL sourceUrl = new URL(SOURCE_URL + dataSource);
71.             BufferedReader in = new BufferedReader(new InputStreamReader(sourceUrl.openStream()));
72.             String readLine = in.readLine(); // expecting only one line
73.             String[] lastTrade = readLine.split(",");
74.             logger.info("Last trade for stock symbol {} was {}", dataSource, lastTrade);
75.             in.close();
76.
77.             //persist
78.             writeToRepository(dataSource, lastTrade, resource);
79.         }
80.         catch (MalformedURLException e) {
81.             logger.error("MalformedURLException", e);
82.         }
83.         catch (IOException e) {
84.             logger.error("IOException", e);
85.         }

```

```

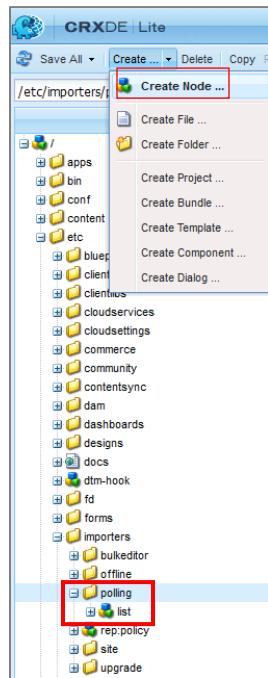
86.         catch (RepositoryException e) {
87.             logger.error("RepositoryException", e);
88.         }
89.
90.     }
91.
92.     /**
93.      * Creates the Yahoo stock data structure
94.      *
95.      * + <STOCK_SYMBOL> [cq:Page]
96.      * + lastTrade [nt:unstructured]
97.      * - lastTrade = <value>
98.      * - requestedDate = <value>
99.      * - requestTime = <value>
100.     * - upDown = <value>
101.     * - openPrice = <value>
102.     * - rangeHigh = <value>
103.     * - rangeLow = <value>
104.     * - volume = <value>
105.    */
106.   private void writeToRepository(final String stockSymbol, final String[] lastTrade
107. , final Resource resource) throws RepositoryException {
108.     Session session= repo.loginService("training",null);
109.     Node parent = resource.adaptTo(Node.class);
110.     Node stockPageNode = JcrUtil.createPath(parent.getPath() + "/" + stockSymbol,
111. "cq:Page",
112.             session);
113.     Node lastTradeNode = JcrUtil.createPath(stockPageNode.getPath() + "/lastTrade"
114. ", "nt:unstructured",
115.             session);
116.     if(lastTrade.length > 8){
117.         lastTradeNode.setProperty(LASTTRADE, lastTrade[1]);
118.         lastTradeNode.setProperty(REQUESTDATE, lastTrade[2].replace("\n", ""));
119.         lastTradeNode.setProperty(REQUESTTIME, lastTrade[3].replace("\n", ""));
120.         lastTradeNode.setProperty(UPDOWN, lastTrade[4]);
121.         lastTradeNode.setProperty(OPENPRICE, lastTrade[5]);
122.         lastTradeNode.setProperty(RANGEHIGH, lastTrade[6]);
123.         lastTradeNode.setProperty(RANGELOW, lastTrade[7]);
124.         lastTradeNode.setProperty(VOLUME, lastTrade[8]);
125.     }
126.     session.save();
127.     session.logout();
128.   }
129.   @Override
130.   public void importData(String scheme, String dataSource, Resource target,
131.     String login, String password) throws ImportException {
132.     importData(scheme, dataSource, target);
133.   }
134. }
```

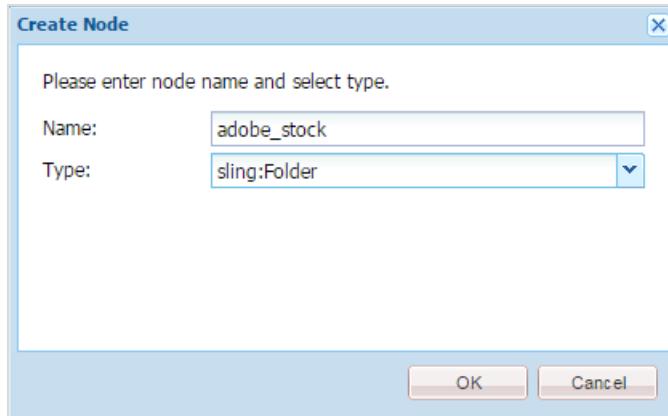
b. Save the changes



NOTE: Note that we are implementing the interface com.day.cq.polling.importer.Importer in the code. You can explore the code and understand that we will be using many variables from JCR. You will create these variables in the JCR repository as a configuration node.

2. Deploy the project by selecting “training.core” project and clicking *Run As > Maven install*.
 - a. Verify the project is successfully deployed
3. Create a new node named adobe_stock.
 - a. Open CRXDE <http://localhost:4502/crx/de/index.jsp> and navigate to *etc > importers > polling*
 - b. Create a new node
 - Name: adobe_stock
 - Type: sling:Folder
 - Save All

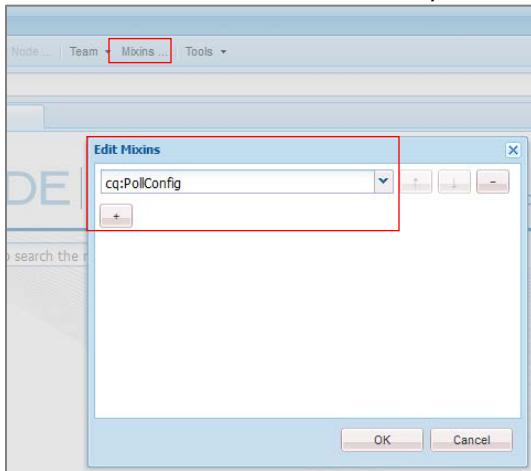




4. On the adobe_stock folder node, add the following properties:

- Name: interval
Type: Long
Value: 300
- Name: source
Type: String
Value: stock:ADBE
- Name: target
Type: String
Value: /content

5. Click **Mixins..** in the toolbar at the top, click **+**, and add cq:PollConfig then click **OK**



This configuration lets the JCR know this is a Mixins configuration.

- a. Click **Save All** to save the properties created on the adobe_stock node.



Note: The configurations we created will provide the information to the custom polling importer we created, StockDataImporter.java class. As a result, in 300 seconds (5 min) you should see a new node created at the target path (/content) provided in the node.

6. Refresh /content in CRXDE Lite and notice that there is a ADBE node created.

Under the ADBE node, there is a lastTrade node and the properties on the lastTrade node is the data we imported from Yahoo stock

The screenshot shows the CRXDE Lite interface. On the left is a tree view of the repository structure, including catalogs, experience-fragments, we-retail, oak:index, trainingproject, ADBE, and etc. The ADBE node is expanded, and its child node 'lastTrade' is selected. On the right is a 'Properties' tab showing the following table:

Name	Type	Value
1 jcr:primaryType	Name	nt:unstructured
2 lastTrade	String	100
3 requestDate	String	11/13/2016
4 requestTime	String	4:00pm

Exercise 2 – Connect External Data to an AEM Component

Overview

In this exercise, you will connect external data to an AEM component and install content package for the Stockplex component.

Steps

7. Observe StockModel.java which was created earlier in the class.

This StockModel.java represents the data model imported by the Polling importer in the previous exercise.

8. Install the **StockPlex component** content package.

We will install the package for the front end Stockplex component and add to the page. This will automatically use the data that is being imported.

- a. Navigate to Package Manager: <http://localhost:4502/crx/packmgr/index.jsp>
- b. Click Upload Package.



- c. In the Upload Package dialog box, click Browse, and select the **Stockplex Frontend Package.zip** package from /Exercise_Files/10_Deep_Dive_AEM_APIs/ then click Open.
- d. Once the Upload Package dialog box pre-populates with the **Stockplex Frontend Package.zip** filename
- e. Click **ok**

A screenshot of the CRX Package Manager showing a package named "We.Train_Ch14_Stockplex_Complete-BackendReady.zip". The package details are as follows:

- Build: 1 | Last built 2016/10/05 | admin
- Install | 14.6 KB
- Edit | Build | Install | Download | Share
- More ▾
- Package: We.Train_Ch14_Stockplex_Complete-BackendReady
- Download: We.Train_Ch14_Stockplex_Complete-BackendReady.zip (14.6 KB)
- Group: my_packages
- Filters: /apps/training/components/content/stockplex

Note: The package will be installed under /app/training/components/content/.

- Click **Install** on the content package

9. Check the Activity Log below the **Stockplex Frontend Package.zip** information.

- a. Verify you see the content that was added from the package:

The screenshot shows a window titled "Activity Log" with a blue header bar. The main content area contains a list of URLs indicating files being imported, such as "/apps/training/components/content/stockplex/clientlibs/validation.js/jcr:content" and "/apps/training/components/content/stockplex/design_dialog/items". At the bottom of the log, it says "Package imported." followed by "Package installed in 369ms.".

```
A /apps/training/components/content/stockplex/clientlibs/validation.js/jcr:content
A /apps/training/components/content/stockplex/clientlibs/validation.js/jcr:content/jcr:data
A /apps/training/components/content/stockplex/clientlibs/css.txt
A /apps/training/components/content/stockplex/clientlibs/css.txt/jcr:content
A /apps/training/components/content/stockplex/clientlibs/css.txt/jcr:content/jcr:data
A /apps/training/components/content/stockplex/clientlibs/js.txt
A /apps/training/components/content/stockplex/clientlibs/js.txt/jcr:content
A /apps/training/components/content/stockplex/clientlibs/js.txt/jcr:content/jcr:data
A /apps/training/components/content/stockplex/stockplex.js
A /apps/training/components/content/stockplex/stockplex.js/jcr:content
A /apps/training/components/content/stockplex/cq:editConfig
A /apps/training/components/content/stockplex/design_dialog
A /apps/training/components/content/stockplex/design_dialog/items
A /apps/training/components/content/stockplex/design_dialog/items/items
A /apps/training/components/content/stockplex/design_dialog/items/items/tab1
A /apps/training/components/content/stockplex/design_dialog/items/items/tab1/items
A /apps/training/components/content/stockplex/design_dialog/items/items/tab1/items/downloadBox
saving approx 54 nodes...
Package imported.

Package installed in 369ms.
```

10. Enable the Stockplex component

- a. Using the Sites Console (<http://localhost:4502/sites.html/content>)
- b. Navigate to *Sites > TrainingProject Site > English* and open the page

The screenshot shows the AEM 6.0 interface. At the top, there's a navigation bar with options: Create, Edit, Properties, Lock, Copy, Move, and Quick Publish. Below the navigation bar is a sidebar with a tree view of site structures: Campaigns, Screens, Community Sites, We.Retail, TrainingProject Site, and ADBE. The 'TrainingProject Site' node is selected. To the right of the sidebar, a modal window titled 'NEW PROJECT FOR AEM 6.0' is displayed. The modal contains the following information:

	Value
Title	English
Name	en
Modified	1 day ago
Modified By	Administrator
Page Title	New Project
Language	English
Published	45 minutes ago

11. Edit the page.

- In the top right corner select the dropdown next to Edit and select Design Mode

The screenshot shows the AEM page editor in Design Mode. The page content includes a heading 'SLING SERVLET INJECTED THIS TITLE' and a 'Developer Info:' section containing the following text:

```
HelloWorldModel says:  
Hello World!  
This is instance: 10d0b03e-d358-4748-b521-0780bb4b22a5
```

On the right side, there is a dropdown menu with options: Edit (which is highlighted), Scaffolding, Developer, and Design.

- Click on the paragraph system, and then click on the wrench.

SLING SERVLET INJECTED THIS TITLE

Developer Info: Created by Scott Reynolds. Hobbies include: [Hiking]. Preferred programming language in AEM is: HTL

SLING SERVLE

HelloWorldModel says:
Hello World!
This is instance:
Resource type is:

SLING SERVL

Duis autem vel eum iriure do
molestie consequat, vel illum
eros et accumsan et iusto od
zzril delenit augue duis dolore
sit amet, consetetur adipisci
tincidunt ut laboreet dolore ma

Ut wisi enim ad minim veniam
suscipit lobortis nisl ut aliquip
vel eum iriure dolor in hendre
consequat, vel illum dolore ei
accumsan et iusto odio dignis
delenit augue duis dolore te feugait nulla facilisi.

Nam liber tempor cum soluta nobis eleifend option congue nihil

ParSys Design

Allowed Components Settings

- > We.Retail
- > We.Retail Client App
- > We.Retail Commerce
- > We.Retail Form
- > We.Retail.Structure
- > St StockPlex Component
- > Workflow
- > (No group)

1

aliquyam erat, sed diam
Lorem ipsum dolor sit
magna aliquyam erat, sed
s est Lorem ipsum dolor

vulputate velit esse
ut nulla facilisis.

et ea rebum. Stet clita
Lorem ipsum dolor sit
sadipscing elitr, sed diam
dolore magna aliquyam
am et justo duo dolores et
akimata sanctus est
or sit amet, consetetur
am dolore dolores duo
invidunt justo labore Stet
clita ea et gubergren, kasd magna no rebum. sanctus sea sed takimata
ut vero voluptua. est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit
amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor

- c. In the dialog of Available Components, navigate the We.Train group and select the StockPlex component
 - d. Click Done
12. Configure the StockPlex component into the paragraph system.
- a. Switch to Edit Mode.

- b. Click on the Left Rail Toggle, and then click on the Components tab.

The screenshot shows the AEM authoring interface. On the left, the 'Components' tab is selected in the left rail, displaying a list of components under 'TrainingProject Components'. The list includes 'Columns', 'Hello World Component', 'Image', 'StockPlex Component', 'Text', 'Text and Image', and 'Title'. The 'StockPlex Component' is highlighted. On the right, the page content area displays the 'SLING SERVLET INJ' component, which outputs developer information and a placeholder for a StockPlex component.

- c. Drag the *StockPlex* component into the paragraph system
d. Click on the Placeholder for the StockPlex and click the Wrench

The screenshot shows the 'StockPlex Component' placeholder being edited. A context menu is open over the component, with the 'Edit' option highlighted. The component itself is labeled 'StockPlex Component'.

- e. Enter the values as shown:

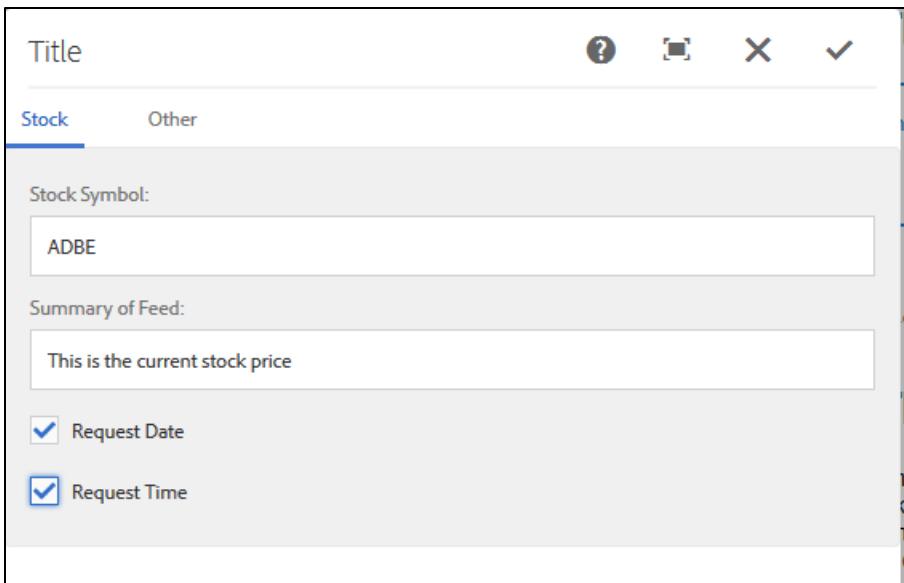
Title

Stock Other

Stock Symbol:
ADBE

Summary of Feed:
This is the current stock price

Request Date
 Request Time

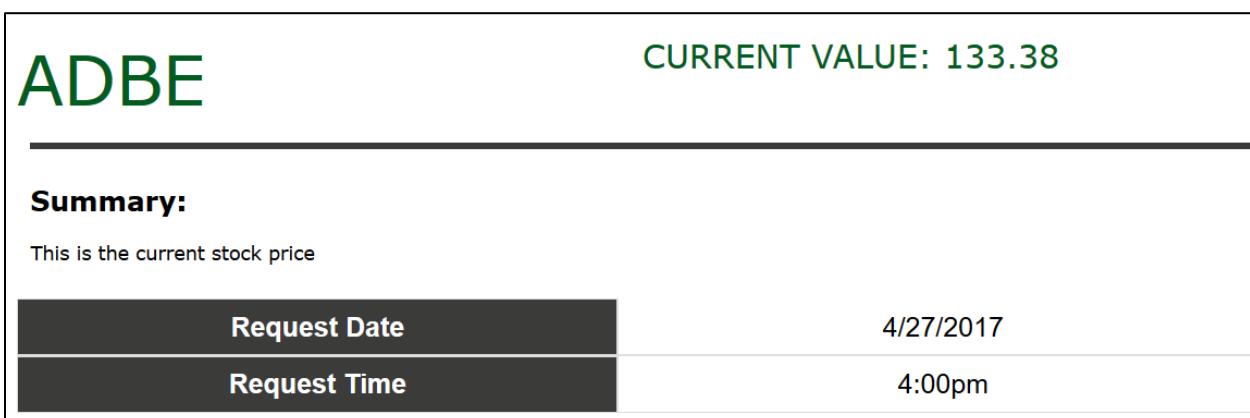


The Stock symbol has to be match the stock symbol in the JCR under /content.

ADBE CURRENT VALUE: 133.38

Summary:
This is the current stock price

Request Date	4/27/2017
Request Time	4:00pm



Exercise 3 – Execute a Workflow

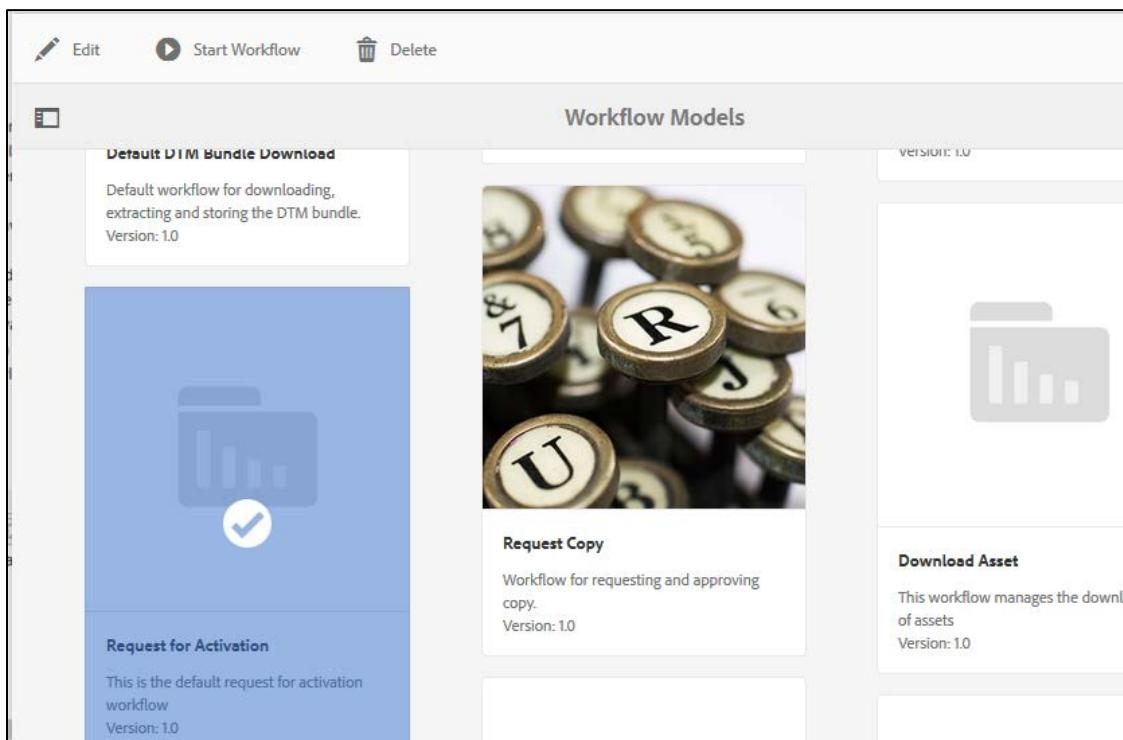
Overview:

Adobe Experience Manager supports multiple ways to implement a process step. You can create your own workflow model as per your requirement.

You have custom or out-of-the-box workflows available in Adobe Experience Manager and they can be executed manually or automatically. To execute a workflow automatically, you should create a launcher that will trigger the workflow. You will manually start a workflow from the workflow console and implement a process step in an existing workflow.

Steps:

1. Execute a workflow from workflow console.
 - a. In the Author instance, navigate to *Tools > Workflow > Models*
 - b. Click the checkmark on the Request for Activation workflow/card (you may want to search for the word “activation” to help you locate it).
 - c. When you select it, a blue highlight appears (as shown).



- d. Click Start Workflow in the top left corner to open the Run Workflow page.

The Run Workflow page will ask for details such as Payload, Title, and Comment.

2. Enter the page location: /content/trainingproject/en) in the mandatory Payload field (as shown in the following screenshot) and then click Run.

It may take a few seconds, but you will see a "green form has been submitted successfully box" notification indicating the process has begun.

The screenshot shows a modal dialog titled "Run Workflow". It contains three input fields: "Payload *", "Title", and "Comment". The "Payload *" field has the value "/content/trainingproject/en" entered. Below the fields are two buttons: "Cancel" and "Run".

The workflow process will begin. A notification will appear in the **Inbox** (upper-right) to approve page content:

Title	Priority	Description	Assignee	Project	Workflow	Status	Start Date	Due Date
Approve content	Medium	Approve page content	administrators			Completed	34 seconds ago	
Configure Analytics & Targeting	Medium	You can opt in for Analytics and Targeting by selecting your configuration and then adding it to your pages.	administrators			Pending		
Apply the AEM Security Checklist	High	It is strongly recommended to Apply the official AEM Security Checklist on	administrators			Pending		



NOTE: The page will not be published until it is approved. Recall the process is two-fold: 1. Edit an initial version of the page, 2. Publish the page. Each of these steps requires approval.

Exercise 4 – Create a Custom Workflow Step

Overview:

In this lab, you will create a custom workflow and test it.

Steps:

1. Implement a process step in an existing workflow model

- a. In Eclipse, create new **MyProcess** class file under *training.core > src/main/java > com.adobe.training.core* with the following code:

```
1. package com.adobe.training.core;
2. import com.day.cq.workflow.WorkflowException;
3. import com.day.cq.workflow.WorkflowSession;
4. import com.day.cq.workflow.exec.WorkItem;
5. import com.day.cq.workflow.exec.WorkflowData;
6. import com.day.cq.workflow.exec.WorkflowProcess;
7. import com.day.cq.workflow.metadata.MetaDataMap;
8. import org.apache.felix.scr.annotations.Component;
9. import org.apache.felix.scr.annotations.Properties;
10. import org.apache.felix.scr.annotations.Property;
11. import org.apache.felix.scr.annotations.Service;
12. import org.osgi.framework.Constants;
13. import javax.jcr.Node;
14. import javax.jcr.RepositoryException;
15.
16. @Component
17. @Service
18. @Properties({ @Property(name = Constants.SERVICE_DESCRIPTION, value = "A s
ample workflow process implementation."),
19.                 @Property(name = Constants.SERVICE_VENDOR, value = "Adobe"),
20.                 @Property(name = "process.label", value = "My Sample Workflow Proc
ess") })
21. public class MyProcess implements WorkflowProcess {
22.
23.     private static final String TYPE_JCR_PATH = "JCR_PATH";
24.
25.     public void execute(WorkItem item, WorkflowSession session, MetaDataMa
p args) throws WorkflowException {
26.         WorkflowData workflowData = item.getWorkflowData();
27.         if (workflowData.getPayloadType().equals(TYPE_JCR_PATH)) {
28.             String path = workflowData.getPayload().toString() + "/jcr:con
tent";
29.             try {
30.                 Node node = (Node) session.getSession().getItem(path);
31.                 if (node != null) {
32.                     node.setProperty("approved", readArgument(args));
33.                     session.getSession().save();
34.                 }
35.             } catch (RepositoryException e) {
36.                 throw new WorkflowException(e.getMessage(), e);
37.             }
38.         }
39.     }
40. }
```

```

37.         }
38.     }
39. }
40.
41. private boolean readArgument(MetaDataMap args) {
42.     String argument = args.get("PROCESS_ARGS", "false");
43.     return argument.equalsIgnoreCase("true");
44. }
45. }

```

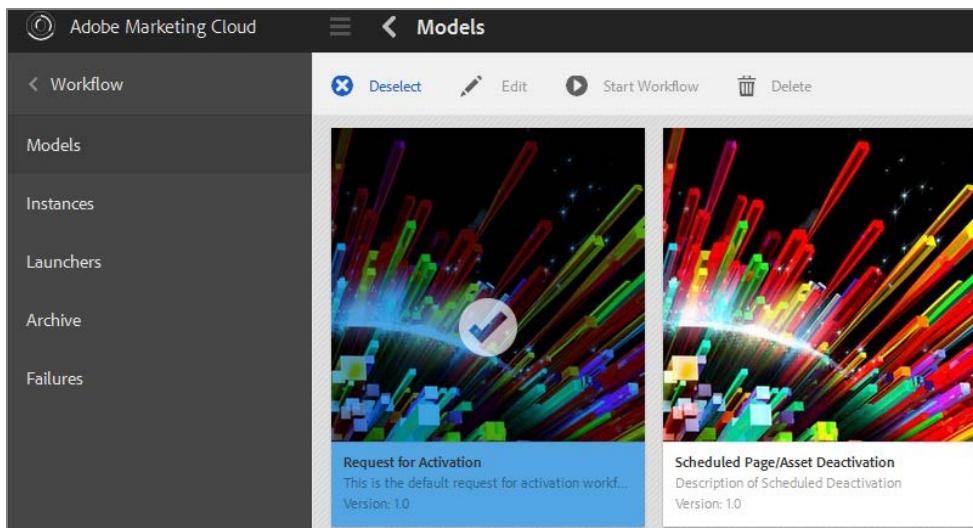
This will create an OSGI service, which will be used as the workflow process step. The service adds the property approved to the page content node when the payload is a page.

2. Deploy the project by selecting “training.core” project and clicking *Run As > Maven install*.

- Verify the project is successfully deployed

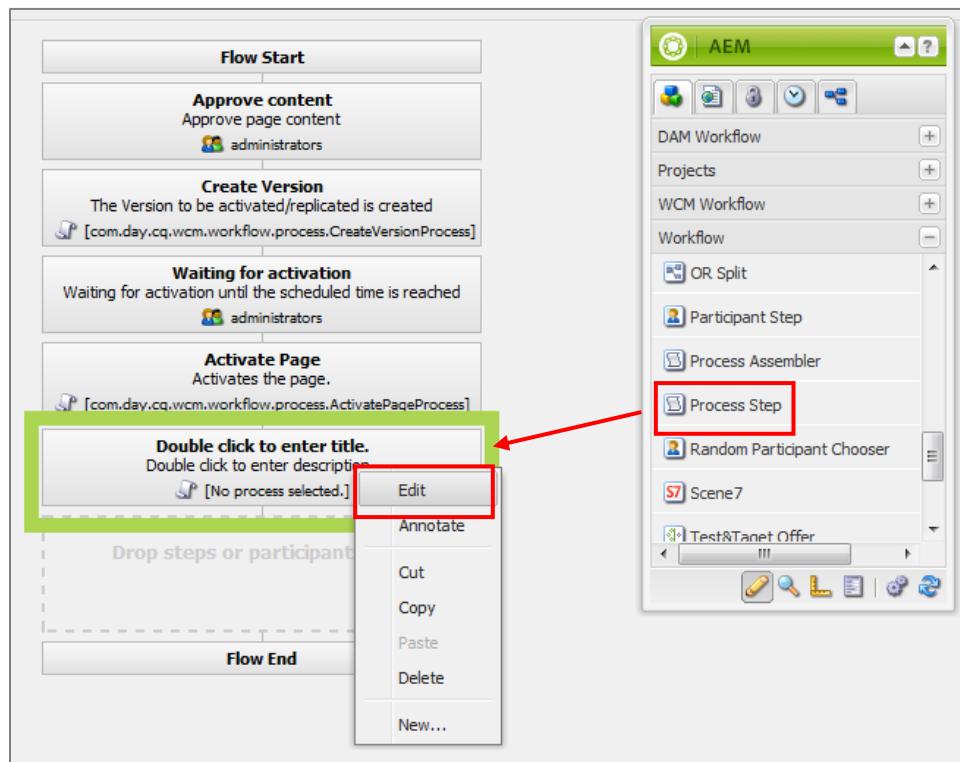
3. Navigate to *Tools > Workflow > Models*, and select Request for Activation workflow.

- a. Click the checkmark then click Edit.

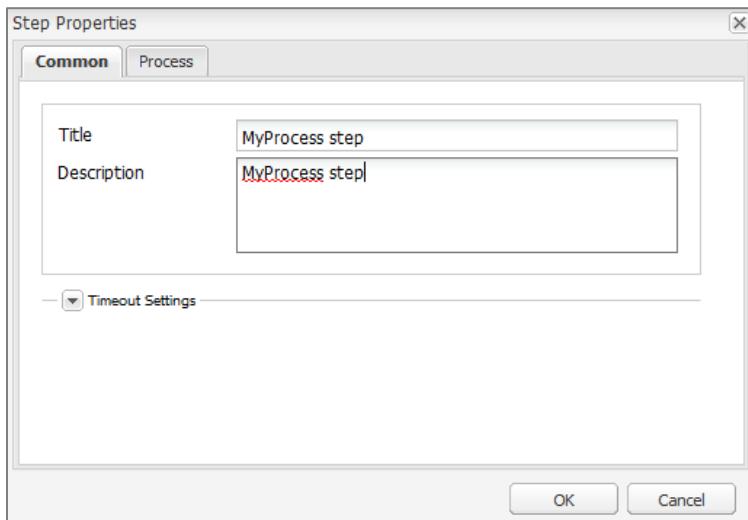


- a. In the editor, the workflow steps will be shown as per the default configuration.
- b. Drag and drop Process Step from the Workflow section (make sure to click + next to Workflow; it is very easy to miss this – clicking the + will expand the workflows available to you) to the area with the dotted line “Drop steps or participants here”

c. Right-click and Edit the Process Step.

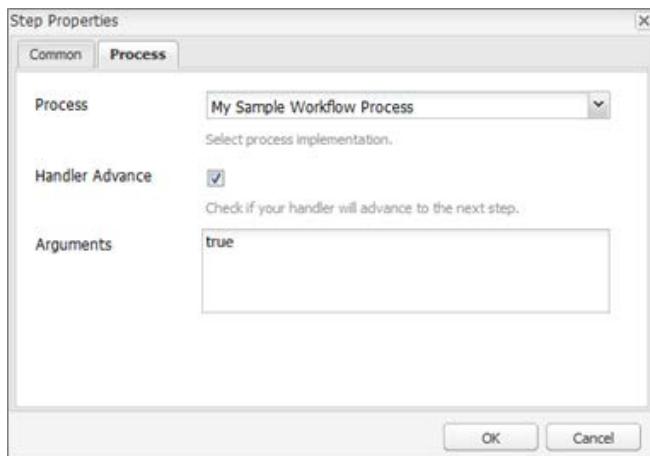


d. Provide the step properties: Title and Description.



e. Click the *Process* tab and select My Sample Workflow Process from the Process drop-down.
Note that this process (OSGI Service) was created when you added the MyProcess.java class.

- f. Select Handler Advance in order to tell the workflow to advance to the next step. Set the Argument value to true.



- g. Click OK
- h. Click Save in the toolbar (underneath Request for Activation at the top of the page) to save the workflow.
4. Return to the Workflow Models card view, and note that the workflow model Request for Activation has a New blue label in the upper-right corner of the card.
- a. Select this workflow model, and click Start Workflow.
5. Select a page for payload field then click Run.
- You will notice that a new node property is created by the workflow.
6. Click on **Inbox** (Bell icon) in the top right corner, and complete the participant steps for the workflow by selecting it and then clicking on Complete.
7. Repeat this for both participant steps.
- a. Use CRXDE Lite to check properties on the page used as payload.

You will notice that a new node property approved is created by the workflow and set to true by the argument provided.

Exercise 5 –Use the Workflow Launcher to Monitor Polling events

Overview:

In this exercise, you will listen for the stock price changing from the custom stock importer we created earlier. If the price goes above a certain threshold value, we want to log it.

Steps:

8. Use the Workflow launcher to monitor polling events

Now, you will create a workflow that alerts authors when a stock value exceeds a certain threshold.

- a. In Eclipse, create new **StockAlertProcess** class file under *training.core > src/main/java > com.adobe.training.core* with following code:



Note: The code is provided as part of the Exercise_Files under /Exercise_Files/10_Deep_Dive_AEM_APIs/. Copy and paste the code from the exercise files to Eclipse. Do not copy from this exercise book. The following is for illustration purposes only.

```
1. package com.adobe.training.core;
2.
3. import java.util.Arrays;
4. import java.util.HashMap;
5. import java.util.Iterator;
6. import java.util.Map;
7. import java.util.regex.Pattern;
8.
9. import javax.jcr.Node;
10. import javax.jcr.RepositoryException;
11. import javax.jcr.Session;
12.
13. import org.apache.felix.scr.annotations.Component;
14. import org.apache.felix.scr.annotations.Property;
15. import org.apache.felix.scr.annotations.Reference;
16. import org.apache.felix.scr.annotations.Service;
17. import org.apache.sling.api.resource.LoginException;
18. import org.apache.sling.api.resource.Resource;
19. import org.apache.sling.api.resource.ResourceResolver;
20. import org.apache.sling.api.resource.ResourceResolverFactory;
21. import org.slf4j.Logger;
22. import org.slf4j.LoggerFactory;
23.
24. import com.adobe.training.core.models.StockModel;
25. import com.day.cq.workflow.WorkflowException;
26. import com.day.cq.workflow.WorkflowSession;
27. import com.day.cq.workflow.exec.WorkItem;
28. import com.day.cq.workflow.exec.WorkflowData;
29. import com.day.cq.workflow.exec.WorkflowProcess;
30. import com.day.cq.workflow.metadata.MetaDataMap;
31.
32. /**
33. * To have this class trigger with every imported stock symbol,
```

```

34. * make sure the workflow launcher has a Globbing Path of /content(/.* /)lastT
   rade (no spaces)
35. *
36. * This class assumes the following jcr data structure for the stock
37. * /content
38. *   + <stock symbol> [cq:Page]
39. *     + lastTrade [nt:unstructured]
40. *       - lastTrade = <imported stock value>
41. */
42.
43. @Service
44. @Component(metatype = false)
45. @Property(name = "process.label", value = "Stock Threshold Checker")
46. public class StockAlertProcess implements WorkflowProcess {
47.
48.     private static final String TYPE_JCR_PATH = "JCR_PATH";
49.     private static final String TYPE_JCR_UUID = "JCR_UUID";
50.     private final Logger logger = LoggerFactory.getLogger(getClass());
51.
52.     @Reference
53.     private ResourceResolverFactory resourceResolverFactory;
54.
55.     @Override
56.     public void execute(WorkItem workItem, WorkflowSession workflowSession, Me
taDataMap args)
57.         throws WorkflowException {
58.     try {
59.         // get the node the workflow is acting on
60.         Session session = workflowSession.getSession();
61.
62.         //allows us to get the resourceResolver based on the current sessi
on
63.         Map<String, Object> params = new HashMap<String, Object>();
64.         params.put("user.jcr.session", session);
65.         ResourceResolver resourceResolver = null;
66.         try {
67.             resourceResolver = resourceResolverFactory.getResourceResolver(
68.                 params);
69.         } catch (LoginException e) {
70.             logger.error("@@@@LoginError" + e);
71.         }
72.         if(resourceResolver != null){
73.             WorkflowData data = workItem.getWorkflowData();
74.             Node payloadNode = null;
75.             String type = data.getPayloadType();
76.             //make sure the payload is a valid jcr path
77.             if(type.equals(TYPE_JCR_PATH) && data.getPayload() != null) {
78.
79.                 String payloadData = (String) data.getPayload();
80.                 if(session.itemExists(payloadData)) {
81.                     payloadNode = session.getNode(payloadData);
82.                 }
83.             }
84.         }
85.
86.         Resource stockResource = resourceResolver.getResource(payload
Node.getParent().getPath());

```

```

87.          //Create StockModel from the resource
88.          StockModel stock = stockResource.adaptTo(StockModel.class);
89.          String symbol = stock.getStockSymbol();
90.          logger.info("@@@@Checking stock: " + symbol);
91.
92.          Double lastTrade;
93.          try {
94.              lastTrade = stock.getLastTrade();
95.              logger.info("@@@@last trade was " + lastTrade);
96.
97.              //parse the passed workflow arguments into an iterator. Ex
98.              : "ADBE=105 \n MSFT=55"
99.              Iterator<String> argumentsIterator = Arrays.asList(
100.                  Pattern.compile("\n").split(args.get("PROCESS_ARGS
101.                  ", ""))).iterator();
102.              //Iterate over each argument
103.              while (argumentsIterator.hasNext()) {
104.                  String argument = argumentsIterator.next();
105.                  //Check if the argument is the stock symbol in questi
106.                  on
107.                  if(argument.contains(symbol)){
108.                      Double thresholdTrade = Double.parseDouble(argument
109.                          .split("=")[1]);
110.                      //Check to see if the newly imported price is hig
111.                      //her than the threshold
112.                      if (thresholdTrade < lastTrade) {
113.                          logger.warn("@@@@ Stock Alert! " + symbol +
114. " is over " + thresholdTrade);
115.                      }
116.                  }
117.                  else
118.                  {
119.                      logger.error("@@@@ResourceResolver is null");
120.                  }
121.              }
122.              catch (RepositoryException e) {
123.                  logger.error("@@@@RepositoryException", e);
124.              }
125.          }
126.      }

```

b. Save the changes

9. Deploy the project.
 - a. Select the **training.core** project and choose **Run As > Maven install**.
10. Navigate to **Tools > Workflow > Models** or open <http://localhost:4502/libs/cq/workflow/admin/console/content/models.html/etc/workflow/models>
 - a. Click **Create** to create a new workflow model with the following details:
 - Title: StockAlert
 - Name: StockAlert
 - b. Click **Done**

Add Workflow Model

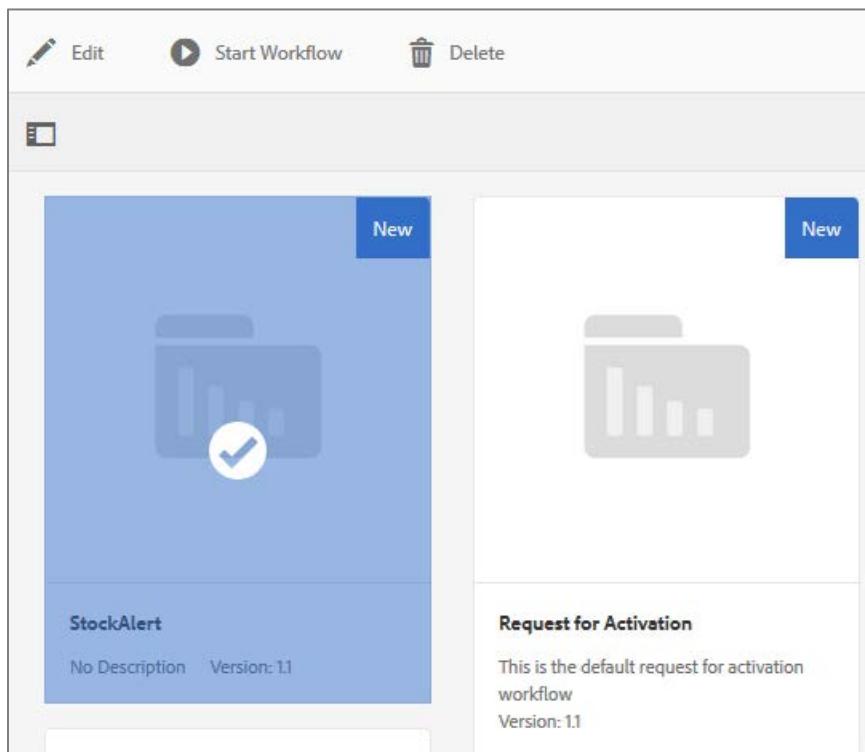
Title *

Name

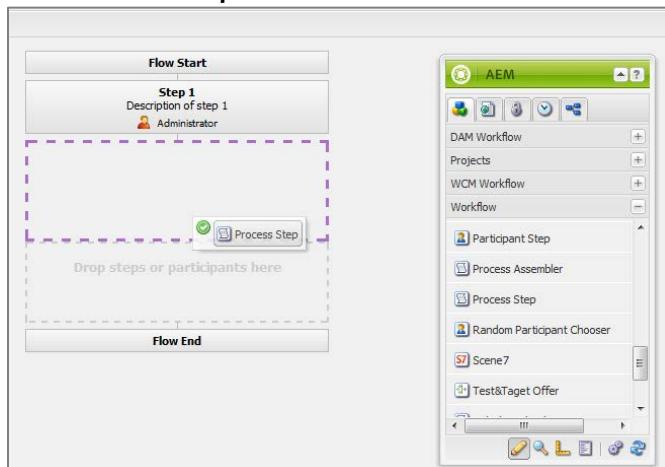
Cancel

Done

11. Select the newly created model **StockAlert**, click the checkmark, and then click **Edit**.

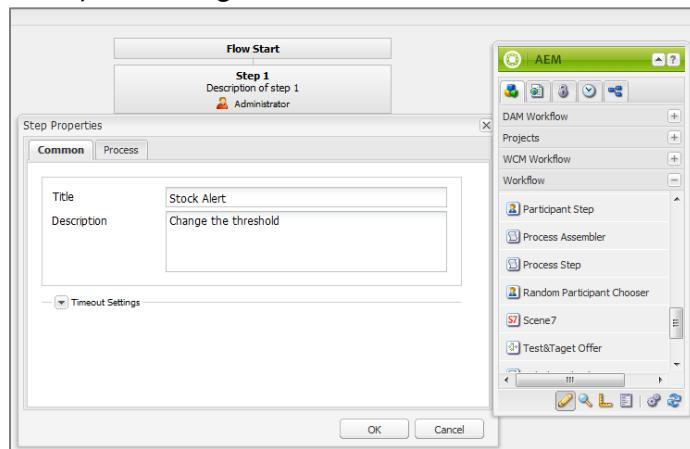


12. Delete the existing **Participant Step** from this workflow. This is not required as this workflow will be entirely automated.
13. Add a **Process Step** from **Workflow** section.



14. Double-click the **Process Step** and enter the following:

- Title: Stock Alert
- Description: Change the threshold



15. Click the **Process** tab and add:

- Process: Stock Threshold Checker
 - Handler Advance: Checked
 - Arguments: ADBE=120
- Click **OK** on the Step Properties dialog box, and then click **Save** (in the top left corner) to save the workflow Model.

16. Navigate to *Tools > Workflow > Launchers*, and click *Create > Add Launcher*.

Workflow Launchers							
Globbing	Description	Event Type	Nodetype	Condition	Workflow	Enabled	Exclude
/content/dam/*/*renditions/original	Parse Word documents (DOC) that have been added to the DAM	Node created	nt:file	jcr:content/jcr:mimeType==application/msword	DAM Parse Word Documents	true	false

17. Enter the following values:

- Event Type: Modified
- Nodetype: nt:unstructured
- Path: /content/ADBE/lastTrade
- Run Mode(s): Author

- Condition: leave blank
- Workflow: StockAlert
- Description: lastTrade launcher for StockAlert workflow
- Activate: Enable
- **Exclude List: Leave Blank**

The screenshot shows the 'Workflow Configuration' dialog box. The fields are as follows:

- Event Type *: Modified
- Nodetype *: nt:unstructured
- Path *: /content/ADBE/lastTrade
- Run Mode(s) *: Author
- Condition: (empty)
- Workflow: StockAlert
- Description: lastTrade launcher for StockAlert workflow
- Activate: Enable (checkbox checked)
- Exclude List: (empty)

a. Click Create.

18. Test the Launcher.



Note: To test the Launcher, you can either wait for the stock price to go above the threshold value, or you can manually change it to go above the threshold value in the JCR.

- In CRXDE Lite, go to the ADBE/lastTrade node and update the lastTrade property to be above the threshold value you set up in the launcher (120).

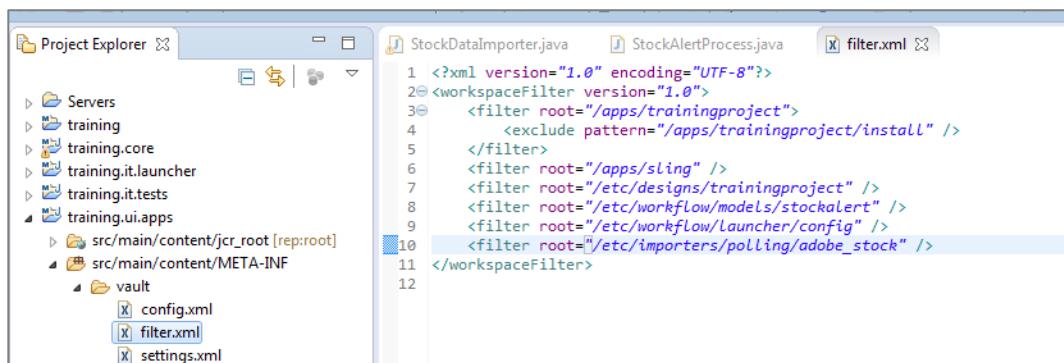
- Save.

19. Observe the log message output in the custom log file.

- In the previous steps, we created workflows and an importer on the AEM Server. We now need to import them into our Eclipse project
- Open Eclipse and update **filter.xml** located at *training.ui.apps>src>main>content>META-INF>vault* with the following code:

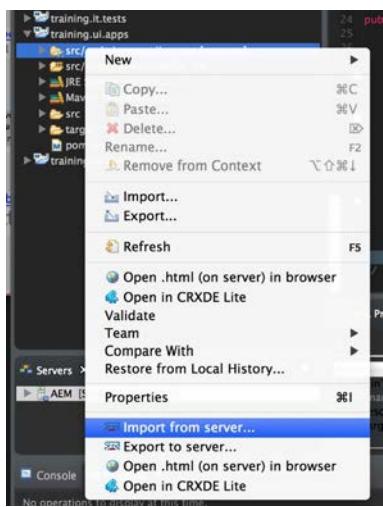
```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <workspaceFilter version="1.0">
3.   <filter root="/apps/trainingproject">
4.     <exclude pattern="/apps/trainingproject/install" />
5.   </filter>
6.   <filter root="/apps/sling" />
7.   <filter root="/etc/designs/trainingproject" />
8.   <filter root="/etc/workflow/models/StockAlert" />
9.   <filter root="/etc/workflow/launcher/config" />
10.  <filter root="/etc/importers/polling/adobe_stock" />
11. </workspaceFilter>
```



20. Import from server.

- a. In Eclipse, right click on src/main/content/jcr_root, and select "Import From Server".



Review:

In this module, you created an alert whenever the price of the stock crosses the limit price given in the workflow.

Exercise 6 – Programmatically create AEM Pages

Overview:

In this exercise, you will programmatically create a page based on pagecreator.jsp using servlet and slightly.

Steps:

1. Create a class named PageCreator.
 - a. Open Eclipse and navigate to `training.core > src/main/java > com.adobe.training.core.servlets`
 - b. Create a new class **PageCreator**



Note: The code is provided as part of the Exercise_Files under `/Exercise_Files/10_Deep_Dive_AEM_APIs/`. Copy and paste the code from the exercise files to Eclipse. Do not copy from this exercise book. The following is for illustration purposes only.

```
1. package com.adobe.training.core.servlets;
2.
3. import java.io.IOException;
4.
5. import javax.jcr.Session;
6. import javax.servlet.ServletException;
7.
8. import org.apache.felix.scr.annotations.Reference;
9. import org.apache.felix.scr.annotations.sling.SlingServlet;
10. import org.apache.sling.api.SlingHttpServletRequest;
11. import org.apache.sling.api.SlingHttpServletResponse;
12. import org.apache.sling.api.resource.Resource;
13. import org.apache.sling.api.servlets.SlingAllMethodsServlet;
14. import org.apache.sling.commons.json.JSONObject;
15. import org.slf4j.Logger;
16. import org.slf4j.LoggerFactory;
17.
18. import com.day.cq.replication.ReplicationActionType;
19. import com.day.cq.replication.Replicator;
20. import com.day.cq.tagging.Tag;
21. import com.day.cq.tagging.TagManager;
22. import com.day.cq.wcm.api.Page;
23. import com.day.cq.wcm.api.PageManager;
24.
25. /**
26. * This Servlet ingests a comma delimited string in the form of:
27. *
28. * New Page Path, Page Title, Page Tag, Auto Publish, Template
29. *
30. * And outputs the AEM page created.
31. *
32. * Make sure /apps/trainingproject/tools/pagecreator/pagecreator.html is added
33. * Test with
```

```

34. * http://localhost:4502/etc/trainingproject/pagecreator.html
35. *
36. *
37. * To allow for POST requests for this importer the OSGi config
38. * "Adobe Granite CSRF Filter" com.adobe.granite.csrf.impl.CSRFFilter
39. * Needs to be configured with:
40. * filter.excluded.paths=["/etc/trainingproject/pagecreator"]
41. *
42. * @author Kevin Nennig (nennig@adobe.com)
43. */
44.
45. @SlingServlet(resourceTypes="trainingproject/tools/pagecreator", methods="POST")
46. public class PageCreator extends SlingAllMethodsServlet{
47.     Logger logger = LoggerFactory.getLogger(this.getClass());
48.
49.     private static final long serialVersionUID = 1L;
50.
51.     @Reference
52.     private Replicator replicator;
53.
54.     private Resource resource;
55.
56.     public void doPost(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException{
57.         response.setHeader("Content-Type", "application/json");
58.         JSONObject resultObject = new JSONObject();
59.         resource = request.getResource();
60.
61.         String param = request.getParameter("importer");
62.         String[] nextPage = param.split(",");
63.         try {
64.             resultObject = createTrainingPage(nextPage[0], nextPage[1], nextPage[2], nextPage[3], nextPage[4]);
65.         } catch (Exception e) {
66.             logger.error("Failure to create page: " + e);
67.         }
68.
69.         if(resultObject != null){
70.             //Write the result to the page
71.             response.getWriter().print(resultObject.toString());
72.             response.getWriter().close();
73.         }
74.     }
75.
76.     /** Helper method to create the page based on available input
77.      *
78.      * @param path JCR location of the page to be created
79.      * @param title Page Title
80.      * @param template AEM Template this page should be created from. The
81.      * template must exist in the JCR already.
82.      * @param tag Tag must already be created in AEM. The tag will be in t
83.      * he form of a path. Ex /etc/tags/marketing/interest
84.      * @param publish boolean to publish the page
85.      * @return
86.      */
87.     private JSONObject createTrainingPage(String path, String title, String template, String tag, String publish) throws Exception{
88.         JSONObject pageInfo = new JSONObject();

```

```

89.         //Parse the path to get the pageNodeName and parentPath
90.         int lastSlash = path.lastIndexOf("/");
91.         String pageNodeName = path.substring(lastSlash+1);
92.         String parentPath = path.substring(0, lastSlash);
93.
94.         if(pageNodeName==null || parentPath==null) return null;
95.
96.         //Set a default template if none is given
97.         if(template == null || template.isEmpty()){
98.             template = "/apps/trainingproject/templates/page-content";
99.         }
100.
101.        //Create page
102.        PageManager pageManager = resource.getResourceResolver().adaptTo(P
ageManager.class);
103.        Page p = pageManager.create(parentPath,
104.                                     pageNodeName,
105.                                     template,
106.                                     title);
107.
108.        //Add a tag to the page
109.        if(tag != null && !tag.isEmpty()) {
110.            //TagManager can be retrieved via adaptTo
111.            TagManager tm = resource.getResourceResolver().adaptTo(TagMana
ger.class);
112.            tm.setTags(p.getContentResource(),
113.                        new Tag[]{tm.resolve(tag)},
114.                        true);
115.        }
116.
117.        //Publish page if requested
118.        boolean publishPage = Boolean.parseBoolean(publish);
119.        if(publishPage){
120.            //Replicator is exposed as a service
121.            replicator.replicate(resource.getResourceResolver().adaptTo(Se
ssion.class),
122.                                  ReplicationActionType.ACTIVATE,
123.                                  p.getPath());
124.        }
125.
126.        pageInfo.put("Status", "Successful");
127.        pageInfo.put("Location", p.getPath());
128.        pageInfo.put("Title", p.getTitle());
129.        pageInfo.put("Template Used", p.getTemplate().getPath());
130.        pageInfo.put("Tagged with", p.getTags()[0].getTitle());
131.        pageInfo.put("Was Published", publishPage);
132.        return pageInfo;
133.    }
134.
135.

```

- c. Examine the code and save the changes

2. Install the Bundle.
 - a. Right-click **training.core** and choose *Run As > Maven install*
 - b. Verify the build was successful
3. Modify the CSRF filter.
 - a. Observe the Granite Filter
 - b. Go to Web Console and Under *OSGi > Configuration*
 - c. Search for *Adobe Granite CSRF Filter*

The screenshot shows the configuration interface for the Adobe Granite CSRF Filter. At the top, it says "Request filter checking the CSRF token of modification requests." Below this, there are sections for "Filter Methods" (with POST, PUT, and DELETE selected), "Filter non-browser User Agents" (unchecked), and "Safe User Agents" (which lists various user agent strings). There are also sections for "Excluded Paths" and "Configuration Information" (showing Persistent Identity (PID) as com.adobe.granite.csrf.impl_CSRFFilter and Configuration Binding as launchpad:resources/install/0/com.adobe.granite.csrf-1.0.14.jar). At the bottom right are buttons for Cancel, Reset, Delete, Unbind, and Save.

- d. Examine the Filter methods and Safe User Agents

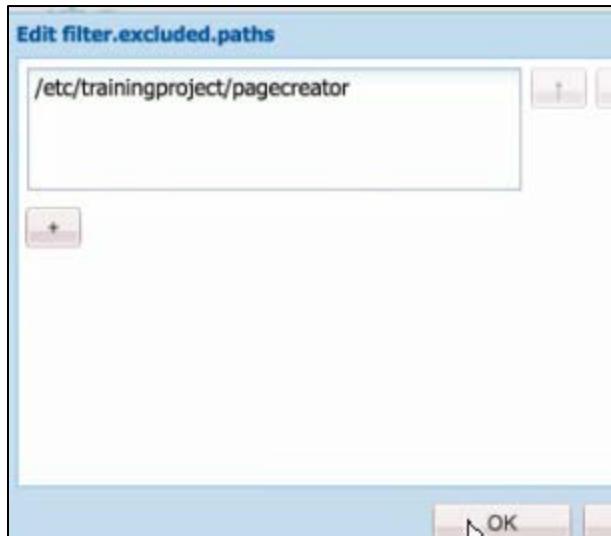


Note: In order for the pagecreator page to work, you have to add the path to the excluded path..

4. Perform the following steps to add the path to the excluded path in the CSRF filter.
 - a. In CRXDE Lite, navigate to the `/apps/trainingproject/config.author` node, which we created in an earlier task
 - b. Right-click **config.author** and choose *Create > Create Node..*
 - Name: **com.adobe.granite.csrf.impl.CSRFFilter**
 - Type: Sling:osgiConfig
 - c. Click **OK**
 - d. Add the following property:
 - Name: `filter.excluded.paths`
 - Type: **String**
 - Value: `/etc/trainingproject/pagecreator`

Properties					
		Access Control		Replication	
		Console		Build Info	
Name		Type	Value		Protected
1	jcr:primaryType	Name	sling:OsgiConfig		true
				Mandatory	Multiple
				false	
Name		Type	String	Value	/etc/trainingproject/pagecreator
				<input type="button" value="Multi"/>	

- e. Click the Multi button and then Add



- f. Click **OK** in the **Multi** window
- g. Save All

Note: Now we have the filter created which will allow you to post from /etc/trainingproject/pagecreator path.

5. Create a node Pagecreator.
 - a. Navigate to /etc and create a node
 - Name:trainingproject
 - Type:sling:Folder
 - b. Click **OK**
 - c. Right-click trainingproject and create a node as shown:
 - Name:pagecreator
 - Type:sling:Folder
 - Save All
6. Add a property to this node:
 - Name:sling:resourceType
 - Type:String
 - Value:trainingproject/tools/pagecreator
 - Click **Add** and Save All
7. Add a script under /apps/trainingproject/tools/pagecreator.
 - a. Create a folder named **tools** under /apps/trainingproject
 - b. Right-click tools and create a node
 - Name:pagecreator
 - Type:sling:Folder
 - c. Save All

8. In **pagecreator** folder, create a new file **pagecreator.html** with the following code:



Note: The code is provided as part of the Exercise_Files under /Exercise_Files/10_Deep_Dive_AEM_APIs/. Copy and paste the code from the exercise files to Eclipse. Do not copy from this exercise book. The following is for illustration purposes only.

```
1. <!--/*
2.      This should be located under /apps/trainingproject/tools/pagecreator/
3.
4.      A resource must be created:
5.          /etc/trainingproject/pagecreator
6.              + sling:resourceType="trainingproject/tools/pagecreator
7.      */-->
8. <h3>Input a comma delimited string for the page to be created</h3>
9. <table>
10.    <tr>
11.        <th>Page Path, </th>
12.        <th>Page Title, </th>
13.        <th>Page Template, </th>
14.        <th>Tag, </th>
15.        <th>Auto Publish?</th>
16.    </tr>
17. </table>
18. <br />
19. <br />
20. <form action="/etc/trainingproject/pagecreator.json" method="POST" enctype="mu
ltipart/form-data">
21.     <input style="width:60%;" name="importer" type="text" Value="/content/trai
ningproject/en/community,Our Community,/apps/trainingproject/templates/page-
home,/etc/tags/we-retail/activity/biking,TRUE">
22.     <input type="submit">
23. </form>
```

- Examine the input named imported in the html code.
- Also notice a preconfigured value is provided for you.

The importer parameter here correlates with the importer parameter in the pagecreator.java class.

- Click **Save All** in the upper-left corner.

9. Open <http://localhost:4502/etc/trainingproject/pagecreator.html> and you will see the following message:

localhost:4502/etc/trainingproject/pagecreator.html

Input a comma delimited string for the page to be created

Page Path, Page Title, Page Template, Tag, Auto Publish?

/content/trainingproject/en/community,Our Community,/apps/trainingproject/templates/page-home,/etc/tags/we-retail/activity/biking,TRUE

Submit Query

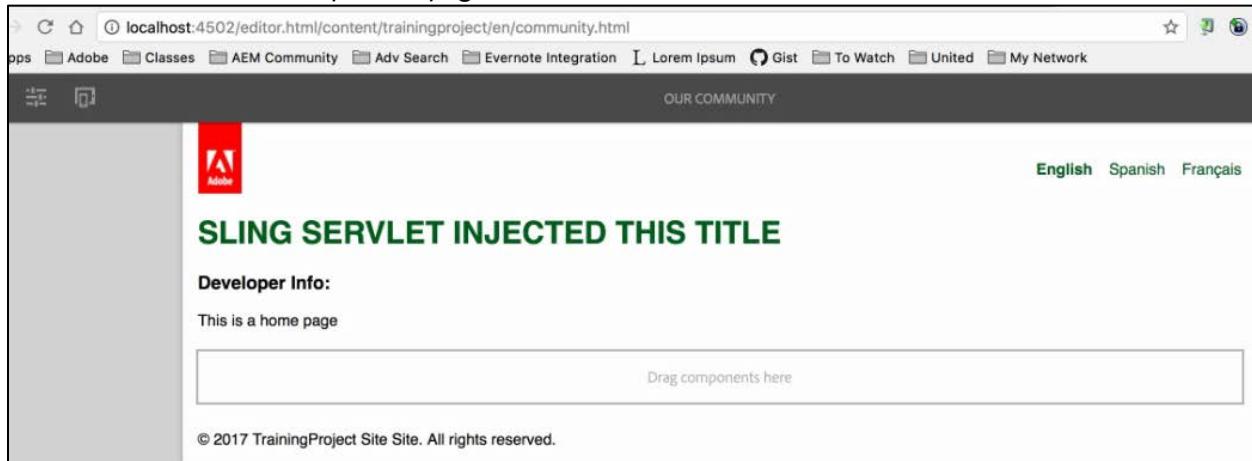
- a. Click Submit Query
- b. If it is successful, you will get a JSON reply as shown below:

```
localhost:4502/etc/trainingproject/pagecreator.json
```

```
{  
    "Status": "Successful",  
    "Location": "/content/trainingproject/en/community14",  
    "Title": "Our Community",  
    "Template Used": "/apps/trainingproject/templates/page-home",  
    "Tagged with": "Biking",  
    "Was Published": true  
}
```

10. Verify if the page is created successfully:

- a. Go to <http://localhost:4502/sites.html/content>
- b. Select *Training Project site >English > Our Community*
- c. Click Edit and open the page:



Exercise 7 – Programmatically create a Website

Overview:

In this exercise, you will programmatically create a website which shows how a site structure could be ingested into AEM using a csv file.

Steps:

1. Create a class named CSVPageCreator.
 - a. Open Eclipse and navigate to `training.core > src/main/java > com.adobe.training.core.servlets`
 - b. Create a new class **CSVPageCreator**

Note: The code is provided as part of the Exercise_Files under `/Exercise_Files/10_Deep_Dive_AEM_APIs/`. Copy and paste the code from the exercise files to Eclipse. Do not copy from this exercise book. The following is for illustration purposes only.

```
1. package com.adobe.training.core.servlets;
2.
3. import java.io.BufferedReader;
4. import java.io.ByteArrayInputStream;
5. import java.io.IOException;
6. import java.io.InputStream;
7. import java.io.InputStreamReader;
8.
9. import javax.jcr.Session;
10. import javax.servlet.ServletException;
11.
12. import org.apache.felix.scr.annotations.Reference;
13. import org.apache.felix.scr.annotations.sling.SlingServlet;
14. import org.apache.sling.api.SlingHttpServletRequest;
15. import org.apache.sling.api.SlingHttpServletResponse;
16. import org.apache.sling.api.resource.Resource;
17. import org.apache.sling.api.servlets.SlingAllMethodsServlet;
18. import org.apache.sling.commons.json.JSONException;
19. import org.apache.sling.commons.json.JSONObject;
20. import org.slf4j.Logger;
21. import org.slf4j.LoggerFactory;
22.
23. import com.day.cq.replication.ReplicationActionType;
24. import com.day.cq.replication.Replicator;
25. import com.day.cq.tagging.Tag;
26. import com.day.cq.tagging.TagManager;
27. import com.day.cq.wcm.api.Page;
28. import com.day.cq.wcm.api.PageManager;
29.
30. /**
31. * This Servlet ingests a .csv file with the following columns:
32. *
```

```

33. * New Page Path, Page Title, Page Tag, Auto Publish, Template
34. *
35. * And outputs AEM pages created.
36. *
37. * Make sure /apps/trainingproject/tools/pagecreator/csv.html is added
38. * Test with
39. * http://localhost:4502/etc/trainingproject/pagecreator.csv.html
40. *
41. * It is assumed there is no header to the csv file
42. *
43. * To allow for POST requests for this importer the OSGi config
44. * "Adobe Granite CSRF Filter" com.adobe.granite.csrf.impl.CSRFFilter
45. * Needs to be configured with:
46. * filter.excluded.paths=["/bin/trainingproject/pagecreator"]
47. *
48. * @author Kevin Nennig (nennig@adobe.com)
49. */
50.
51. @SlingServlet(resourceTypes="trainingproject/tools/pagecreator", selectors="csv", methods="POST")
52. public class CSVPageCreator extends SlingAllMethodsServlet{
53.
54.     private static final long serialVersionUID = 1L;
55.     Logger logger = LoggerFactory.getLogger(getClass());
56.
57.     @Reference
58.     private Replicator replicator;
59.
60.     private Resource resource;
61.
62.     public void doPost(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException{
63.         response.setHeader("Content-Type", "application/json");
64.         JSONObject resultObject = new JSONObject();
65.         resource = request.getResource();
66.
67.         String param = request.getParameter("importer");
68.         byte[] input = param.getBytes();
69.         InputStream stream = new ByteArrayInputStream(input);
70.         try {
71.             resultObject = readCSV(stream);
72.         } catch (JSONException e) {
73.             logger.error("Failure to Read CSV: " + e);
74.         }
75.
76.         if(resultObject != null){
77.             //Write the result to the page
78.             response.getWriter().print(resultObject.toString());
79.             response.getWriter().close();
80.         }
81.     }
82.
83.     /**
84.      * Reads the CSV file. The CSV file MUST be in the form of:
85.      *
86.      * JCR path, Page Title, Page Template, AEM Tag, Publish boolean
87.      *
88.      * @param stream Stream from the CSV
89.      * @return JSON object that contains the results of the page creation process
90.     */

```

```
91.     private JSONObject readCSV(InputStream stream) throws IOException, JSONException
92.     {
93.         JSONObject out = new JSONObject();
94.         BufferedReader br = new BufferedReader(new InputStreamReader(stream));
95.         if(br != null){
96.             String line;
97.             String[] nextPage;
98.             JSONObject createdPageObject = null;
99.             //Read each line of the CSV
100.            while ((line = br.readLine()) != null){
101.                nextPage = line.split(",");
102.                String aemTag = null;
103.                String publishFlag = null;
104.                String aemTemplatePath = null;
105.
106.                //If the line has a template, tag, publish flag, set those variables
107.                if(nextPage.length == 5){
108.                    aemTemplatePath = nextPage[2];
109.                    aemTag = nextPage[3];
110.                    publishFlag = nextPage[4];
111.                }else if(nextPage.length == 4){
112.                    aemTemplatePath = nextPage[2];
113.                    aemTag = nextPage[3];
114.                }else if(nextPage.length == 3){
115.                    publishFlag = nextPage[2];
116.                }
117.
118.                //As long as there is a path and title, the page can be created
119.                if((nextPage.length > 1)
120.                   && !nextPage[0].isEmpty()
121.                   && !nextPage[1].isEmpty()){
122.                    String path = nextPage[0];
123.                    String title = nextPage[1];
124.                    try {
125.                        createdPageObject = createTrainingPage(path, title, aemTemplatePath, aemTag, publishFlag);
126.                    } catch (Exception e) {
127.                        logger.error(path + " not created successfully: " + e);
128.                    }
129.
130.                    //add the status of the row into the json array
131.                    if(createdPageObject != null){
132.                        out.put(path, createdPageObject); //Print Title of Page
133.                        createdPageObject = null;
134.                    }else{
135.                        out.put(path, new JSONObject().put("Status", "Could not create
136.                            a page"));
137.                    }
138.                }else {
139.                    out.put(line, new JSONObject().put("Status", "Could not properly p
140.                        arce"));
141.                }
142.            }
143.            br.close();
144.            return out;
145.        }
146.
```

```

147.    /** Helper method to create the page based on available input
148.     *
149.     * @param path JCR location of the page to be created
150.     * @param title Page Title
151.     * @param template AEM Template this page should be created from. The template mu
152.     * st exist in the JCR already.
153.     * @param tag Tag must already be created in AEM. The tag will be in the form of
154.     * a path. Ex /etc/tags/marketing/interest
155.     * @return
156.     */
157.    private JSONObject createTrainingPage(String path, String title, String template,
158.                                         String tag, String publish) throws Exception{
159.        JSONObject pageInfo = new JSONObject();
160.
161.        //Parse the path to get the pageNodeName and parentPath
162.        int lastSlash = path.lastIndexOf("/");
163.        String pageNodeName = path.substring(lastSlash+1);
164.        String parentPath = path.substring(0, lastSlash);
165.
166.        if(pageNodeName==null || parentPath==null) return null;
167.
168.        //Set a default template if none is given
169.        if(template == null || template.isEmpty()){
170.            template = "/apps/trainingproject/templates/page-content";
171.        }
172.
173.        //Create page
174.        PageManager pageManager = resource.getResourceResolver().adaptTo(PageManager.
175. class);
176.        Page p = pageManager.create(parentPath,
177.                                     pageNodeName,
178.                                     template,
179.                                     title);
180.        //Add a tag to the page
181.        if(tag != null && !tag.isEmpty()) {
182.            //TagManager can be retrieved via adaptTo
183.            TagManager tm = resource.getResourceResolver().adaptTo(TagManager.class);

184.            tm.setTags(p.getContentResource(),
185.                       new Tag[]{tm.resolve(tag)},
186.                       true);
187.
188.            //Publish page if requested
189.            boolean publishPage = Boolean.parseBoolean(publish);
190.            if(publishPage){
191.                //Replicator is exposed as a service
192.                replicator.replicate(resource.getResourceResolver().adaptTo(Session.class
193. ),
194.                               ReplicationActionType.ACTIVATE,
195.                               p.getPath());
196.
197.                pageInfo.put("Status", "Successful");
198.                pageInfo.put("Location", p.getPath());
199.                pageInfo.put("Title", p.getTitle());
200.                pageInfo.put("Template Used", p.getTemplate().getPath());
201.                pageInfo.put("Tagged with", p.getTags()[0].getTitle());

```

```
202.         pageInfo.put("Was Published", publishPage);
203.         return pageInfo;
204.     }
205.
206. }
```

- a. Examine the code. Notice the method `createTrainingPage` is similar to the method in the `pagecreator`. The only difference is the input which is in CSV.
 - b. Save the changes
2. Install the Bundle.
 - a. Right-click **training.core** and choose Run As > Maven install
 - b. Verify the build was successful
 3. Add a new script to the `pagecreator`.
 - a. In CRXDE Lite, navigate to `/apps/trainingproject/tools/pagecreator`
 - b. Create a file named **csv.html**
 - c. Add the code from `/Exercise_Files/10_Deep_Dive_AEM_APIs/csv.html`

```
1. <!--/*
2.      This should be located under /apps/trainingproject/tools/pagecreator/
3.
4.      A resource must be created:
5.      /etc/trainingproject/pagecreator
6.          + sling:resourceType="trainingproject/tools/pagecreator"
7.  *!-->
8.  <h3>Upload a csv list of pages to be created</h3>
9.  <table>
10. <tr>
11.   <th>Page Path, </th>
12.   <th>Page Title, </th>
13.   <th>Page Template, </th>
14.   <th>Tag, </th>
15.   <th>Auto Publish?</th>
16. </tr>
17. </table>
18. <br />
19. <br />
20. <br />
21. <form action="/etc/trainingproject/pagecreator.csv.json" method="POST" enctype="multi
   part/form-data">
22.   <input name="importer" type="file" accept=".csv">
23.   <input type="submit">
24. </form>
```

- d. Examine the code. Notice the only difference from the previous html is here we accept the .csv files for input.
 - e. Save All
5. Verify the page is rendered successfully
- a. Go to <http://localhost:4502/etc/trainingproject/pagecreator.csv.html>

The screenshot shows a web browser window with the URL <http://localhost:4502/etc/trainingproject/pagecreator.csv.html> in the address bar. The page title is "Upload a csv list of pages to be created". Below the title, there is a question: "Page Path, Page Title, Page Template, Tag, Auto Publish?". At the bottom left is a "Choose File" button with the text "No file chosen" next to it. At the bottom right is a "Submit" button.

6. Test with two different files.

For instance , the first file PageCreator-badData.csv does not have a location for the page and it will fail for this reason.

- a. Click **Choose File** and select /Exercise_Files/10_Deep_Dive_AEM_APIs/PageCreator-badData.csv
- b. Click **Submit Query**

```
▼ {
    ▼ "/content/trainingproject/en/products": {
        "Status": "Could not create a page"
    },
    ▼ "/content/trainingproject/en/products/category1": {
        "Status": "Could not create a page"
    },
    ▼ "/content/trainingproject/en/products/category1/prod1": {
        "Status": "Could not create a page"
    },
    ▼ "/content/trainingproject/en/products/category1/prod1/overview": {
        "Status": "Could not create a page"
    },
    ▼ "/content/trainingproject/en/products/category1/prod1/specs": {
        "Status": "Could not create a page"
    },
    ▼ ",Product 2,,/etc/tags/marketing/interest/product,TRUE": {
        "Status": "Could not properly parce"
    },
    ▼ "/content/trainingproject/en/products/category1/prod2/overview": {
        "Status": "Could not create a page"
    },
    ▼ "/content/trainingproject/en/products/category1/prod2/specs": {
        "Status": "Could not create a page"
    },
    ▼ "/content/trainingproject/en/products/category1/prod3": {
        "Status": "Could not create a page"
    },
    ▼ "/content/trainingproject/en/products/category1/prod3/overview": {
        "Status": "Could not create a page"
    },
    ▼ "/content/trainingproject/en/products/category1/prod3/specs": {
        "Status": "Could not create a page"
    },
    ▼ "/content/trainingproject/en/products/category1/prod4": {
        "Status": "Could not create a page"
    },
    ▼ "/content/trainingproject/en/products/category1/prod4/overview": {
        "Status": "Could not create a page"
    },
    ▼ ",Specifications,,/etc/tags/marketing/interest/product,": {
        "Status": "Could not properly parce"
    }
}
```

- c. Examine the error returned due to bad data in the CSV file. Also, notice some of the pages are created.

7. Test with good csv file.

- a. Go to <http://localhost:4502/etc/trainingproject/pagecreator.csv.html>
- b. Browse and select /Exercise_Files/10_Deep_Dive_AEM_APIs/PageCreator.csv
- c. Click Submit
- d. Verify all pages are successfully created:



The screenshot shows a browser window displaying the JSON response from the AEM API. The URL in the address bar is `localhost:4502/etc/trainingproject/pagecreator.csv.json`. The JSON object contains several entries, each representing a successful creation of a page or component. The keys in the JSON are paths starting with `/content/trainingproject/en/products`, followed by specific category names like `women`, `coats`, `raincoat`, `insulated`, and `shirts`. Each entry includes fields for Status (Successful), Location (the path where the page was created), Title (the name of the page), Template Used (the template used for creation), Tagged with (tags applied to the page), and Was Published (whether the page was published).

```
{  
    "/content/trainingproject/en/products": {  
        "Status": "Successful",  
        "Location": "/content/trainingproject/en/products3",  
        "Title": "Products",  
        "Template Used": "/apps/trainingproject/templates/page-home",  
        "Tagged with": "Apparel",  
        "Was Published": true  
    },  
    "/content/trainingproject/en/products/women": {  
        "Status": "Successful",  
        "Location": "/content/trainingproject/en/products/women",  
        "Title": "Women",  
        "Template Used": "/apps/trainingproject/templates/page-content",  
        "Tagged with": "Women",  
        "Was Published": true  
    },  
    "/content/trainingproject/en/products/women/coats": {  
        "Status": "Successful",  
        "Location": "/content/trainingproject/en/products/women/coats",  
        "Title": "Coats",  
        "Template Used": "/apps/trainingproject/templates/page-content",  
        "Tagged with": "Coat",  
        "Was Published": true  
    },  
    "/content/trainingproject/en/products/women/coats/raincoat": {  
        "Status": "Successful",  
        "Location": "/content/trainingproject/en/products/women/coats/raincoat",  
        "Title": "Rain Coats",  
        "Template Used": "/apps/trainingproject/templates/page-content",  
        "Tagged with": "Coat",  
        "Was Published": false  
    },  
    "/content/trainingproject/en/products/women/coats/insulated": {  
        "Status": "Successful",  
        "Location": "/content/trainingproject/en/products/women/coats/insulated",  
        "Title": "Insulated Coats",  
        "Template Used": "/apps/trainingproject/templates/page-content",  
        "Tagged with": "Coat",  
        "Was Published": false  
    },  
    "/content/trainingproject/en/products/women/shirts": {  
        "Status": "Successful",  
        "Location": "/content/trainingproject/en/products/women/shirts",  
        "Title": "Shirts",  
        "Template Used": "/apps/trainingproject/templates/page-content",  
        "Tagged with": "Shirt",  
        "Was Published": true  
    }  
}
```

Note: When you are done testing this lab, make sure to delete the node `com.adobe.granite.csrf.impl.CSRFFilter` that you created in Exercise 6 of this lab. This has to be deleted in order for the curl lab in Writing Tests module to work.

Review:

In this exercise, you did the following:

- Programmatically created AEM pages
- Programmatically created a website

11 WRITING TESTS

Overview

This module looks at the available testing frameworks such as Maven, Mockito, Sling JUnit, and Hobbes tests on your project. This testing chapter is based on how the AEM Archetype approaches testing in a real project.

Objectives

By the end of this module, you will be able to:

- Run unit tests and functional tests on your project

Understanding Testing Frameworks

Software testing is done to ensure that the system performs as expected. The results of a test are compared with the expected outcomes to validate the functionalities of the system. Some of these tests can be extensive and repetitive. In such cases, you can use testing frameworks to automate the testing process.

A testing framework is a system with a set of rules for automation of software testing. This system makes use of function libraries, test data sources, object details, and various reusable modules.

The Mockito Framework

Mockito is an open source testing framework that allows the creation of mock objects in automated unit tests. Mock testing frameworks effectively fake some external dependencies so that the object being tested has a consistent interaction with its outside dependencies. Mockito intends to streamline the delivery of these external dependencies that are not subjects of the test.

Features of Mockito:

- Mocks concrete classes as well as interfaces
- Verification errors are clean—click on stack trace to see failed verification in test; click on exception's cause to navigate to actual interaction in code. Stack trace is always clean.
- Allows flexible verification in order
- Supports exact-number-of-times and at-least-once verification
- Flexible verification or stubbing using argument matchers
- Allows creating custom argument matchers or using existing hamcrest matchers

Performing Unit Tests

In this chapter, you will look at the various unit tests that can be performed with Adobe Experience Manager. Unit testing is the process where the application is divided into units, and each unit is individually tested for its successful operation.

Writing Sling Tests

You can use Sling to perform a number of tests, including:

- JUnit tests using OSGi bundles in an OSGi system.
- Scriptable tests in a Sling instance, using any supported scripting language.
- Integration tests against a Sling instance that is started during the Maven build cycle or independently.

Performing Sling-based Tests on the Server

To perform a Sling test, you need the `org.apache.sling.junit.core` bundle. This bundle provides a service that allows bundles to register JUnit tests, and these tests are executed on the server by the `JUnitServlet` registered by default at `/system/sling/junit`. You should also note that this bundle is independent of Sling, and can work in other OSGi environments as well.

Performing Sling Scriptable Tests

To perform a Sling scriptable test, in addition to the above mentioned bundle, you also need the `org.apache.sling.junit.scriptable` bundle. While executing these tests, you need to make note of the following:

- A node with the `sling:Test` mixin is a scriptable test node.
- Scriptable test nodes are executed only if they belong to the Sling's `ResourceResolver` search path. For example, `/libs` or `/apps`.

To learn more about Sling Testing, visit <https://sling.apache.org/documentation/development/sling-testing-tools.html>



NOTE: To execute a test, the scriptable tests provider makes an HTTP request to the test node's path, with a `.test.txt` selector and extension, and expects the output to contain only the string `TEST_PASSED`. Empty lines and comment lines starting with a hash sign (#) are ignored in the output, and other lines are reported as failures.

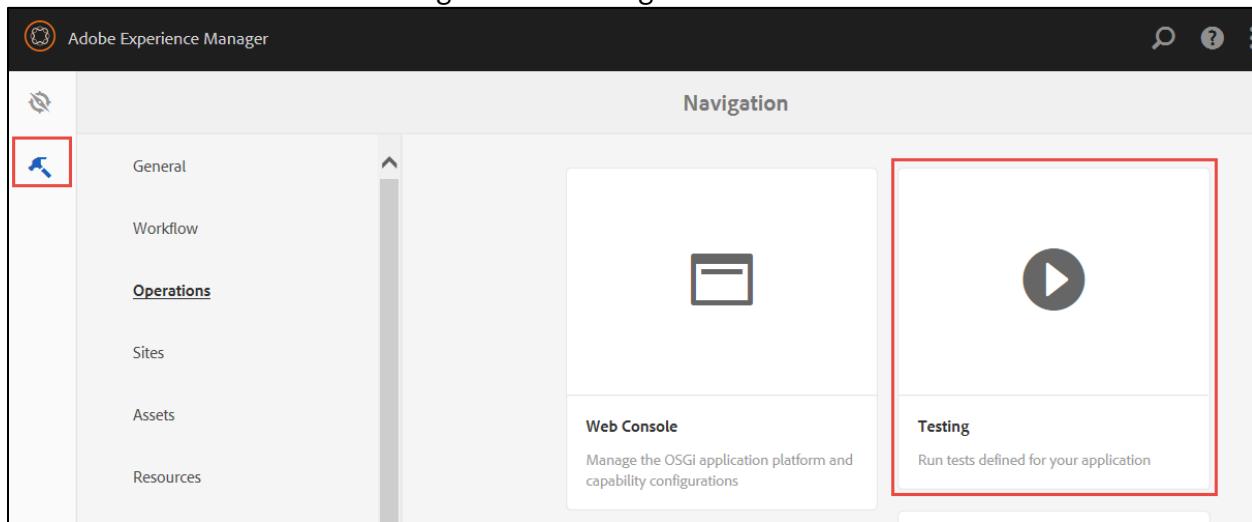
Performing Functional Tests

While unit tests check each small unit of an application, functional tests are more involved in evaluating whether the web application performs according to the business requirements. These requirements may be in the form of functional specifications or design specifications.

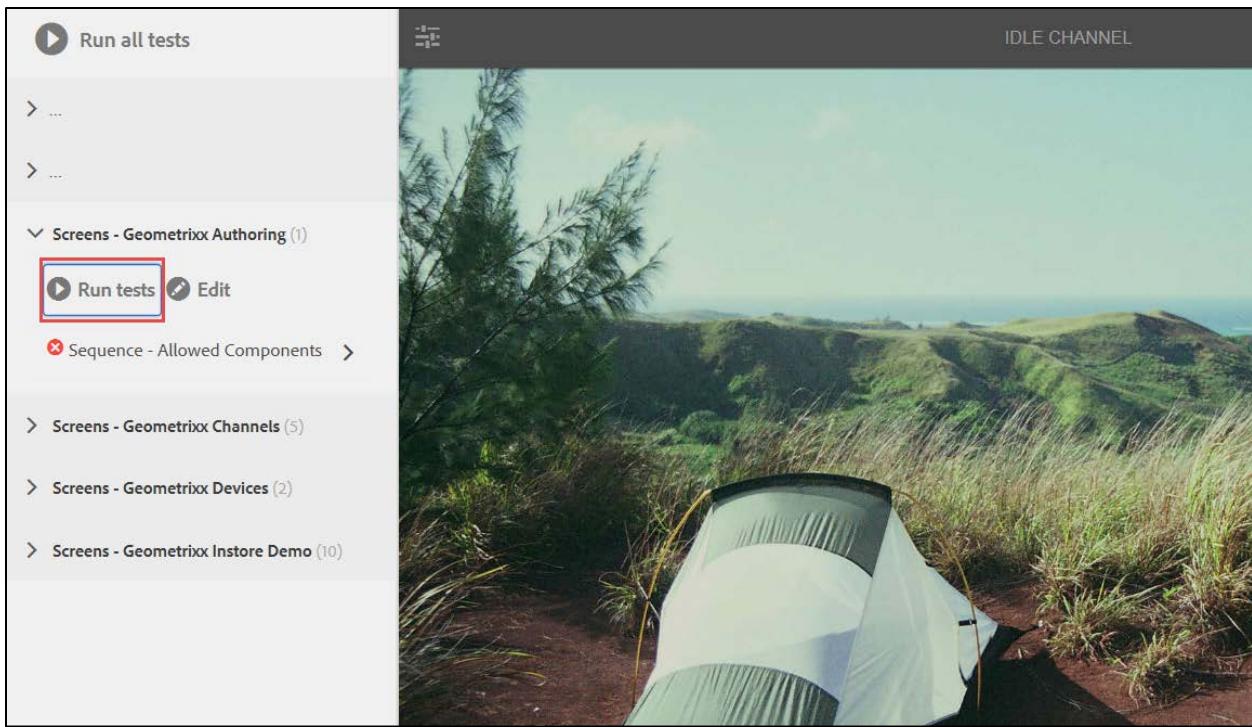
Functional testing does not involve testing of back-end code, but rather the front-end is tested to check if it reacts properly to the user input. Using Test-Driven Development (TDD), you will write a test first, then write the simplest code possible to implement the desired functionality. You will then refactor the code to meet production standards.

Performing Hobbes Tests

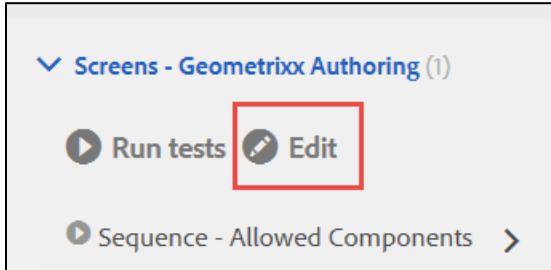
You can access this framework through Tools > Testing in AEM:



In a new browser tab, you can perform Hobbes testing on a particular screen (site) or run all test suites. You can select a test suite and run a test case within it:



You can also edit the test cases in a new browser tab by clicking the pencil icon in the toolbar below a test suite. This opens the corresponding code file in CRXDE Lite.



Jenkins for Continuous Integration

Continuous integration is a process where all development work is integrated to a system at a predefined time, and is automatically tested and built. The purpose of using continuous integration is to identify and catch errors early in the process.

Jenkins is an open source continuous integration tool used for build automation. Its main functionality is to execute a predefined set of steps based on a trigger such as a change in version control system, or a time based trigger such as creating a build every 30 minutes.

The steps to be executed could be any of the following:

- Perform a software build with Apache Maven.
- Run a shell script.
- Archive the build result.
- Start the integration tests.

With Jenkins, you can:

- Monitor the execution of steps.
- Stop the process if any of the steps fail.
- Notify respective users of the status of the build, whether the build is a success or failure.

If you are working on an Adobe Experience Manager project, here is how you can use Jenkins:

Work on the feature or bug fix.

Test it on your local instance.

Commit or push the changes to the central repository; for example, SVN or GIT.

Have Jenkins configured to kick start the build production, and deploy packages to the Adobe Experience Manager Integration Server.

Move the feature/bug-fix to the Quality Assurance (QA) team.

QA team will pick up the feature/bug-fix, and test code changes on the Integration Server.



NOTE: If the build is not successful, Jenkins can identify the faulty source code that was checked in to the repository, and then notify the developer of the error. This needs to be fixed before the next build process can take place.

LAB J – Writing Tests

Scenario

You are at a stage when the work that you have done in your project needs to be tested before you can push them to the central repository; both in terms of unit tests, as well as functional tests.

Challenge

Identifying the appropriate test framework.

Overview

- Write Unit tests using Mockito
- Create server-side Sling junit test
- Create scriptable server-side tests

Prerequisites

- AEM installed on your machine:
 - Running Adobe Experience Manager Author instance
 - An Eclipse project from the AEM Archetype
 - The Eclipse project should contain StockModel.java

Directions

Complete the exercises that follow

Exercise 1 – Create Unit Tests using Mockito

Overview

In this lab exercise, you will create a unit test and test StockModel.java class you created previously. This is a basic unit Test outside the server.

Steps

1. Create a class named StockModelMockitoTest to test StockModel.java.
 - a. In Eclipse and navigate to training.core > src/test/java > com.adobe.training.core.models
 - b. Create a new class StockModelMockitoTest



NOTE: The code is provided as part of the Exercise_Files under /Exercise_Files/11_Writing_Tests/. Copy and paste the code from the exercise files to Eclipse.

```
1. package com.adobe.training.core.models;
2.
3. import static org.junit.Assert.assertFalse;
4. import static org.junit.Assert.assertNotNull;
5. import static org.junit.Assert.assertTrue;
6. import static org.mockito.Mockito.mock;
7. import static org.mockito.Mockito.when;
8.
9. import java.text.SimpleDateFormat;
10. import java.util.Calendar;
11. import java.util.Date;
12. import java.util.Random;
13.
14. import org.apache.sling.api.resource.Resource;
15. import org.junit.Before;
16. import org.junit.Test;
17.
18. /**
19. * Tests the StockModel using the Mockito testing framework
20. * Note that the testable class is under /src/main/java:
21. * com.adobe.training.core.models.StockModel.java
22. *
23. * To correctly use this testing class:
```

```
24. * -  
    put this file under training.core/src/test/java in the package com.adobe.training.  
    core.models  
25. *  
26. */  
27. public class StockModelMockitoTest {  
28.  
29.     private StockModel stock;  
30.  
31.     @Before  
32.     public void setup() throws Exception {  
33.  
34.         //Adapt the Resource if needed  
35.         Resource RESOURCE_MOCK = mock(Resource.class);  
36.         StockModel STOCKMODEL_MOCK = mock(StockModel.class);  
37.         when(RESOURCE_MOCK.adaptTo(StockModel.class)).thenReturn(STOCK  
            MODEL_MOCK);  
38.  
39.         stock = STOCKMODEL_MOCK;  
40.  
41.         //Setup lastTrade Property  
42.         Random rand = new Random();  
43.         double n = Math.round(100*(rand.nextInt(150) + 100)+rand.nextDouble())/  
            100; //random value between 100.00 and 150.00  
44.         when(STOCKMODEL_MOCK.getLastTrade()).thenReturn(n);  
45.  
46.  
47.         //Setup requestDate Property  
48.         int numberOfDays = randBetween(1,365);  
49.         Date date = new SimpleDateFormat("D").parse(numberOfDays + " " + Calen  
            dar.getInstance().get(Calendar.YEAR));  
50.         String tradeDate = new SimpleDateFormat("MM/dd/yyyy").format(date);  
51.         when(STOCKMODEL_MOCK.getRequestDate()).thenReturn(tradeDate);  
52.  
53.         //Setup requestTime Property  
54.         int hours = randBetween(1, 12);  
55.         int minutes = randBetween(1, 60);  
56.         String time = hours+":"+minutes+"pm";  
57.         when(STOCKMODEL_MOCK.getRequestTime()).thenReturn(time);  
58.     }  
59.  
60.     private static int randBetween(int start, int end) {  
61.         return start + (int)Math.round(Math.random() * (end - start));  
62.     }  
63.  
64.     @Test
```

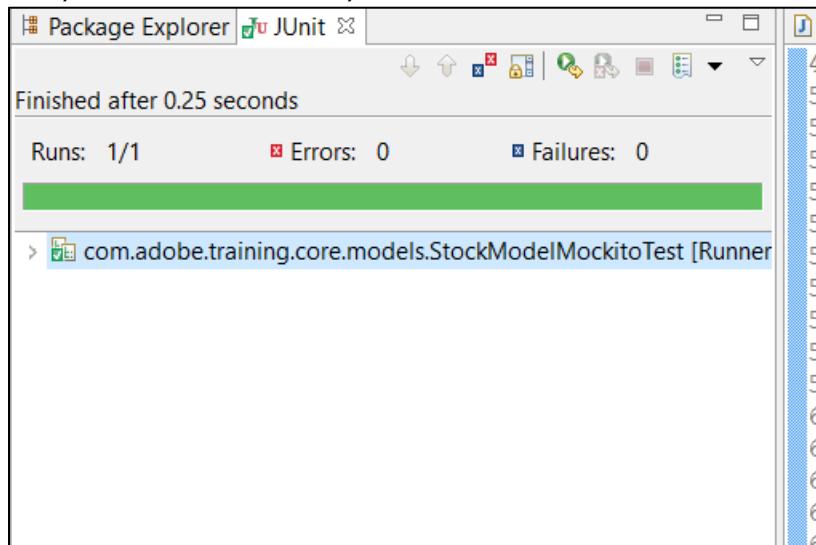
```
65. public void testGetLastTradeValue() throws Exception{  
66.     assertNotNull("lastTradeModel is null", stock);  
67.     assertTrue("lastTrade value is incorrect", stock.getLastTrade() > 100);  
68.     assertFalse("requestDate value is incorrect", stock.getRequestDate().isEmpty());  
69.     assertFalse("requestTime value is incorrect", stock.getRequestTime().isEmpty());  
70. }  
71.  
72. }
```

c. Examine the code

This class will test the java class StockModel.java

2. Perform a single unit test using JUnit .

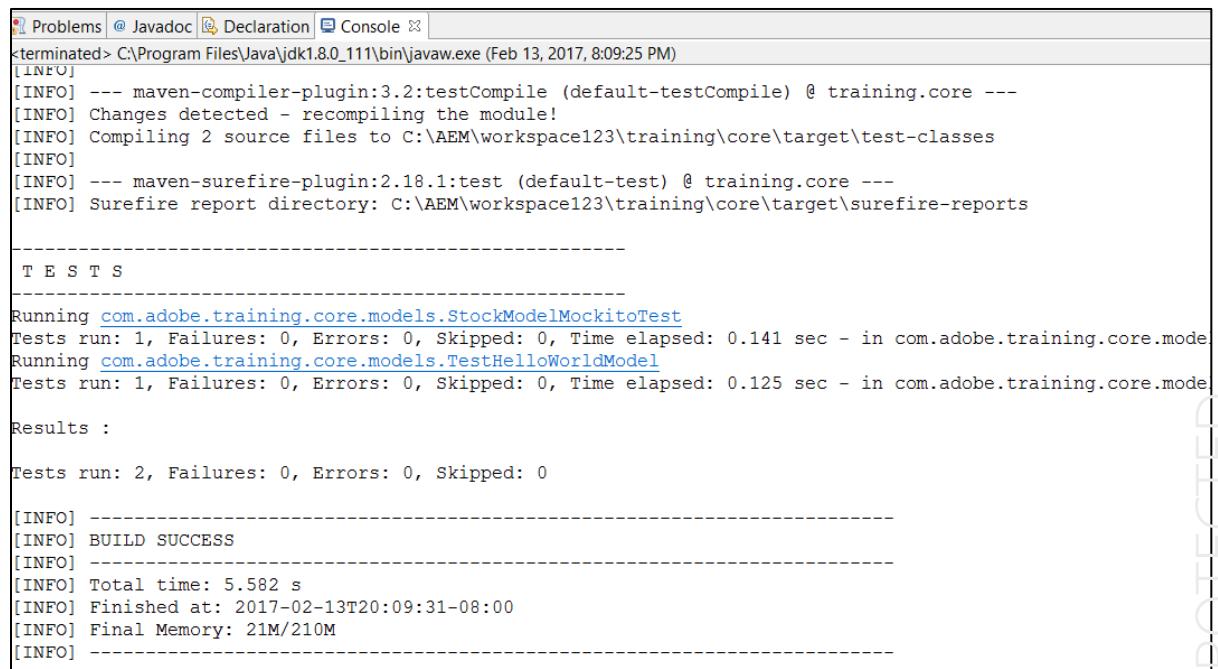
- Right-click on **StockModelMockitoTest.java** and select *Run As > Junit Test*
- Verify the test ran successfully:



3. Alternatively, you can run all tests under src/test/java by using Maven.

- Right-click training.core and select *Run AS > Maven Test*

b. Verify Success:



The screenshot shows a Java IDE's terminal or console window. The title bar includes tabs for 'Problems', '@ Javadoc', 'Declaration', and 'Console'. The console output is as follows:

```
<terminated> C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Feb 13, 2017, 8:09:25 PM)
[INFO] --- maven-compiler-plugin:3.2:testCompile (default-testCompile) @ training.core ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 2 source files to C:\AEM\workspace123\training\core\target\test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.18.1:test (default-test) @ training.core ---
[INFO] Surefire report directory: C:\AEM\workspace123\training\core\target\surefire-reports

-----
T E S T S
-----
Running com.adobe.training.core.models.StockModelMockitoTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.141 sec - in com.adobe.training.core.mode
Running com.adobe.training.core.models.TestHelloWorldModel
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.125 sec - in com.adobe.training.core.mode

Results :

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.582 s
[INFO] Finished at: 2017-02-13T20:09:31-08:00
[INFO] Final Memory: 21M/210M
[INFO] -----
```

Exercise 2 – Create Server-side Sling JUnit Test

Overview

In this lab exercise, you will perform a server-side unit test using the Sling JUnit testing framework and test StockModel java class in you created previously.

Steps

1. Create Server-side test.

- In Eclipse, navigate to training.it.tests > src/main/java > com.adobe.training.it.tests



Note: There is a java class named HelloWorldModelServerSideTest.java. This class will only work on version 6.1. Delete this class HelloWorldModelServerSideTest.java.

- Right click HelloWorldModelServerSideTest.java and choose Delete

- Create a new class StockModelServerSideTest.java

- Right-click **com.adobe.training.it.tests** and choose *New > class*
- Name: StockModelServerSideTest



Note: The code is provided as part of the Exercise_Files under /Exercise_Files/11_Writing_Tests/. Copy and paste the code from the exercise files to Eclipse.

```
1.  
2. * Copyright 2015 Adobe Systems Incorporated  
3. *  
4. * Licensed under the Apache License, Version 2.0 (the "License");  
5. * you may not use this file except in compliance with the License.  
6. * You may obtain a copy of the License at  
7. *  
8. *   http://www.apache.org/licenses/LICENSE-2.0  
9. *  
10. * Unless required by applicable law or agreed to in writing, software  
11. * distributed under the License is distributed on an "AS IS" BASIS,  
12. * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

```
13. * See the License for the specific language governing permissions and
14. * limitations under the License.
15. */
16. package com.adobe.training.it.tests;
17.
18. import static org.junit.Assert.assertNotNull;
19. import static org.junit.Assert.assertTrue;
20.
21. import java.util.HashMap;
22. import java.util.Map;
23. import java.util.concurrent.Callable;
24.
25. import org.apache.sling.api.resource.Resource;
26. import org.apache.sling.api.resource.ResourceResolver;
27. import org.apache.sling.api.resource.ResourceResolverFactory;
28. import org.apache.sling.junit.annotations.SlingAnnotationsTestRunner;
29. import org.apache.sling.junit.annotations.TestReference;
30. import org.junit.After;
31. import org.junit.Before;
32. import org.junit.Test;
33. import org.junit.runner.RunWith;
34.
35. import com.adobe.training.core.models.StockModel;
36.
37.
38. /**
39. * Test case which uses OSGi services injection
40. * to get hold of the StockModelServerSideTest which
41. * it wants to test server-side.
42. *
43. * Based on: https://github.com/Adobe-Marketing-Cloud/aem-project-archetype/blob/master/src/main/archetype/it.tests/src/main/java/it/tests/HelloWorldModelServerSideTest.java
44. *
45. * Example URI: http://localhost:4502/system/sling/junit/com.adobe.training.html
46. *
47. * To correctly use this testing class:
48. * -put this file under training.it.tests/src/main/java
49. * -Delete HelloWorldModelServerSideTest.java
50. * -On training.it.tests run: mvn clean install
51. * -On training.it.launcher run: mvn install -P integrationTests
52. */
53. @RunWith(SlingAnnotationsTestRunner.class)
54. public class StockModelServerSideTest {
55.
56.     @TestReference
```

```
57. private ResourceResolverFactory rrf;
58.
59. @Before
60. public void prepareData() throws Exception {
61.     new AdminResolverCallable() {
62.         @Override
63.         protected void call0(ResourceResolver rr) throws Exception {
64.             Map<String, Object> properties = new HashMap<String, Object>();
65.             properties.put("lastTrade", 109);
66.             properties.put("requestDate", "11/13/2016");
67.             properties.put("requestTime", "5:00pm");
68.             properties.put("upDown", .13);
69.             properties.put("openPrice", 105);
70.             properties.put("rangeHigh", 110);
71.             properties.put("rangeLow", 104);
72.             properties.put("volume", 2131988);
73.             Resource parent = rr.create(rr.getResource("/tmp"), "TEST", null);
74.             rr.create(parent, "lastTrade", properties);
75.         }
76.     }.call();
77. }
78.
79. @After
80. public void cleanupData() throws Exception {
81.     new AdminResolverCallable() {
82.         @Override
83.         protected void call0(ResourceResolver rr) throws Exception {
84.             Resource testResource = rr.getResource("/tmp/TEST");
85.             if ( testResource != null ) {
86.                 rr.delete(testResource);
87.             }
88.         }
89.     }.call();
90. }
91.
92. @Test
93. public void testStockModelServerSide() throws Exception {
94.
95.     assertNotNull("Expecting the ResourceResolverFactory to be injected by Sling test runner",
rrf);
96.
97.     new AdminResolverCallable() {
98.         @Override
99.         protected void call0(ResourceResolver rr) throws Exception {
100.             Resource testResource = rr.getResource("/tmp/TEST");
101. }
```

```
102.     StockModel stock = testResource.adaptTo(StockModel.class);
103.
104.     assertNotNull("Expecting LastTradeModel to be adapted from Resource", stock);
105.     assertNotNull("Expecting LastTradeModel to have the last Trade", stock.getLastTrade()
106. );
107.     assertTrue("lastTrade property incorrect",stock.getLastTrade() == 109);
108.     assertTrue("requestDate property incorrect",stock.getRequestDate().equals("11/13/2016"
109. ));
110.     assertTrue("requestTime property incorrect",stock.getRequestTime().equals("5:00pm")
111. );
112.     assertTrue("timestamp property incorrect",stock.getTimestamp().equals(stock.getRequestDate() + " +stock.getRequestTime()));
113.     assertTrue("upDown property incorrect",stock.getUpDown() == .13);
114.     assertTrue("openPrice property incorrect",stock.getOpenPrice() == 105);
115.     assertTrue("rangeHigh property incorrect",stock.getRangeHigh() == 110);
116.     assertTrue("rangeLow property incorrect",stock.getRangeLow() == 104);
117.     assertTrue("volume property incorrect",stock.getVolume() == 2131988);
118. }
119.
120. private abstract class AdminResolverCallable implements Callable<Void> {
121.
122.     @Override
123.     public Void call() throws Exception {
124.
125.         if ( rrf == null ) {
126.             throw new IllegalStateException("ResourceResolverFactory not injected");
127.         }
128.
129.         @SuppressWarnings("deprecation") // fine for testing
130.         ResourceResolver rr = rrf.getAdministrativeResourceResolver(null);
131.         try {
132.             call0(rr);
133.             rr.commit();
134.         } finally {
135.             if ( rr != null ) {
136.                 rr.close();
137.             }
138.         }
139.         return null;
140.     }
141.
142.     protected abstract void call0(ResourceResolver rr) throws Exception;
143.
```

```
144. }
```

- d. Examine the code.



NOTE: The TestReference allows you to use the sling testing framework within AEM.

- Save the changes

2. Build the bundle.

- a. Right-click on **training.it.tests** and Choose *Run As > Maven install*

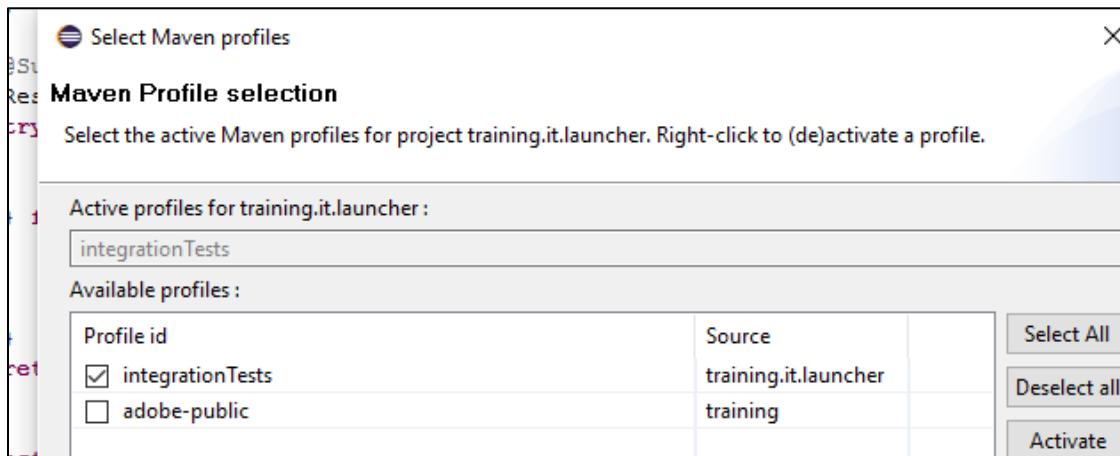
- b. Verify the build was successful:

```
[INFO] skip non existing resourceDirectory C:\AEM\workspace123\training\it.tests\src\test\resource
[INFO]
[INFO] --- maven-compiler-plugin:3.2:testCompile (default-testCompile) @ training.it.tests ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.18.1:test (default-test) @ training.it.tests ---
[INFO]
[INFO] --- maven-bundle-plugin:2.5.3:bundle (default-bundle) @ training.it.tests ---
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ training.it.tests ---
[INFO] Installing C:\AEM\workspace123\training\it.tests\target\training.it.tests-0.0.1-SNAPSHOT.jar
[INFO] Installing C:\AEM\workspace123\training\it.tests\pom.xml to C:\Users\prpurush\.m2\repository\com\adobe\training\it.tests\0.0.1-SNAPSHOT\training.it.tests-0.0.1-SNAPSHOT.jar
[INFO] Writing OBR metadata
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.878 s
[INFO] Finished at: 2017-02-14T12:56:55-08:00
[INFO] Final Memory: 22M/213M
[INFO] -----
```



Note: Now that our testing bundle (**it.tests**) is built for testing, we need to deploy it and the testing framework to the AEM server so we can run the server-side tests. To do this, we can use **it.launcher** module which allows us to install the testing framework into the AEM instance and install our testing bundle. On top of that, once it's installed on the AEM instance, it will automatically run our tests (from **it.tests**) in the console window.

3. To install and run the testing framework, we need to do a Maven install for training.it.launcher.
 - a. First, verify the Maven profile is set for for training.it.launcher module
 - b. Right-click training.it.launcher and choose *Maven > Select Maven Profiles*
 - c. Verify the Integration Tests is selected



- Click OK
4. Navigate to *training.it.launcher > src/test/java > com.adobe.training.it.launcher*
 5. Open the **SlingServerSideTest.java** and examine the code



Note: This class allows us to call the Sling Testing framework and run the tests on the server and output them in the console window as a part of the command 'mvn install – integrationTests'.

6. Test this by right-clicking on training.it.launcher and Choose *Run As > Maven Install*

- Verify the build is successful
- Scroll through the maven console and examine the number of tests that ran. Notice how SlingServerSideTest.java was run which allows our Sling tests to be run through the maven console, specifically StockModelServerSideTest.java

```
[terminated> C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe [Feb 14, 2017, 1:01:37 PM]
9652 [main] INFO org.apache.sling.testing.tools.osgi.WebconsoleClient - Starting bundle com.adobe.training.core via /system/console/bundles/com.adobe.t
10420 [main] INFO org.apache.sling.testing.tools.osgi.WebconsoleClient - Starting bundle org.apache.sling.junit.remote via /system/console/bundles/org.
11154 [main] INFO org.apache.sling.testing.tools.osgi.WebconsoleClient - Starting bundle org.apache.sling.testing.tools via /system/console/bundles/or
11797 [main] INFO org.apache.sling.testing.tools.osgi.WebconsoleClient - Starting bundle org.apache.httpcomponents.httpclient via /system/console/bundl
12501 [main] INFO org.apache.sling.testing.tools.osgi.WebconsoleClient - Starting bundle org.apache.httpcomponents.httpcore via /system/console/bundles
13182 [main] INFO org.apache.sling.testing.tools.sling.BundlesInstaller - Ok - all bundles are in the active state
13229 [main] INFO com.adobe.training.it.launcher.SlingServerSideTest - Checking that /test/sling/1487106103618.junit returns status 200, timeout=30 sec
13260 [main] INFO com.adobe.training.it.launcher.SlingServerSideTest - /test/sling/1487106103618.junit is ready, returns expected content
13260 [main] INFO org.apache.sling.junit.remote.httpclient.RemoteTestHttpClient - Executing test remotely, path=/com.adobe.training.it.tests.json JUnit
13307 [main] INFO org.apache.sling.junit.remote.testrunner.SlingRemoteTestRunner - Server-side tests executed as admin at http://localhost:4502/test/s
Running com.adobe.training.it.launcher.SlingServerSideTest
13307 [main] INFO org.apache.sling.junit.remote.httpclient.RemoteTestHttpClient - Executing test remotely, path=/com.adobe.training.it.tests.json JUnit
13339 [main] INFO org.apache.sling.junit.remote.testrunner.SlingRemoteTestRunner - Server-side tests executed as admin at http://localhost:4502/test/s
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.048 sec - in com.adobe.training.it.launcher.SlingServerSideTest

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is platform dependent! The file encoding for reports output files
[INFO]
[INFO] --- maven-failsafe-plugin:2.18.1:verify (verify) @ training.it.launcher ---
[INFO] Failsafe report directory: C:\AEM\workspace123\training\it.launcher\target\failsafe-reports
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is platform dependent! The file encoding for reports output files
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ training.it.launcher ---
[INFO] Installing C:\AEM\workspace123\training\it.launcher\target\training.it.launcher-0.0.1-SNAPSHOT.jar to C:\Users\ppurush\.m2\repository\com\adobe\trai
[INFO] Installing C:\AEM\workspace123\training\it.launcher\pom.xml to C:\Users\ppurush\.m2\repository\com\adobe\training.it.launcher\0.0.1-SNAPSHOT\t
[INFO] -----
[INFO] BUILD SUCCESS
```

7. Run scriptable server-side tests using the browser.



NOTE: This is another method to test the class.

- Browse to: <http://localhost:4502/system/sling/junit/>

JUnitServlet

Test selector: RequestParser, testSelector [], methodName [], extension []

Test classes

- com.adobe.cq.aam.client.testing.SynchronisationJobServerTest
- com.adobe.training.it.tests.StockModelServerSideTest
- org.apache.sling.junit.remote.exported.ExampleRemoteTest
- org.apache.sling.junit.remote.testrunner.SlingRemoteTest
- org.apache.sling.junit.scriptable.ScriptableTestsProvider

Execute these tests

We see all the Tests that are on the server since JunitServlet is installed on the AEM by the training.it.launcher bundle.

c. Execute the test StockModelServerSideTest by browsing to:
<http://localhost:4502/system/sling/junit/com.adobe.training.html>

d. Click the button Execute these Tests

JUnitServlet

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

Test classes

- com.adobe.training.it.tests.StockModelServerSideTest

Execute these tests

d. Notice the test ran successfully:

JUnitServlet

Running tests

com.adobe.training.it.tests.StockModelServerSideTest

Test finished: testStockModelServerSide(com.adobe.training.it.tests.StockModelServerSideTest)

TEST RUN FINISHED: tests:1 , failures:0 , ignored:0

Exercise 3 – Create Scriptable Server-Side Tests

Overview

In this lab exercise, you will create scriptable server-side tests in JCR. 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.

Steps

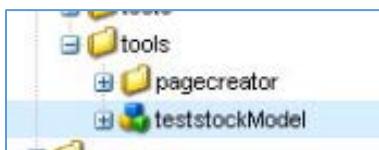
1. Create a node named teststockNode with a mixin type sling:Test.

- a. Open CRXDE Lite, and navigate to *apps > trainingproject > tools*

- b. Under the tools folder, create a node

- Name:**testStockModel**

- Type **nt:unstructured**



- c. Save All

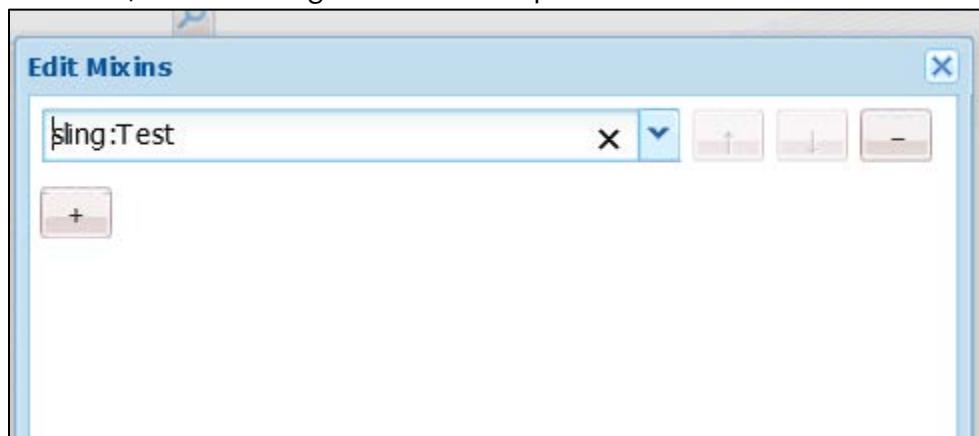
- d. Add the following property to the teststockModel node:

Name	Type	Value
sling:resourceType	String	trainingproject/tools/testStockModel

- Save All

- f. Select the teststockModel node, and from the top of CRXDE Lite, select Mixins...

- g. Click the +, and select sling:Test from the drop-down list.



Note: You may have to refresh the browser screen before the sling:Test mixin appears in the drop-down list.

- h. Click OK and save your changes.
2. Create a rendering script.
- Right-click **testStockModel**, choose *Create > Create file...*
 - Name: test.txt.html
 - Add the following code to test.txt.html from the */Exercise_Files/11_Writing_Tests/*

```

1. <!--/* To test this script, use the ScriptableTestsProvider in the Sling Junit Servlet */-->
2. <!--
   /* http://localhost:4502/system/sling/junit/org.apache.sling.junit.ScriptableTestsProvider.html */-->
3. <!--/* curl -u admin:admin -
   X POST http://localhost:4502/system/sling/junit/org.apache.sling.junit.ScriptableTestsProvider.json */-->
4.
5. <!--/* In order for this test to successfully pass, it must output TEST_PASSED */-->
6.
7. <!--/* Test to make sure Last Trade info is successfully imported */-->
8. <sly data-sly-
   use.output="${'logic.js' @ symbol=['ADBE','MSFT','GOOG']}">${output}</sly>
```

- d. Save the changes

3. Create the business logic

- a. Right-click testStockModel, Create > Create file...

- Name: logic.js

- Add the following code to logic.js from the /Exercise_Files/11_Writing_Tests/

```

1. "use strict";
2.
3. use(["/libs/wcm/foundation/components/utils/AuthoringUtils.js"], function (AuthoringU
   tils) {
4.
5.     var symbols = this.symbol;
6.     var failures = "";
7.
8.     if(!symbols || symbols.length < 1)
9.         return " NO_SYMBOLS_TO_TEST ";
10.
11.    var i;
12.    for (i = 0; i < symbols.length; i++) {
13.        var curSymbol = symbols[i];
14.        var res = resource.getResourceResolver().getResource("/content/" + curSymbol);
15.        log.info("Testing: " + curSymbol);
16.        //Check if Stock Symbol exists
17.        if(res != null){
18.            var stockModel = res.adaptTo(com.adobe.training.core.models.StockModel);
19.
20.            //if there is no last trade or the last trade is below 0, add to failures
21.            if((stockModel == null)
22.               || (stockModel.getLastTrade() < 0)){
23.                failures = failures + " " + curSymbol;
24.            }
25.        }
26.        else {
27.            failures = failures + " " + curSymbol;
28.        }
29.    }
30.
31.    //If there are any symbols that fail, return them
32.    if(failures)
33.        return " FAILED_SYMBOLS: [" + failures.trim().replace(/\ /g,"") + "] ";
34.    else
35.        return "TEST_PASSED"; //Keyword to allow the test to pass
36. });

```

- b. Examine the code and save the changes



Note: Here we are calling logic.js from test.text.html and passing the stock symbols and then testing against them. If all the three of them exist in the correct format, the output "Test Passed" is printed.

4. Use the browser to test.

- a. Execute the test by browsing to:

<http://localhost:4502/system/sling/junit/org.apache.sling.junit.scriptable.ScriptableTestsProvider.html>

- b. Click the button Execute these Tests
c. Your test should fail:

JUnitServlet

Running tests

org.apache.sling.junit.scriptable.ScriptableTestsProvider

TEST FAILED: verifyContent[0](org.apache.sling.junit.scriptable.TestAllPaths)

verifyContent[0](org.apache.sling.junit.scriptable.TestAllPaths): Unexpected content at path / just TEST_PASSED (lines starting with # and empty lines are ignored) content was: FAILED_SYMBOLS

Test finished: verifyContent[0](org.apache.sling.junit.scriptable.TestAllPaths)
TEST RUN FINISHED: tests:1 , failures:1 , ignored:0

Note: This is because we did not do the polling importer for the Stock symbols for MSFT, or GOOG.

5. Go to test.text.html and delete these two stock symbols as shown below:

```
1 <!--/* To test this script, use the ScriptableTestsProvider in the Sling Junit Servlet */-->
2 <!--/* http://localhost:4502/system/sling/junit/org.apache.sling.junit.scriptable.ScriptableTes
3 <!--/* curl -u admin:admin -X POST http://localhost:4502/system/sling/junit/org.apache.sling.ju
4
5 <!--/* In order for this test to successfully pass, it must output TEST_PASSED */-->
6
7 <!--/* Test to make sure Last Trade info is successfully imported */-->
8 <sly data-sly-use.output="${'logic.js' @ symbol=['ADBE']}">${output}</sly>
```

- Save the changes

6. Re-execute the test:

<http://localhost:4502/system/sling/junit/org.apache.sling.junit.scriptable.ScriptableTestsProvider.html>

This time it should execute successfully:

JUnitServlet

Running tests

org.apache.sling.junit.scriptable.ScriptableTestsProvider

Test finished: verifyContent[0](org.apache.sling.junit.scriptable.TestAllPaths)

TEST RUN FINISHED: tests:1 , failures:0 , ignored:0

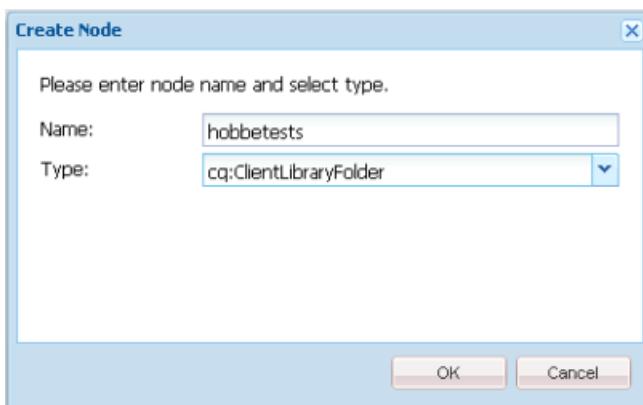
Exercise 4 – Create an Automated test with Hobbes

Overview

In this lab exercise, you will create an automated test with Hobbes. In this exercise, we will create a new client library to support Hobbes testing in AEM 6.2.

Steps

1. Create a node in CRXDE lite.
 - a. In CRXDE lite, navigate to `/apps/trainingprojects/` folder
 - b. Create a Node of type `cq:ClientLibraryFolder` under training project called **hobbestests**
 - c. Save All



- d. Add property of type `String[]`
 - Name: **categories**
 - Type: `String[]`
 - Value: **granite.testing.hobbes.tests**



Note: This property needs to be of type String Array. To do this, you need to select the Multi button.

- e. Click **Multi** and click **OK**
- f. Click Add
- g. Add another property of type `String[]`

- Name: **dependencies**
- Type: String[]
- Value: **granite.testing.hobbes.testrunner**



Note: This property needs to be of type String Array. To do this, you need to select the Multi button.

- h. Click **Multi** and click **OK**
- i. Click Add

Properties		Access Control	Replication	Console	Build Info
	Name	Type	Value		
1	categories	String[]	granite.testing.hobbes.test		
2	dependencies	String[]	granite.testing.hobbes.testrunner		
3	jcr:primaryType	Name	cq:ClientLibraryFolder		

- j. Save All

2. Under the **hobbestests** node, Create > Create File

- Name: **SampleTests.js**



Note: The code is provided as part of the Exercise_Files under /Exercise_Files/11_Writing_Tests/. Copy and paste the code from the exercise files to Eclipse.

```

1. /*
2. * Copyright 2015 Adobe Systems Incorporated
3. *
4. * Licensed under the Apache License, Version 2.0 (the "License");
5. * you may not use this file except in compliance with the License.
6. * You may obtain a copy of the License at
7. *
8. *      http://www.apache.org/licenses/LICENSE-2.0
9. */

```

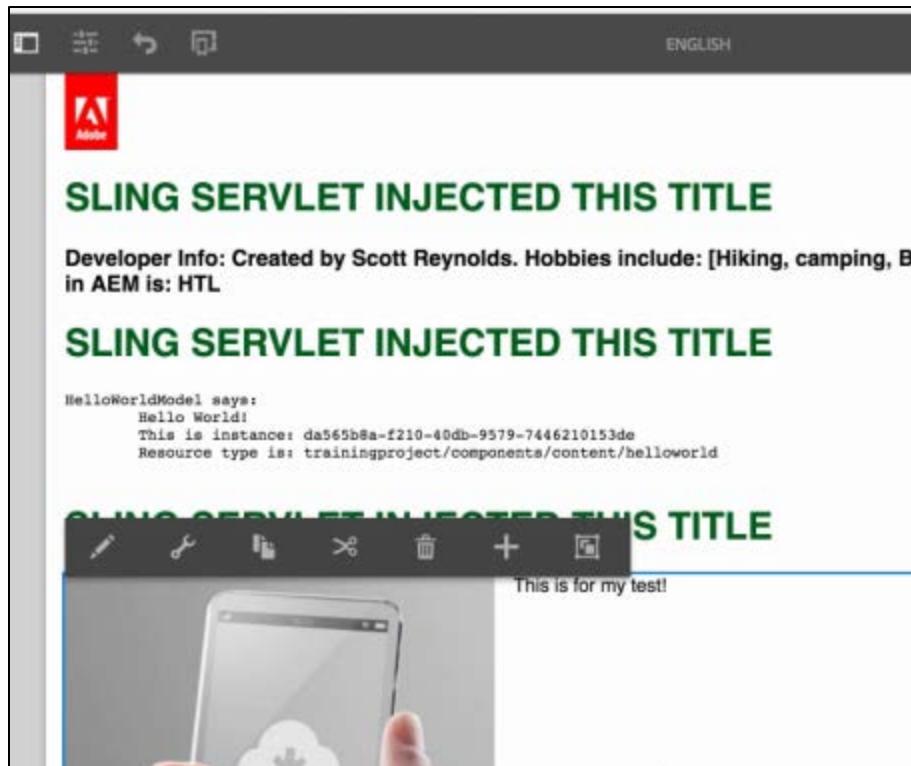
```
10. * Unless required by applicable law or agreed to in writing, software
11. * distributed under the License is distributed on an "AS IS" BASIS,
12. * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13. * See the License for the specific language governing permissions and
14. * limitations under the License.
15. */
16. hobs.param("enPage", "/content/trainingproject/en.html")
17. new hobs.TestSuite("TrainingProject Tests", {path:"/apps/trainingproject/tests/SampleTests.js", register: true})
18. .addTestCase(new hobs.TestCase("Hello World component on english page")
19. .navigateTo("%enPage%")
20. .asserts.location("%enPage%", true)
21. .asserts.visible(".helloworld", true)
22. )
23.
24. .addTestCase(new hobs.TestCase("Hello World component on french page")
25. .navigateTo("/content/trainingproject/fr.html")
26. .asserts.location("/content/trainingproject/fr.html", true)
27. .asserts.visible(".helloworld", true)
28. )
29. .addTestCase(new hobs.TestCase("Text and Image Component on English Page")
30. .navigateTo("%enPage%")
31. .asserts.location("%enPage%", true)
32. .asserts.exists("div.textimage.parbase", true)
33. );
```

3. Create another file.
 - a. Under the **hobbestests** node, create a file
 - Name: **js.txt**
 - Add the following code:
SampleTests.js
 - b. Save the changes

4. To test it, you can navigate to: <http://localhost:4502/libs/granite/testing/hobbes.html>
 - a. Click on TrainingProject Tests
 - b. Select 'Run all Tests'
 - c. Notice the test fails:

The screenshot shows a browser window titled 'CRXDE Lite' with the URL 'localhost:4502/libs/granite/testing/hobbes.html'. The page displays a list of test cases under the 'TrainingProject Tests' category, which has 3 items. The first two tests ('Hello World component on english page' and 'Hello World component on french page') are marked as successful (green checkmark). The third test ('Text and Image Component on English Page') is marked as failed (red X). A tooltip for this failed test provides details: 'asserts location : assert TEST PAGE LOCATION qui blandit nulla facilis etiam facilis'. The tooltip also includes a screenshot icon and a link to 'ELEMENT not found in the DOM'. At the bottom of the page, there is a note: 'All rights reserved by Adobe Systems Incorporated.'

5. Examine the error. It fails because there is no text and image component added to the page.
 - a. Fix the error by adding a 'Text and Image' component to the page:
 - b. Go to Sites (<http://localhost:4502/sites.html/content>), open the English page and drag and drop a 'Text and image' component onto the page
 - c. Add an Image and some text to it:



6. Test it and verify it succeeds

- a. Navigate to: <http://localhost:4502/libs/granite/testing/hobbes.html>
- b. Click on TrainingProject Tests
- c. Select 'Run Tests'
- d. Click on Run tests icon and verify it runs successfully:

The screenshot shows the AEM Testing interface. On the left, there's a sidebar with a tree view of test categories. Under 'TrainingProject Tests', two items are listed: 'Hello World component on english page' and 'Hello World component on french page'. Both items have green checkmarks next to them, indicating they have run successfully. The main content area displays three identical green headers: 'SLING SERVLET INJECTED THIS TITLE'. Below each header is a snippet of developer information and a large amount of placeholder text ('Lorem ipsum...'). The developer info includes the name 'Scott Reynolds', hobbies like 'Hiking, camping, Bikin', and 'in AEM is: HTL'. The placeholder text is a long string of Latin words.

7. Import code back into eclipse with "import from server" option.



NOTE: Now that we created nodes in CRXDE Lite, we have to add the code to our local eclipse project. Import code with the "Import from server" option.

Review:

You have performed the following tests on your code:

- A Unit test using Mockito
- Sling-JUnit test
- Scriptable server-side test
- Hobbes Test

12 CONTENT INGESTION

Overview

This module explains the various methods available to migrate your data from an existing legacy system to Adobe Experience Manager. The Lab Activity section includes hands-on exercises on performing migration using Content Package manipulation, Sling POST servlets, and the JCR API.

Objectives

By the end of this module, you will be able to:

- Migrate your data from an existing legacy system to Adobe Experience Manager

Understanding Data Migration

Migration occurs when you have to transfer your content from an existing system (legacy system) to Adobe Experience Manager. A major challenge that you would face is when you have to migrate content from different systems. Combining content from various Content Management Systems (CMS) involves a thorough study of its architecture.

The Migration Process

The following are best practices for migrating data from legacy systems:

1. Identify the source.

Most legacy systems are based on databases that relate to the data being stored. You need to extract this data from the source format into a neutral format. For example, you can extract the data into XML.

2. Transform the existing content to the target format.

The extracted data must be converted into a format (xml format) that is recognizable by Adobe Experience Manager. That is, it should translate itself into nodes and properties as identified by the JCR. You need to use David's model to map data into the JCR repository.

3. Import into the JCR.

Import and store the content in the JCR using tools such as FileVault or Package Manager.

There are two approaches to data migration:

- One-time migration. For example, when converting a Drupal site to an Adobe Experience Manager site.
- Continuous migration. For example, in ecommerce systems, products, vouchers, payments, and such need to move from the database into Adobe Experience Manager.

There are different types of migration:

- Automated migration

This type of migration is completed automated, with no manual intervention during the migration.

- Partially automated migration

Depending on the size and type of data, there would be instances where some parts of the migration is done manually, the rest done through automation.

- Manual migration
This type of migration is done manually.
- Migration on request
This type of migration is an ad-hoc migration and depends on the size and type of data.

As a final note, whenever you need to migrate content, consider the following:

- Migration is not just copying and pasting data into the JCR.
- The migration plan differs with each project.
- Make good use of the xml format.

Identifying the Challenges of Data Migration

A few of the challenges faced during data migration from legacy systems to Adobe Experience Manager are:

- Migrating data from different systems.
- Most legacy systems already have a CMS in place.
- Migration can be time-consuming as legacy systems can lack taxonomy and metadata.

Migrating Data from Legacy Systems

Adobe Experience Manager comes with a set of predefined sample sites that you can use to analyze for its data structure. You can then either make use of the same sites with modifications or recreate the site to suit your requirements.

Consider the following when planning your migration from legacy systems:

- Analyze the source system.
- Define the actual data architecture and all the special considerations for the migration.
- Pre-format data considerations, and make them presentable to the content migrator.

Implementation considerations:

- Define the data manipulation that needs to take place.
- Consider breaking the migration process into sub-migrations.
- Modify the data coming out of the source system based on the business requirements.

- Define the node structure for the new system.
- Perform a test on dummy data before testing on real data. You can use the Sling POST servlet to perform these tests.

There are three basic possibilities to migrate content from legacy systems into Adobe Experience Manager:

- Using content packages to install structure into the JCR
- Using the Sling POST servlet
- Using the JCR API

Content Package

Content package migration is efficient for large quantities of content, but it may not be easy to debug if you get the format wrong. Content packages use a protocol call FileVault (VLT). When migrating content via content packages you may use package manager, eclipse, or the command line tool. All methods use the underlying VLT tool.

The Package Manager is a packaging tool that is available with Adobe Experience Manager, which you can use to create packages to export content into external systems, or to import content from other systems. In this module, you will use the Package Manager to migrate data from your legacy systems into the JCR.

When to Use Packages for Migration

You can use content packages to migrate large amounts of data (Gigabytes or Terabytes) in a short period of time. However, a major requirement for this process is that the structure of the data being imported must be in the right format. It must match the JCR repository, in its nodes and properties.

Using the Package Manager for Migration

The Package Manager is a utility that runs on FileVault. When a package is uploaded, the install function of the package manager unzips the file and uses FileVault to write the content into the repository. When a package is created, the package manager uses FileVault to export the content, and then packages the resulting file structure into a zipped folder.

Content packages can contain content or project-related data. It is a zipped file that resembles the Vault serialization. The `.content.xml` file represents the content from the repository, in the form of files and folders. It contains vault meta-information, filter definitions, and import-configuration information.

The package must have the following structure:

- **jcr_root:** This folder represents the root node of the repository, and contains the actual content of the package.
- **META-INF:** This folder contains metadata regarding node definitions, and also contains the filter.xml, which gives directions to FileVault about which paths to include.



Perform Task –[Exercise 1 – Create a Page using a Content Package](#) from Lab K.

Sling POST Servlet

The main purpose of using this method is to test the migration and ensure the content structure is accurate before doing the actual migration. To use this method, you must first verify you have curl installed in your system. cURL is used to issue an http request, and call the Sling POST servlet to post new content into the repository.

Migrating content

Follow these steps to migrate content using the Sling POST servlet:

1. Export the content from the legacy system to your file system.
2. Use a shell script to transform the content into curl commands to the Sling POST servlet.



Perform Task –[Exercise 2 – Add Content with the Sling POST Servlet](#) from Lab K.

Using the JCR API

JCR APIs are used to automate the migration of content from legacy systems to Adobe Experience Manager. This method is efficient and allows the use of Adobe Experience Manager/JCR methods. It is also useful for scripted fixes to the content structure.



Perform Task –[Exercise 3 – Add Content with the JCR API](#) from Lab K.

LAB K - USE DIFFERENT METHODS TO INGEST CONTENT

Scenario

Your company plans to migrate from an existing Content Management System (CMS) to Adobe Experience Manager. Using existing data, you need to perform a series of steps to move all content into the new repository.

Challenge

There are many ways to migrate content from an existing CMS. Selecting the right method is essential for a quick and accurate migration. These methods differ based on the type and bulk of the existing data.

Objectives

- Create a page with Content Package
- Add content with the Sling POST Servlet
- Add Content with the JCR API

Prerequisites

- AEM installed on your machine:
 - Running Adobe Experience Manager Author instance
 - An Eclipse project from the AEM Archetype

Directions

Complete the exercises that follow

Exercise 1 – Create a Page using a Content Package

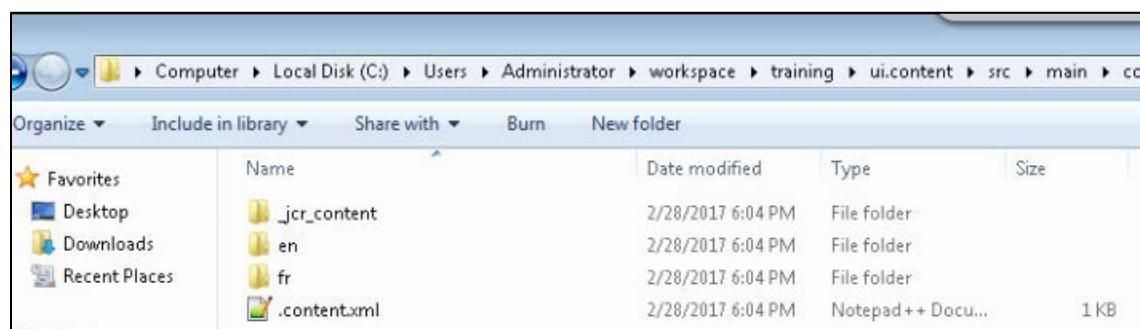
Overview

In this lab exercise, you will use the AEM plugin in eclipse to modify the underlying content package to create a new language root in the website created by the AEM archetype.

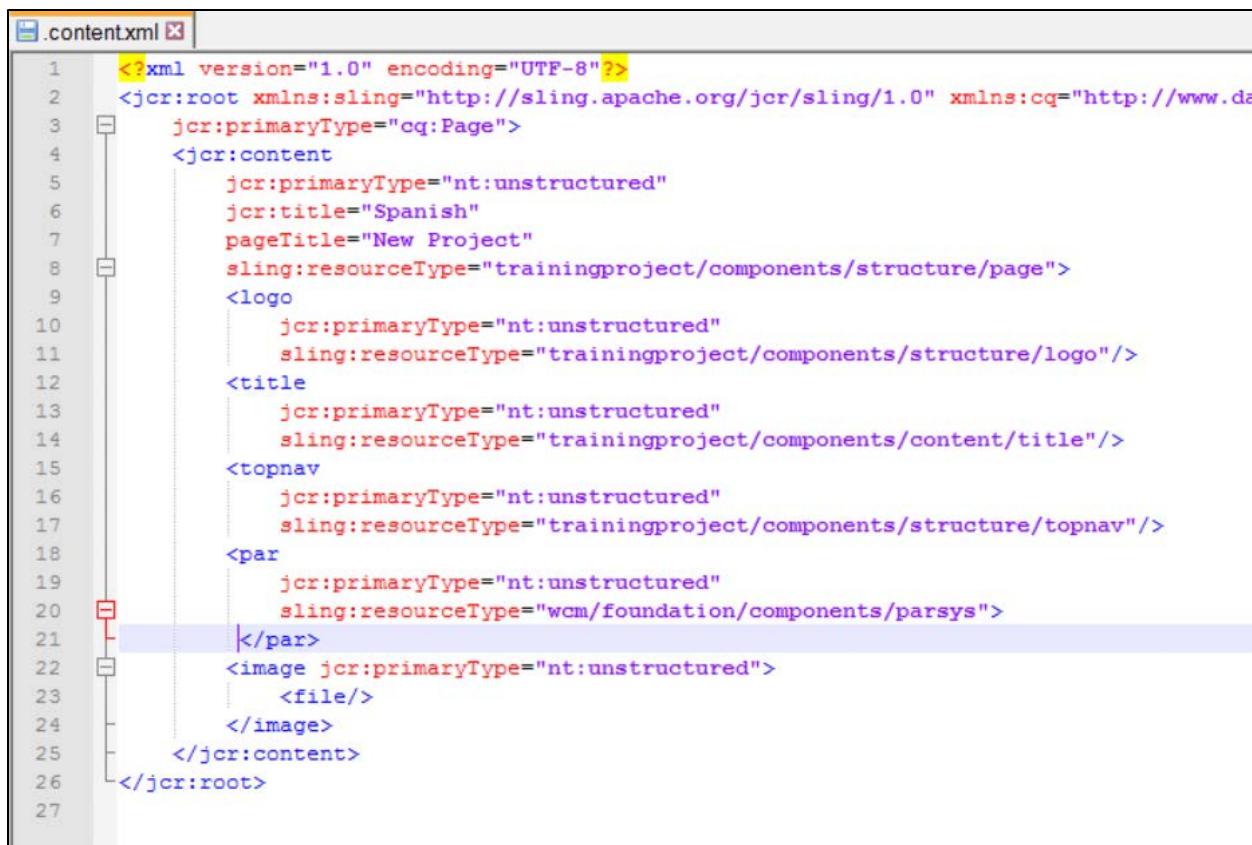
Steps

1. Edit the content package in the file system.

- a. Navigate to `/training/ui.content/src/main/content/jcr_root/content/trainingproject` under your workspace folder



- b. Copy and paste the previously existing page 'en' and rename it to 'es'
- c. Under the **es** folder, open the `.content.xml` file in a text editor
 - In the `jcr:title` property, replace 'English' with '**Spanish**'
- d. Delete all the components inside `<par>` tag as shown below:



The screenshot shows a code editor window with the file name ".content.xml" at the top. The content of the file is as follows:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0" xmlns:cq="http://www.adobe.com/jcr/cq/1.0" jcr:primaryType="cq:Page">
3   <jcr:content>
4     <jcr:primaryType="nt:unstructured"
5       jcr:title="Spanish"
6       pageTitle="New Project"
7       sling:resourceType="trainingproject/components/structure/page">
8       <logo>
9         <jcr:primaryType="nt:unstructured"
10        sling:resourceType="trainingproject/components/structure/logo"/>
11       <title>
12         <jcr:primaryType="nt:unstructured"
13           sling:resourceType="trainingproject/components/content/title"/>
14       <topnav>
15         <jcr:primaryType="nt:unstructured"
16           sling:resourceType="trainingproject/components/structure/topnav"/>
17       <par>
18         <jcr:primaryType="nt:unstructured"
19           sling:resourceType="wcm/foundation/components/parsys">
20           </par>
21           <image jcr:primaryType="nt:unstructured">
22             <file/>
23           </image>
24         </jcr:content>
25       </jcr:root>
26     </jcr:content>
27   </jcr:root>
```

- e. Save the changes

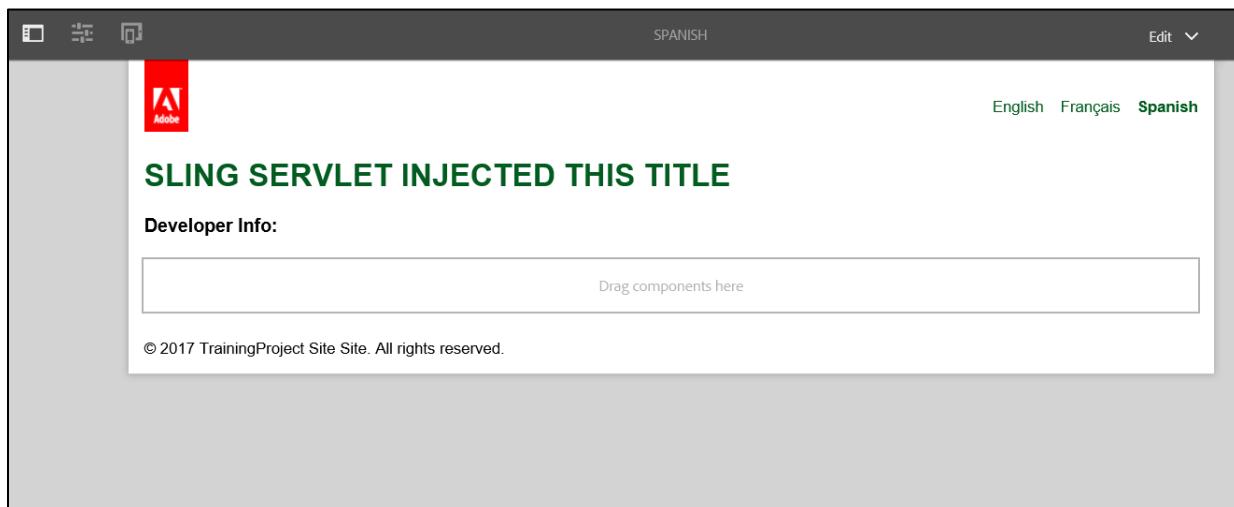
2. Import the updated page into the JCR.

- a. Back in Eclipse, navigate to
training.ui.content>src/main/content/jcr_root>content>trainingproject
- b. Right-click **trainingproject** and select **Refresh**

Note: You should now see the **es** page in Eclipse.

- c. Right-click **training.ui.content** and select *Run As > Maven Install*
- d. Verify the build was success

<http://localhost:4502/editor.html/content/trainingproject/es.html>



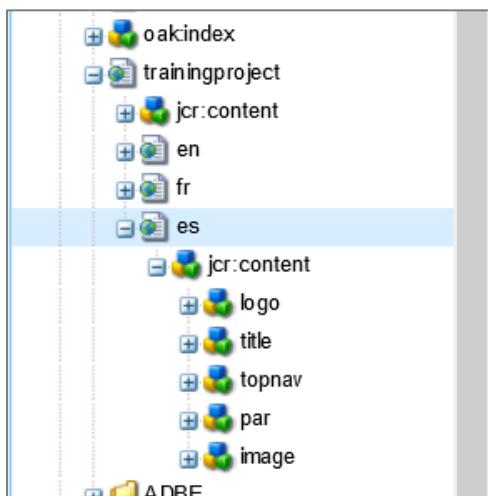
Exercise 2 – Add Content with the Sling POST Servlet

Overview

In this lab exercise, you will add content with the Sling POST Servlet.

Steps

1. Use Sling POST servlet to test XML.
 - a. Inspect the es site in CRXDE Lite.
 - b. You need to add a text node to/content /trainingproject/es/.



2. On the OS command line, execute the following command (it is a one-line command, with no spaces in the URI):

Note: It is available as part of /Exercise_Files/12_Content_Ingestion/curl.txt

```
$ curl -u admin:admin -X POST -
d "sling:resourceType=foundation/components/text&text=<h3>I am a new Text Component
from the Sling POST servlet</h3>&textIsRich=true" http://localhost:4502/content/trainingproject/es/jcr:content/par/*
```

This is POSTing a rich text string to the par node under /content//trainingproject/es/.

3. Verify the content was added to the Spanish page

- <http://localhost:4502/editor.html/content/trainingproject/es.html>

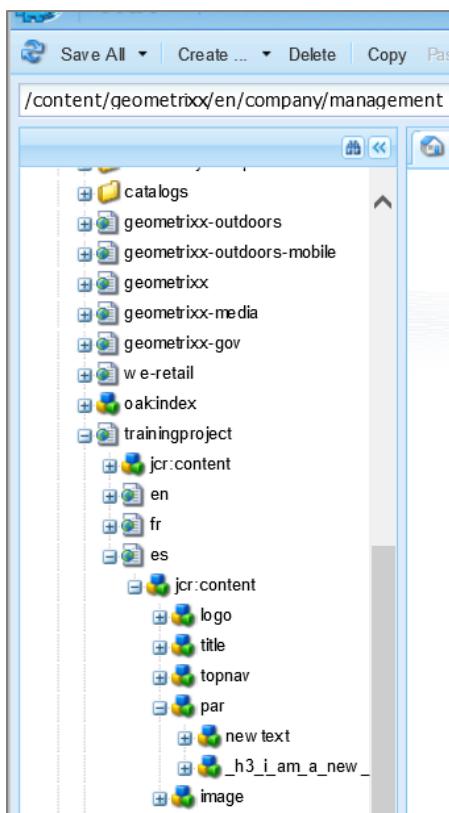
SLING SERVLET INJECTED THIS TITLE

Developer Info:

I am a new Text Component from the Sling POST servlet

© 2017 TrainingProject Site Site. All rights reserved.

If you check the nodes in CRXDE Lite, you can see the new node added.



Exercise 3 – Add Content with the JCR API

Overview

In this exercise, you will use Java code that makes use of the JCR API to search for specific content, and add new content wherever it is found. You can use this technique to update content from a legacy system, by adding the required nodes or properties to make the content usable in Adobe Experience Manager.

Steps

1. Add the ImporterServlet to the core project.
 - a. In Eclipse, navigate to `/training.core>src/main/java>com.adobe.training.core.servlets`
 - b. Right-click `com.adobe.training.core.servlets` and choose `New >class`
 - Name: `ImporterServlet`
 - Click `Finish`
 - c. Copy and paste the code from `/Exercise_Files/12_Writing_Tests/`

```
1. package com.adobe.training.core.servlets;
2.
3. import java.io.IOException;
4.
5. import javax.jcr.NodeIterator;
6. import javax.jcr.RepositoryException;
7. import javax.jcr.Session;
8. import javax.jcr.query.Query;
9. import javax.jcr.Node;
10. import javax.servlet.ServletException;
11.
12. import org.apache.felix.scr.annotations.sling.SlingServlet;
13. import org.apache.sling.api.SlingHttpServletRequest;
14. import org.apache.sling.api.SlingHttpServletResponse;
15. import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
16. import org.slf4j.Logger;
17. import org.slf4j.LoggerFactory;
18.
19. import com.day.cq.commons.jcr.JcrUtil;
20.
21. /**
22. * Servlet that allows a text component to be added to pages
23. *
24. * To use this servlet, a new node must be created:
25. * /etc/trainingproject/importer (nt:unstructured)
26. *      +sling:resourceType=trainingproject/tools/importer
27. *
28. * To Test:
```

```

29. * http://localhost:4502/etc/trainingproject/importer.html
30. *
31. * @author Kevin Nennig (nennig@adobe.com)
32. */
33. @SlingServlet(resourceTypes="trainingproject/tools/importer")
34. public class ImporterServlet extends SlingSafeMethodsServlet {
35.     Logger logger = LoggerFactory.getLogger(this.getClass());
36.
37.     private static final long serialVersionUID = 1L;
38.
39.     @Override
40.     public final void doGet(final SlingHttpServletRequest request, final
41.     SlingHttpServletResponse response)
42.             throws ServletException, IOException {
43.         response.setHeader("Content-Type", "text/html");
44.
45.         Session session = request.getResourceResolver().adaptTo(Session.class);
46.
47.         try{
48.             String q = "/jcr:root/content/trainingproject/es/*" +
49.                     "[@sling:resourceType='wcm/Foundation/components/parsy
50. s']";
51.             Query query = session.getWorkspace().getQueryManager()
52.                     .createQuery(q, "xpath");
53.             NodeIterator result = query.execute().getNodes();
54.             while (result.hasNext()) {
55.                 Node n = result.nextNode();
56.                 Node newTextNode = JcrUtil.createUniqueNode(n, "newtext",
57.                         "nt:unstructured", session);
58.                 newTextNode.setProperty("sling:resourceType",
59.                         "foundation/components/text");
60.                 newTextNode.setProperty("text", "<h3>A Text Component from
61. a Servlet using the JCR API</h3>");
62.                 newTextNode.setProperty("textIsRich", "true");
63.                 response.getWriter().print("Added node: " + n.getPath() +
64.                     "<br />");
65.             }
66.             session.save();
67.         } catch (RepositoryException e){
68.             response.getWriter().print("Could not create nodes");
69.             logger.error(e.getMessage() + e);
70.         }
71.         response.getWriter().close();
72.     }
73. }
74. }
```

- d. Examine the code
- e. Save the changes

2. Deploy the project to server (Run As > Maven install)

3. Create a new Node to test invoke the ImporterServlet class.
 - a. In CRXDE lite, navigate to /etc
 - b. Create >Create Node
 - Name: trainingproject
 - Type: sling:Folder
 - c. Create another node under trainingproject
 - Name: importer
 - Type: nt:unstructured
 - d. Add the following property:
 - Name: sling:resourceType
 - Type: String
 - Value: trainingproject/tools/importer

Properties				Access Control	Replication	Console	Build Info
	Name	Type	Value				
1	jcr:primaryType	Name	nt:unstructured				
2	sling:resourceType	String	trainingproject/tools/importer				

- e. Click Add
- f. Select Save All

4. Invoke the ImporterServlet class

- a. Go to <http://localhost:4502/etc/trainingproject/importer.html>
- b. Verify the added node:

The screenshot shows a browser window with the URL <http://localhost:4502/etc/trainingproject/importer.html> in the address bar. Below the address bar, there are several icons and links: 'my open issues', 'SharePoint', 'JIRA', 'Adobe Connect Central L...', and 'Adobe Con...'. The main content area displays the message: 'Added node: /content/trainingproject/es/jcr:content/par'.

5. Verify the content was added to the Spanish page

- a. Go to <http://localhost:4502/editor.html/content/trainingproject/es.html>

The screenshot shows a page titled 'SLING SERVLET INJECTED THIS TITLE'. The page includes developer information: 'A Text Component from a Servlet using the JCR API' (which is highlighted with a red box) and 'I am a new Text Component from the Sling POST servlet'. At the bottom, it says '© 2017 TrainingProject Site Site. All rights reserved.' The page has a red Adobe logo in the top left and language links for English, Français, and Spanish in the top right.

Review:

In this lab, you did the following exercises:

- Created a page using content package
- Added content with the Sling POST Servlet
- Added Content with the JCR API

13 USERS, GROUPS AND PERMISSIONS

Overview

This module includes content and hands-on activities to determine what actions a user or group can take and where it can perform those actions. Adobe Experience Manager uses Access Control Lists (ACLs) to perform these tasks.

Objectives

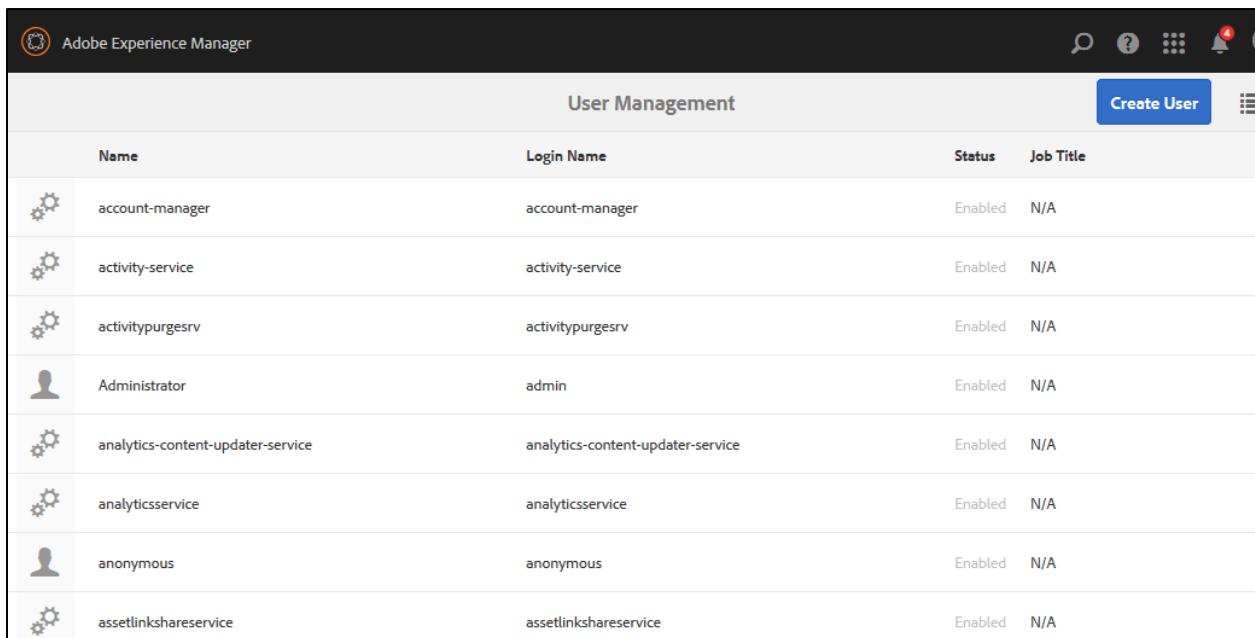
By the end of this module, you will be able to:

- Create and assign users, groups, and permissions
- Automate Permissions

Users and Groups

Users: A user models either a human user or an external system connected to the system. The user account holds the details needed for accessing Adobe Experience Manager. A key purpose of an account is to provide the information for the authentication and login processes—allowing a user to log in. Each user account is unique and holds the basic account details, together with the privileges assigned. Users are often members of groups, which simplifies the allocation of these permissions and/or privileges.

Navigation: <http://localhost:4502/libs/granite/security/content/useradmin.html>



Name	Login Name	Status	Job Title
account-manager	account-manager	Enabled	N/A
activity-service	activity-service	Enabled	N/A
activitypurgesrv	activitypurgesrv	Enabled	N/A
Administrator	admin	Enabled	N/A
analytics-content-updater-service	analytics-content-updater-service	Enabled	N/A
analyticsservice	analyticsservice	Enabled	N/A
anonymous	anonymous	Enabled	N/A
assetlinksharebservice	assetlinksharebservice	Enabled	N/A

Groups: These are collections of users and/or other groups; these are all called members of a group. Their primary purpose is to simplify the maintenance process by reducing the number of entities to be updated, as a change made to a group is applied to all members of the group.

Navigation: <http://localhost:4502/libs/granite/security/content/groupadmin.html>

Group Management				Create Group	
Name	Description	Members	Published	Modified	
administrators		1	Not Published	Not Modified	
af-template-script-writers	AEM forms group with permissions to write scripts in Adaptive Forms template	0	Not Published	Not Modified	
Analytics Administrators	Analytics Administrators group	12	Not Published	Not Modified	
Communities Administrators		1	Not Published	Not Modified	
Community Community Communitymanagers	communitymanagers for site /content/we-retail/us/en/community	2	Not Published	Not Modified	
Community Community Desert Hiking Cnhxq Communitymanagers		3	Not Published	Not Modified	
Community Community Desert Hiking Cnhxq Members		7	Not Published	Not Modified	
Community Community Desert Hiking Cnhxq Moderators		5	Not Published	Not Modified	

For more information on User Administration and Security, refer to: <https://docs.adobe.com/docs/en/aem/6-2/administer/security/security.html>

Rights Storage

While this module covers user administration and security, the following webpage is recommended:

<https://docs.adobe.com/docs/en/aem/6-2/administer/security/user-group-ac-admin.html>

The User, Group and Access Rights Administration webpage will provide you with in-depth explanations of the following topics, most of which are covered in the exercises:

- Access Rights
- User Administration
- Group Administration
- Access Right Management

Access Right Management is important, especially with different users accessing and performing different tasks within Adobe Experience Manager. With the access Control tab in CRXDE lite, you can configure access control policies and assign their corresponding privileges. The “Permissions and ACLs” on page 14-8 features an overview of these topics along with corresponding exercises to help you better understand how you can create “bespoke” content for your audiences (for example, localized content based on user).

Permissions and ACLs

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

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 check boxes for the individual Adobe Experience Manager actions. A checkmark indicates an action is allowed. No check mark indicates an action is denied.

Properties	Groups	Members	Permissions	Impersonators	Preferences			
Path	Read	Modify	Create	Delete	Read A...	Edit ACL	Replicate	
	<input checked="" type="checkbox"/> *	Details						
apps	<input checked="" type="checkbox"/>	Details						

The location of the checkmark in the grid also indicates what permissions users have in what locations within Adobe Experience Manager (that is, which paths).

Action	Allow (Checkmark)	Deny (No Checkmark)
Description	Adobe Experience Manager allows the user to perform the action on this page or on any child pages.	Adobe Experience Manager does not allow the user to perform the action on this page or 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 Content Repository 2.3 interface (Wildcard in the path of ACLs, currently can only be created in Content Repository.)

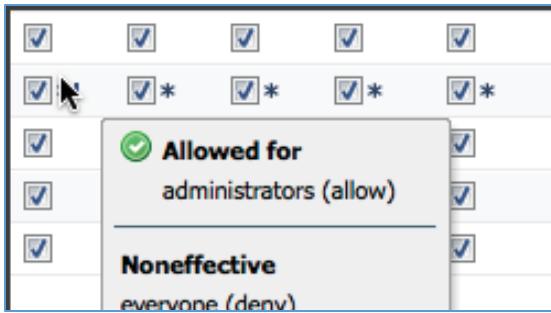
For an action at a given path:

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

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

Upper part List the effective entries.

Lower part	List the non-effective 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. Create new paragraphs on the page or on any child page. At the JCR level, users can modify a resource by modifying its properties, locking, visioning, and nt-modifications, and they have complete write permission on nodes defining a jcr:content child node, for example, cq:Page, nt:file, and cq:Asset.
Create	The user can create a new page or child page. If modify is denied, the subtrees below jcr:content are specifically excluded as the creation of jcr:content and its child nodes are considered 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 sub-trees 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 ACL of the page or child pages.
Edit ACL	The user can modify the ACL of the page or any child pages.
Replicate	The user can replicate content to another environment (for example, the Publish instance). The privilege is also applied to any child pages.

ACLs and How They Are Evaluated

Adobe Experience Manager WCM uses ACLs to organize the permissions being applied to the various pages. ACLs are made up of individual permissions and are used to determine the order in

which these permissions are applied. The list is formed per 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.

Concurrent Permission on ACLs

When two concurrent (and opposing) permissions are listed on the same ACL for the same resource, the ACL applied for such a resource is the one at the bottom.

Suppose, you have the following ACL for the same resource under /content/we-retail/us/en/products.

If a user is part of the two groups 'allowed-it' and 'restricted-it', you can see that the access to the page products is denied because the ACL deny in read access is the rule at the bottom.

Now, if the order of the ACL is the opposite, a user, part of two groups, allowed-it and restricted-it, will have access to the products page (because the ACL allow in read access is the rule at the bottom).

LAB L- WORK WITH ACLS

Scenario

A Project Manager wants to assign work (pages) to their team without involving IT. And once the work is completed, the page will then go through an approval process. If it's approved it will be published, otherwise it will go back to the editor with comments, for a proper feedback loop. A typical author has read access to the website and write permissions are only given on request (when work is assigned). Authors should only have access to the pages they are currently assigned.

Solution: Create a custom AEM Project for each PM that allows them to add team members for editing and reviewing. When new work needs to be assigned:

1. The PM will start a project workflow instance with the work (or payload) and complete details of the work to be done. The PM will also specify the editor that is assigned to the work.
2. The workflow will then assign edit permissions to the editor and notify the editor with a task that contains the details of the work to be done.
3. Once the editor has completed their task, a new task is assigned to the Reviewers group. If they approve the work done, the page is published. If they deny the work, they can comment and the workflow will be stepped back to the editor to make the proper changes.

Objectives

- Create a new user
- Create a new group
- Add a user to a group
- Create and participate in custom AEM Project

Prerequisites

- AEM installed on your machine:
 - Running Adobe Experience Manager Author instance
 - An Eclipse project from the AEM Archetype

Directions

Complete the exercises that follow.

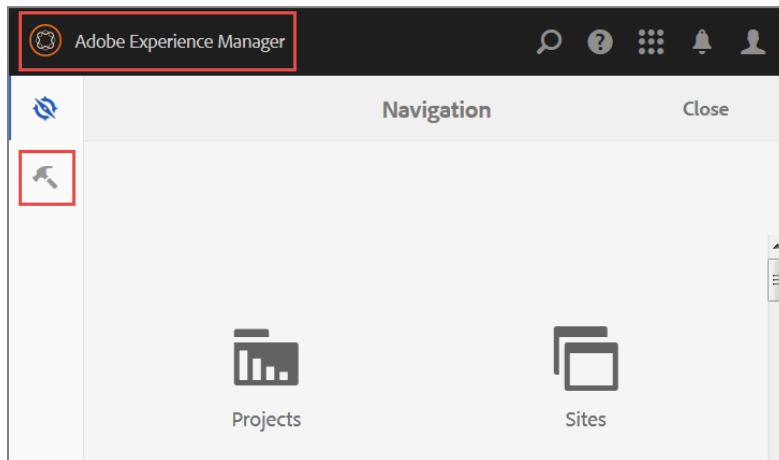
Exercise 1 – Create a New User

Overview

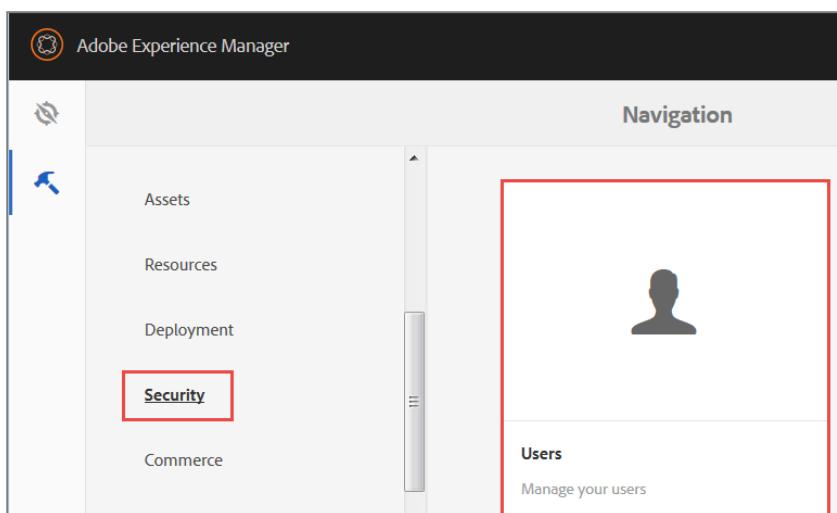
In this exercise, you will add yourself as a user in Adobe Experience Manager.

Steps

1. Navigate to the Navigation console, or click the link: <http://localhost:4502>
2. Click **Adobe Experience Manager** in the upper left corner of the screen, and then click the **Tools** icon.



3. Click **Security > Users**. The **User Management** page is displayed.



4. Click **Create User** from the actions bar. The **Create New User** wizard is displayed.

The screenshot shows the 'User Management' page in Adobe Experience Manager. At the top right is a blue 'Create User' button. Below it is a table with three columns: 'Name', 'Login Name', and 'Job Title'. There are three rows of data:

Name	Login Name	Job Title
Aaron McDonald	aaron.mcdonald@mailinator.com	
account-manager	account-manager	
activity-service	activity-service	

5. Enter the your details, such as ID, email address, password, first name, and last name.

The screenshot shows the 'Create New User' wizard. At the top right are 'Close' and 'Save' buttons. The main area is titled 'User Details' and contains the following fields:

ID *	cgrant	Email Address	cgrant@adobe.com
Password *	*****	Retype Password *	*****
First Name	Chuck	Last Name	Grant
Phone Number		Job Title	

6. Click **Save**.

The success message is displayed on the **User Management** page, as shown below:

The screenshot shows the 'User Management' page again. A green success message at the top says 'SUCCESS You have successfully saved changes to user cgrant.' The table now has four rows, including the new user 'Chuck Grant' which is highlighted with a red box.

Name	Login Name	Job Title
Carlene Avery	carlene.javery@mailinator.com	Custom inspector
Chuck Grant	cgrant	

The user that you have created will be added to the users list.

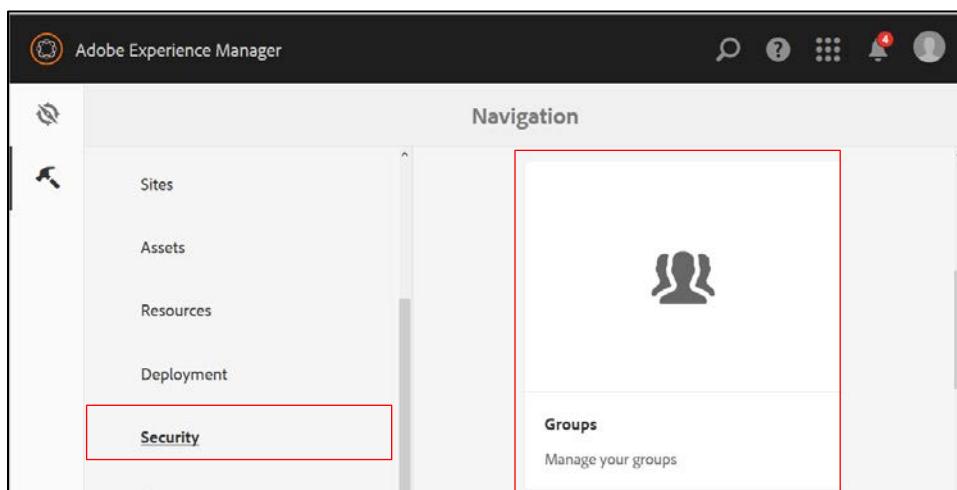
Exercise 2 – Create a New Group

Overview

In this exercise, you will create a new group named Legal.

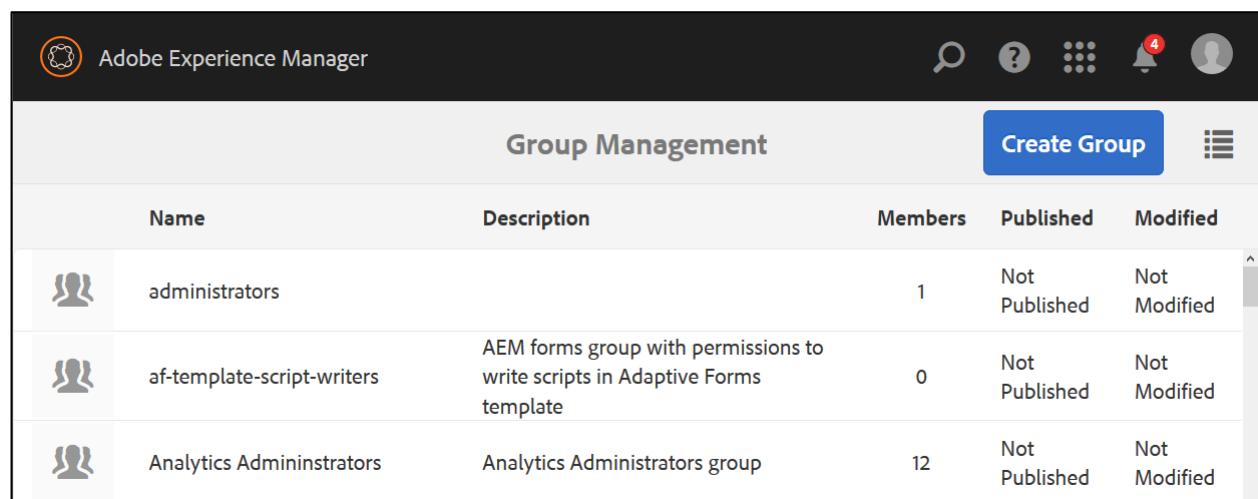
Steps

1. Navigate to the Navigation console, or click the link: <http://localhost:4502>
2. Click **Adobe Experience Manager** in the upper left corner of the screen, and then click the **Tools** icon.
3. Click **Security > Users**. The **Groups** page is displayed.



The screenshot shows the Adobe Experience Manager navigation console. The left sidebar has links for Sites, Assets, Resources, Deployment, and Security. The Security link is highlighted with a red box. The main content area is titled "Navigation" and shows a "Groups" section with a user icon and the text "Manage your groups". A red box highlights the "Groups" section.

4. Click **Create Group** from the actions bar. The **Create New Group** wizard is displayed.



The screenshot shows the "Group Management" page. The header includes a "Create Group" button. The main table lists three groups:

Name	Description	Members	Published	Modified
administrators		1	Not Published	Not Modified
af-template-script-writers	AEM forms group with permissions to write scripts in Adaptive Forms template	0	Not Published	Not Modified
Analytics Administrators	Analytics Administrators group	12	Not Published	Not Modified

5. Enter the group details such as id and name.

The screenshot shows the 'Create New Group' dialog box. It has two main sections: 'Group Details' and 'Add Members to Group'. In the 'Group Details' section, there are three input fields: 'ID *' containing 'legal', 'Name' containing 'legal', and 'Description' containing 'Legal'. The 'Description' field is currently selected. In the 'Add Members to Group' section, there is a single input field with the placeholder 'Select user or group'. At the top right of the dialog box are 'Cancel' and 'Save' buttons.

6. Click **Save**.

The success message is displayed on the **Group Management** page, as displayed below:

The screenshot shows the 'Group Management' page. At the top, there is a success message: 'SUCCESS You have successfully saved changes to group legal'. Below this, there is a table with the following data:

Name	Description	Members	Published	Modified
administrators		1	Not Published	Not Modified
af-template-script-writers	AEM forms group with permissions to write scripts in Adaptive Forms template	0	Not Published	Not Modified

The group that you created will be added to the groups list.

The screenshot shows the 'Group Management' page in Adobe Experience Manager. At the top, there is a navigation bar with icons for search, help, and notifications (with a red '4' badge). Below the header, the title 'Group Management' is centered above a blue 'Create Group' button. The main content is a table with the following data:

Name	Description	Members	Published	Modified
forms-xfa-users	AEM forms user group having permissions to upload XDP	1	Not Published	Not Modified
legal	Legal	0	Not Published	a minute ago Administrator
mp-contributors	Media Publishing Contributors	0	Not Published	Not Modified
mp-designers	Media Publishing Designers	0	Not Published	Not Modified

Exercise 3 – Add an User to a Group

Overview

In this exercise, you will add yourself to the **Legal** group.

Steps

1. Navigate to the **Navigation** console, or click the link:<http://localhost:4502>
2. Click **Adobe Experience Manager** in the upper left corner of the screen, and then click the **Tools** icon.
3. Click **Security > Groups**. The Group Management page is displayed.
4. Look for the **legal** group from the list, and then click the **Legal. Edit Group Settings** wizard appears.
5. In the **Add Members to Group** section, select your **ID**. For example, select **cgrant** from the drop-

The screenshot shows the 'Edit Group Settings' dialog box. At the top, there are buttons for 'Copy Group', 'Activate', 'Deactivate', 'Cancel', and a blue 'Save' button. Below these are fields for 'ID *' (containing 'legal'), 'Name' (containing 'legal'), and 'Description' (containing 'Users of the Legal Group'). Under the heading 'Add Members to Group', there is a search input field containing 'chu' and a list below it. The list shows a user profile for 'Chuck Grant' with the ID 'cgrant'. The entire dialog box is enclosed in a light gray border.

down list, and click **Save**.

The success message that confirms the user (**Chuck Grant**) is added to **Legal** group is displayed on the **Group Management** page, as shown below:

Name	Description	Members	Published	Modified
Authors	The group responsible for content editing.	14	Not published	
context-simulation	Members of this group will be able to use the ClientContext and the ContextHub to simulate users that this group can read.	2	Not published	
Contributors	Base group for all user and groups that must be able to access the authoring environment.	50	Not published	
DAM Users	Users of the DAM system	2	Not published	a few seconds ago Administrator
DSC	DSC Show Berlin	18	Not published	

Let us add **Legal** group to **Contributors** group which grants access to the foundation client libraries.

6. Look for the **Contributors** group from the list, and then click the **Contributors. Edit Group Settings** wizard appears.
7. In the **Add Members to Group** section, select the **Legal** group and click **Save**
8. Click **Adobe Experience Manager** on the upper left corner of the screen, and then click the **Tools** icon.
9. Click Security > Permissions..
10. Double-click your ID (for example, cgrant) in the left pane to view various properties, groups, and permissions associated with the user in the right pane.
11. From the right pane, click the **Groups** tab. You will notice the user is given access to the **Legal** and **Contributors** groups.
12. Open the **Permissions** tab.

The screenshot below shows the weretail folder under "apps" with "read" access selected:

The screenshot displays the AEM Security interface. On the left, a list of users and groups is shown, with 'cgrant' (Chuck Grant) selected. On the right, the 'Permissions' tab is active, showing a detailed view of access rights for various paths. The 'Path' column lists directory structures starting from '/libs'. The 'Read' column contains checkboxes, many of which are checked for the 'apps' and 'weretail' folders. Other columns include 'Modify', 'Create', 'Delete', 'Read A...', 'Edit ACL', and 'Replicate'. A vertical 'Details' link is present next to each row.

Path	Read	Modify	Create	Delete	Read A...	Edit ACL	Replicate
/libs	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
apps	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
community-comp...	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
core	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
cq	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
dam	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
public	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
settings	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
sling	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
social	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
wcm	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
we-retail-client-app	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
we-retail-commu...	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
we-retail-screens	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
we-unlimited-app	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
weretail	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
bin	<input checked="" type="checkbox"/>	<input type="checkbox"/>					

Exercise 4 – Create and Participate in a Custom AEM Project

Overview

In this exercise, you will automate ACLS using the java class named AutoAssignACL and test the approval process. We will accomplish this in 5 parts:

Part 1: Create a custom workflow process

Part 2: Increase permissions on the workflow process user

Part 3: Create a custom workflow

Part 4: Create a project template

Part 5: Testing the business process with dynamic permissions.

Part 1 – Create a Custom Workflow Process

Steps:

1. Create a class file named **AutoAssignACL** under the the training.core package at ***training.core > src/main/java > com.adobe.training.core***.

- a. Copy the following code into the class (\Exercise Files\13_Users_Groups_Permissions\AutoAssignACL.java):

```

1. package com.adobe.training.core;
2. import javax.jcr.security.AccessControlList;
3. import javax.jcr.security.AccessControlManager;
4. import javax.jcr.security.Privilege;
5.
6. import org.apache.felix.scr.annotations.Component;
7. import org.apache.felix.scr.annotations.Properties;
8. import org.apache.felix.scr.annotations.Property;
9. import org.apache.felix.scr.annotations.Service;
10. import org.apache.jackrabbit.api.JackrabbitSession;
11. import org.apache.jackrabbit.api.security.user.Authorizable;
12. import org.apache.jackrabbit.api.security.user.UserManager;
13. import org.apache.jackrabbit.commons.jackrabbit.authorization.AccessControlUtils
14. import org.osgi.framework.Constants;
15. import org.slf4j.Logger;
16. import org.slf4j.LoggerFactory;
17.
18. import com.day.cq.workflow.WorkflowException;
19. import com.day.cq.workflow.WorkflowSession;
20. import com.day.cq.workflow.exec.WorkItem;
21. import com.day.cq.workflow.exec.WorkflowData;
22. import com.day.cq.workflow.exec.WorkflowProcess;
23. import com.day.cq.workflow.metadata.MetaDataMap;
24.
25. /**

```

```

26. * This is a project workflow process step. When starting a workflow from a proj
ect, you can specify the:
27. *
28. * Payload: The page that will have 'modify' permissions added/removed from its
ACL
29. * User: The user attached to the ACL
30. *
31. * This process will add/remove jcr privileges that correlate to 'Modify, Create
, Delete' in the /useradmin
32. * for the given user on the payload. An argument is used to specify to add or r
emove 'modify' permissions.
33. * The argument must follow permission=<add || remove>
34. *
35. * @author Kevin Nennig (nennig@adobe.com)
36. */
37. @Component
38. @Service
39. @Properties({ @Property(name = Constants.SERVICE_DESCRIPTION, value = "A process
    to auto assign permissions to edit a given page."),
40.             @Property(name = Constants.SERVICE_VENDOR, value = "Adobe Digital Learni
ng Services"),
41.             @Property(name = "process.label", value = "Auto Assign Edit Permissions"
) })
42. public class AutoAssignACL implements WorkflowProcess{
43.     private final Logger logger = LoggerFactory.getLogger(getClass());
44.
45.     /**
46.      * @param item - Holds the payload and user
47.      * @param workflowSession - Session with service user (workflow-process-
service) that will be modifying the permissions
48.      * @param workflowArgs - permission=add will add 'modify' permissions. permi
ssion=remove will remove 'modify' permissions
49.     */
50.     @Override
51.     public void execute(WorkItem item, WorkflowSession workflowSession, MetaData
Map workflowArgs) throws WorkflowException {
52.
53.         WorkflowData workflowData = item.getWorkflowData(); //get the workflow p
roperties
54.         String userID = workflowData.getMetaDataMap().get("assignee", String.cla
ss);
55.         logger.info("User to add permissions: " + userID);
56.         String payload = workflowData.getPayload().toString();
57.         logger.info("Content path to add permissions: " + payload);
58.
59.         UserManager uM;
60.         try {
61.             JackrabbitSession jcrSession = (JackrabbitSession)workflowSession.ge
tSession();
62.             //The user that actually modifies the permissions is the workflow-
process-service
63.             //The workflow-process-
service must have Read ACL and Write ACL permissions for the payload
64.             logger.info("Service user to change permissions: " + jcrSession.getU
serID());
65.
66.             uM = jcrSession.getUserManager();
67.             //Get the Authorizable object for the new user
68.             Authorizable authorizable = uM.getAuthorizable(userID); //this might
be a user or a group
69.

```

```

70.         AccessControlManager accessControlManager = jcrSession.getAccessCont
    rolManager();
71.
72.         //JCR privileges that encompass AEM Permissions: "Modify, Create, De
    lete" in the /useradmin
73.         Privilege[] privileges = {accessControlManager.privilegeFromName(Pri
    vilege.JCR MODIFY_PROPERTIES),
74.             accessControlManager.privilegeFromName(Privilege.JCR_LOCK_MA
    NAGEMENT),
75.             accessControlManager.privilegeFromName(Privilege.JCR_VERSION
    _MANAGEMENT),
76.             accessControlManager.privilegeFromName(Privilege.JCR_REMOVE_
    CHILD_NODES),
77.             accessControlManager.privilegeFromName(Privilege.JCR_REMOVE_
    NODE),
78.             accessControlManager.privilegeFromName(Privilege.JCR_ADD_CHI
    LD_NODES),
79.             accessControlManager.privilegeFromName(Privilege.JCR_NODE_TY
    PE_MANAGEMENT)};
80.
81.         //Get the ACL for the payload
82.         AccessControlList acl = AccessControlUtils.getAccessControlList(jcrS
    ession, payload);
83.         //Add the user/privileges to the payload ACL
84.         acl.addAccessControlEntry(authorizable.getPrincipal(), privileges);

85.
86.
87.         //Based on the process step arguments, permissions are either added
    or removed
88.         //Assumes arguments are given as: permission=add
89.         String arguments = workflowArgs.get("PROCESS_ARGS", "");
90.         if(arguments.contains("permission")){
91.             String permission = arguments.split("=")[1];
92.             if(permission.equals("add")){
    //Adds permissions to edit the pay
    load
93.                 logger.info("Adding permissions");
94.                 accessControlManager.setPolicy(payload, acl);
95.                 logger.info("Added modify permissions to " + payload + " for
    " + userID);
96.             } else if(permission.equals("remove")){
    //Removes permissions to
    edit the payload
97.                 logger.info("Removing permissions");
98.                 accessControlManager.removePolicy(payload, acl);
99.                 logger.info("Removed modify permissions to " + payload + " f
    or " + userID);
100.            }
101.        } else{
    //do nothing if the workitem argument is not set
102.            logger.info("No arguments given for workitem");
103.        }
104.
105.
106.        jcrSession.save();
107.    } catch (Exception e) {
108.        logger.error(e.getMessage(), e);
109.        e.printStackTrace();
110.    }
111.}
112.}

```

b. Examine the code

Note that based on the process step argument received, the permission is either added or removed.

- c. Save the changes

Part 2 – Increase Permissions on the Workflow Process USER

Overview

Because our java class implements the Workflow Process service, the workflow-process-service service user is the user that updates permissions for our editors. To allow the workflow-process-service to do this, we need to give it Read ACL and Edit ACL permissions.

1. Open URL <http://localhost:4502/useradmin>

2. Find the user named **workflow-process-service**

The screenshot shows the AEM Security interface with the title bar 'AEM | Security'. Below it is a search bar with the text 'workflow'. A table lists various users with columns for 'T...', 'ID', 'Name', 'Pub.', and 'Mod.'. The user 'workflow-process-service' is highlighted with a green background, indicating it is selected.

T...	ID	Name	Pub.	Mod.
	communities-workflow-lau...	Communities W...		
	wcm-workflow-service	wcm-workflow-s...		
	workflow-administrators	workflow-admini...		
	workflow-editors	workflow-editors		
	workflow-process-service	workflow-proces...		
	workflow-repo-reader-servi...	workflow-repo-r...		
	workflow-service	workflow-service		
	workflow-user-service	workflow-user-s...		
	workflow-users	workflow-users		

3. Select the **Permissions** tab for the workflow-process-service user

4. Check Read ACL and Edit ACL

The screenshot shows the user properties interface for 'workflow-process-service'. The 'Permissions' tab is selected. Below it is a table with columns for 'Path', 'Read', 'Modify', 'Create', 'Delete', 'Read ACL', 'Edit ACL', and 'Replicate'. The 'Read ACL' and 'Edit ACL' columns contain checkboxes, many of which are checked. The table includes rows for 'META-INF', 'apps', and 'bin'.

Path	Read	Modify	Create	Delete	Read ACL	Edit ACL	Replicate
META-INF	<input checked="" type="checkbox"/> *	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> *			
apps	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
bin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

➤ Save the changes

Part 3 – Create a Custom Workflow



Note: Part 3 and Part 4 show how to create a custom workflow and a custom project template for this exercise. For time purposes, we have provided a content package in the Exercises_Files (*automate-acls-part3-part4.zip*) that contains Part 3 and 4 completed. Depending on time constraints you can either install this content package or run through Part 3 and 4 on your own. If you install the content package please move on to Part 5.

Overview:

Now that we have our java class setup and permissions increased, we need to create a workflow to use our workflow process. The workflow will:

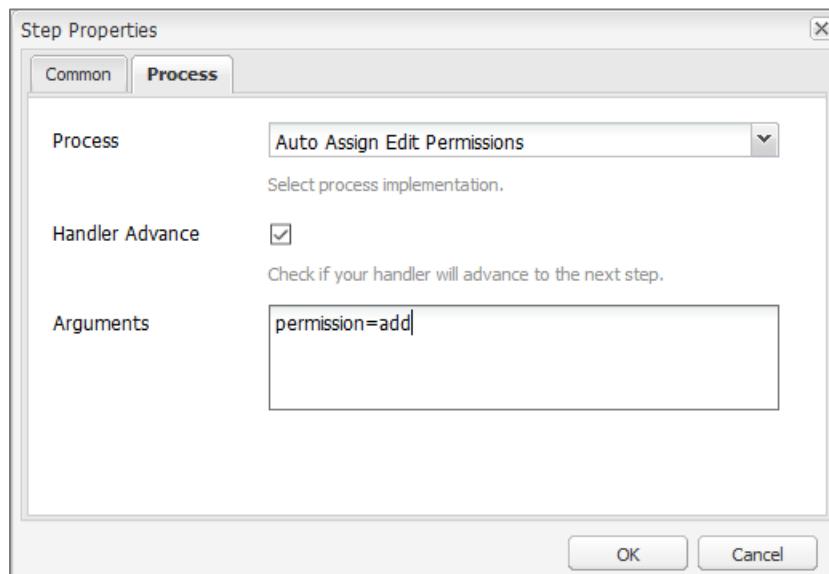
- Auto assign edit permissions to the user to whom the workflow is assigned to
- Assign a project task to the editor
- Assign an approval task to the reviewer of the project
- Add a GOTO step that will conditionally move the workflow forward or step it back to the editor
- Auto remove edit permissions to whoever the workflow is assigned to
- Auto publish the page

If you have installed *automate-acls-part3-part4.zip*, you can skip this section and move onto Part 5.

Steps:

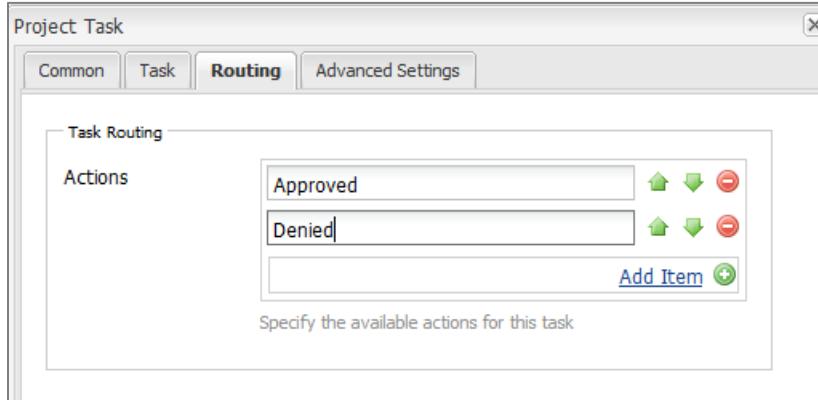
1. Create a new workflow by navigating to *Tools > Workflow > Models*
 - a. Click Create > Create Model
 - Title: Dynamic Editing, Approval and Publish
 - Name: dynamicediting
 - b. Click **Done**
 - c. Open the workflow by selecting the new workflow and clicking **Edit**
2. Drag and drop Process Step from the Workflow section to the area with the dotted line “Drop steps or participants here”
 - a. Right-click and Edit the Process Step.

- b. Provide the step properties: Title and Description.
 - Title: Add Edit Permissions
- c. Select the *Process* tab:
 - Process: Auto Assign Edit Permissions (created when you added the AutoAssignACL.java class)
 - Handler Advance: checked
 - Arguments: **permission=add**



- d. Click **OK**
3. Add "Create Project Task" from the Projects section
- Title: Edit the Page
 - Name: Edit the Page (on the second tab)
 - Task Priority: High
 - Days: 2
 - Pre-create Task Script: /etc/workflow/scripts/projects/simpleTaskConfiguration.ecma (Advanced Settings tab)
4. Add "Create Role Based Project Task" from the Projects section
- Title: Approval
 - Name: Approval

- Task Priority: High
- Days: 2
- Actions: ["Approved","Denied"] (Separate entries on the Routing tab)



5. Add "Goto Step" from the Workflow section

- Title: Goto Step
- Description: Go back to "Edit the Page" if not approved
- The step to go to: Edit the Page
- Script: function check() {

```
6. if(workflowData.getMetaDataMap().get("lastTaskAction","","") == "Approved") {
7.   return false
8. }
9. return true;
```

Note: Go to the Exercises_Files and copy and paste the approver-script.js for the approval logic.

6. Add another Process step from the Workflow section.

- Title: Remove Edit Permissions
- Select the Process tab:
- Process: Auto Assign Edit Permissions
- Handler Advance: checked
- Arguments: permission=remove

7. Add "Activate Page/Asset" from the WCM Workflow section.



8. Click Save in the toolbar to save the workflow.

Part 4: Create a Project Template.

Overview:

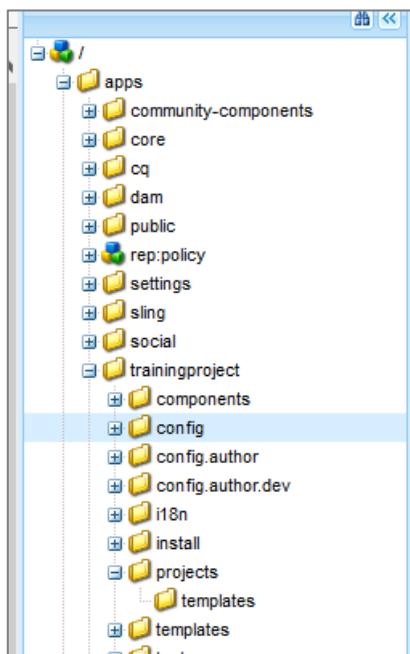
Now that our project workflow is created, we need to create a custom AEM project that supports our workflow. We will:

- Copy a simple project template
- Remove any unnecessary tiles
- Add our custom workflow
- Add a reviewers group

If you have installed automate-acls-part3-part4.zip you can skip this section and move onto Part 5.

Steps:

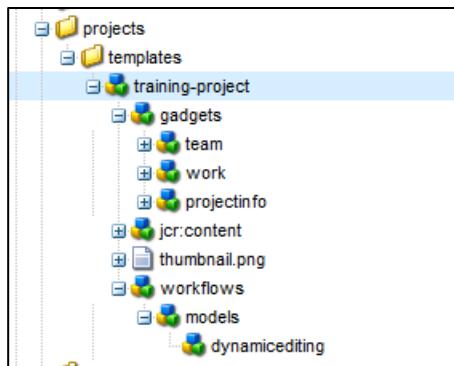
1. In CRXDE lite, create the struture trainingproject/projects/templates as shown:



a. Save All

2. Copy /libs/cq/core/content/projects/templates/**default** to trainingproject/projects/**templates**

a. Rename to training-project



- Title: ACL Automation Project
- Description: Creates an ACL Automation Project

	Name	Type	Value
1	includeInCreateProject	Boolean	true
2	jcr:created	Date	2017-04-03T14:06:52.005-07:00
3	jcr:createdBy	String	admin
4	jcr:description	String	Creates an ACL Automation Project
5	jcr:primaryType	Name	cq:Template
6	jcr:title	String	ACL Automation Project

- a. Save All
3. Delete the following:
 - gadgets/experiences
 - gadgets/asset
 - Workflows/models/launch
 - Wokflow/models/landing
 - Workflow/models/email
- a. Save All
4. Rename workflows/models/sampleworkflow to **dynamicediting**

- a. Edit the properties:
 - modelId=/etc/workflow/models/dynamicediting/jcr:content/model
- b. Save All

Properties		Access Control	Replication	Console	Build Info
	Name ▲	Type	Value		
1	jcr:primaryType	Name	nt:unstructured		
2	modelId	String	/etc/workflow/models/dynamicediting/jcr:content/model		
3	wizard	String	/libs/cq/core/content/projects/workflowwizards/sample_team.html		

5. Create a node ~/training-project/roles [nt:unstructured]
6. Copy /libs/cq/core/content/projects/templates/retail-product-photoshoot/roles/reviewer to ~/roles
7. Save All

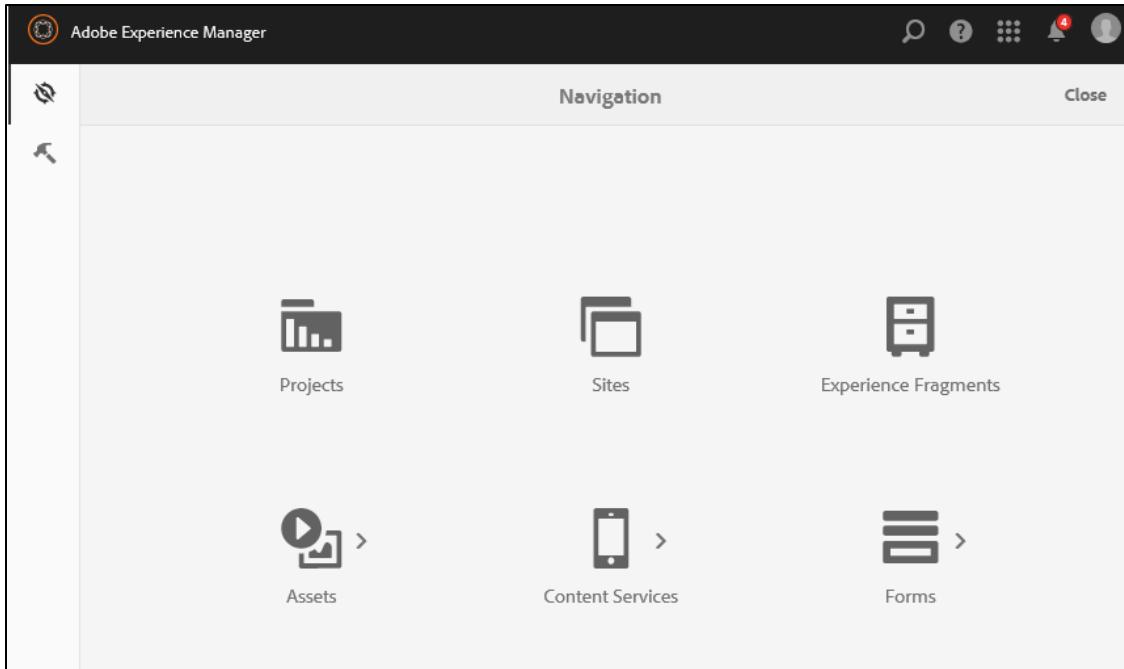
Part 5: Testing the Business Process with Dynamic Permissions

Overview:

We now have our business process created that was outlined by our PM. We have an AEM Project template that uses our custom workflow that implements our auto assign ACL java class. In this last step we will run through the AEM Project as the Project Manager.

Steps:

1. Before we create and run through our custom AEM project, observe that Chuck Grant has only read permissions to the We.Retail Site. Impersonate Chuck Grant and open <http://localhost:4502/editor.html/content/we-retail/language-masters/en/men.html> and observe he has only read permissions.
2. Revert back to the admin
3. Navigate to Projects area
 - a. Go to <http://localhost:4502/projects.html>



4. Select Create > Project

- a. Choose ACL Automation Project and click **Next**
 - Title: Page Editing and Approval
 - Start Date: <Specify a date>
 - Due Date: <Specify a date>
 - User: **Chuck Grant** (Select Editors from drop-down list)
 - Click **Add**
 - Add another user:
 - User: **Carlene Avary** (Select Reviewers from drop-down list)

Basic Advanced

Title *

Description

Start Date

Due Date

User

Members

	Administrator	Owners
	chuck Grant	cgrant@adobe.com
	Carlene Avery	cavery@we-retail.net

	Editors	Reviewers	Remove
			<input type="button" value="Delete"/>

8. Click Create and select Open in the dialog box

9. On the Workflows tile, click Add Work

Adobe Experience Manager

Team (3)

--	--	--

Workflows (0)

Add Work

Project Info

	Administrator
Editing and Approval	
<input type="button" value="Clock"/>	Active
Started on Apr 22	

10. In the Start Workflow screen, Select Dynamic Editing, Approval, and Publish and click Next

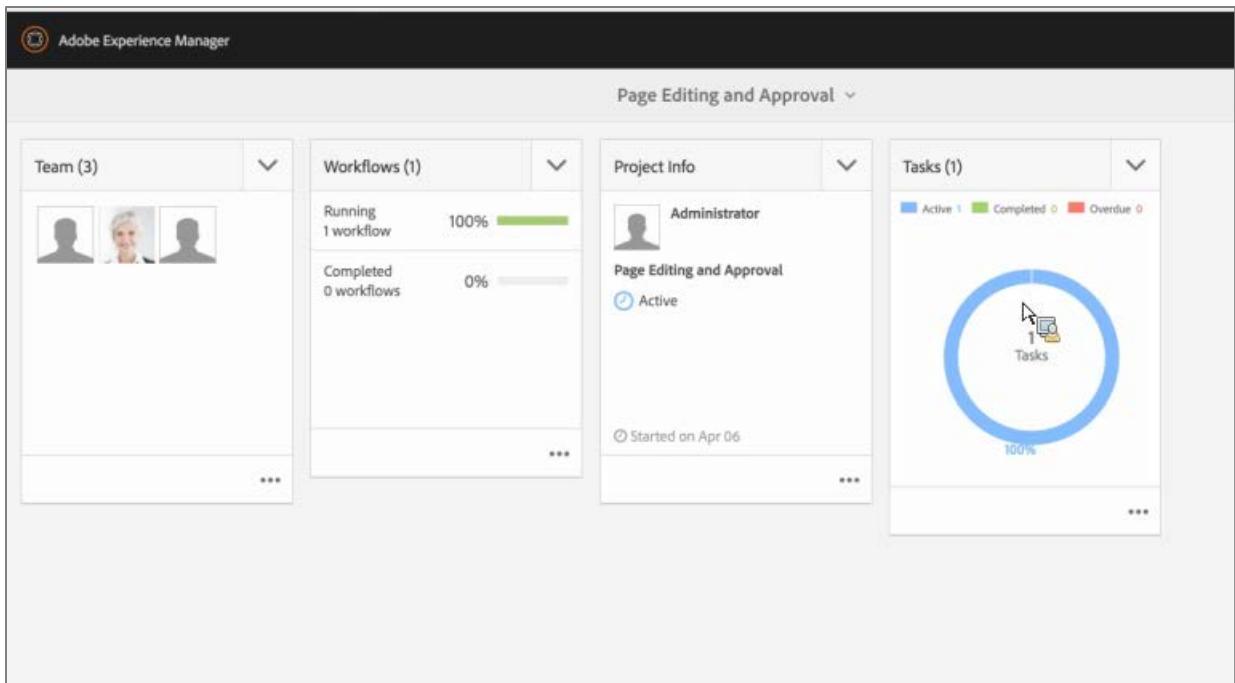
a. Enter the values as shown:

The screenshot shows a form for creating a workflow task. The fields are as follows:

- Title ***: Edit the Mens Page
- Assign To ***: chuck Grant
- Description**: (empty text area)
- Content Path**: /content/we-retail/language-masters/en/men
- Task Priority**: Medium
- Due Date**: (empty date input field with a calendar icon)

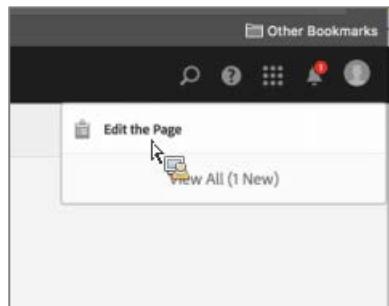
b. Click Submit

11. Refresh the browser and notice the new task



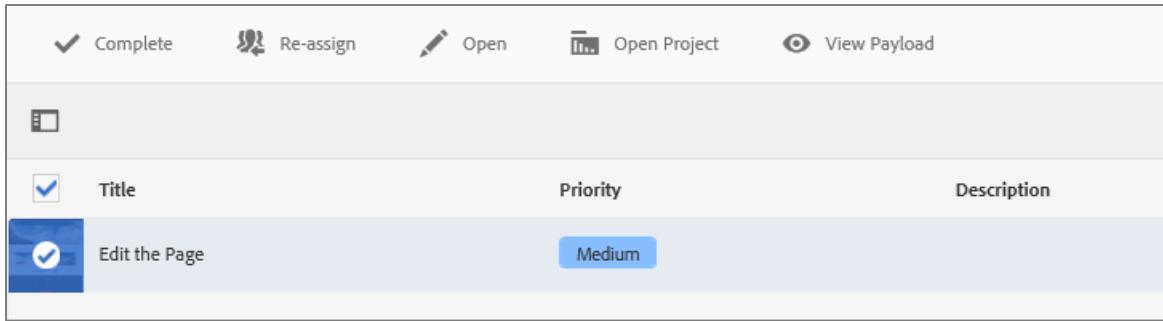
12. Open another browser and login as Chuck user

a. Find the new message in the inbox



13. Click click "View All (1 New)"

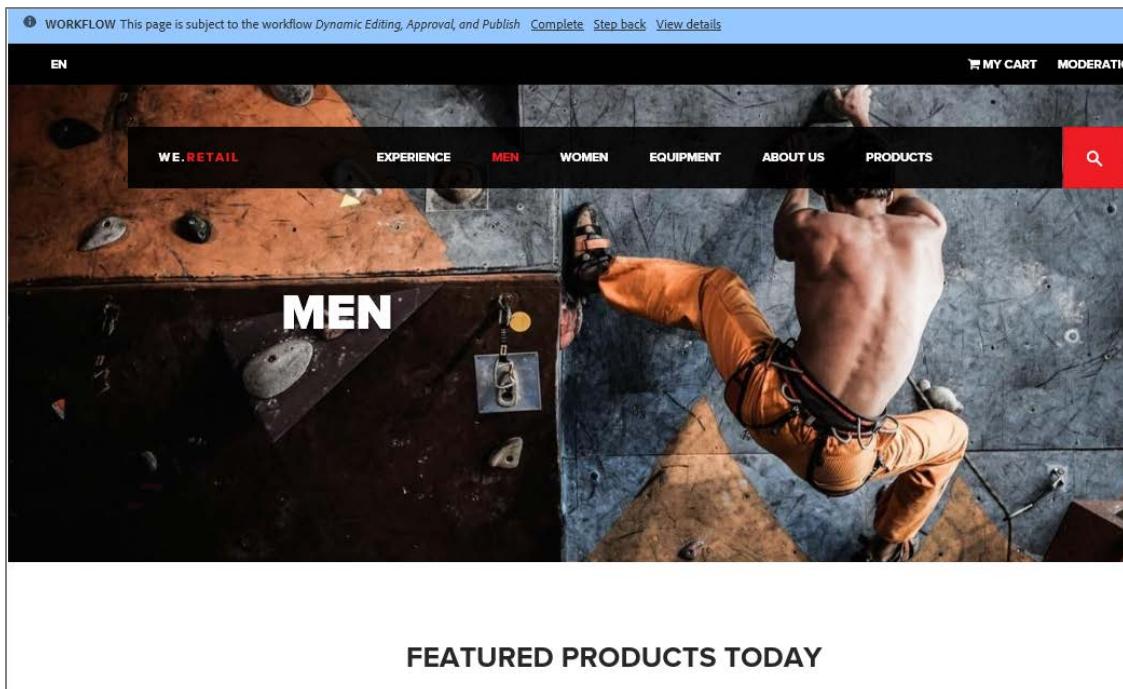
14. Select the Edit Page and click View Payload



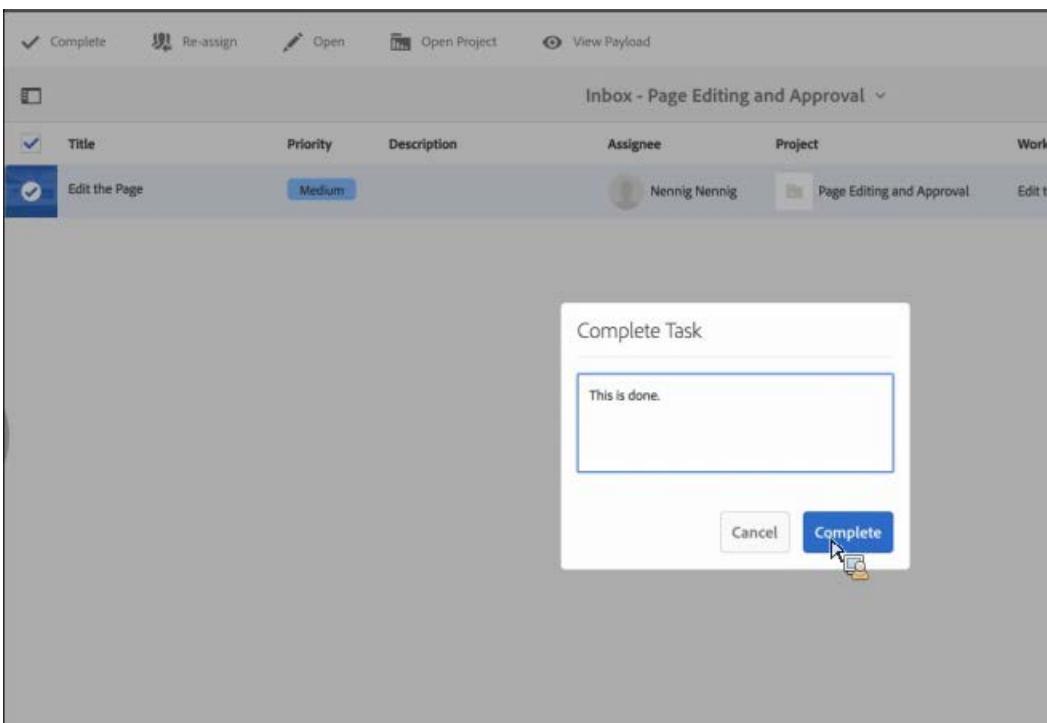
15. Notice the page is in the workflow and you have the option to edit the page:

A screenshot of the AEM page editor. At the top, a blue header bar displays the text 'WORKFLOW This page is subject to the workflow Dynamic Editing, Approval, and Publish' with links for 'Complete', 'Step back', and 'View details'. Below the header is the main content area, which shows a dark-themed website for 'WE.RETAIL'. The website features a navigation bar with 'WE.RETAIL', 'EXPERIENCE', 'MEN', 'WOMEN', 'EQUIPMENT', 'ABOUT US', and 'PRODUCTS'. A large image of a shirtless man climbing a wall is the central visual. On the left side of the image, the word 'MEN' is prominently displayed in white. At the bottom of the content area, there is a toolbar with various edit icons. Below the toolbar, the text 'FEATURED PRODUCTS' is visible, indicating the current component being edited. The entire interface is set against a light gray background.

9. Click on the Title component called "Featured Products". Notice all Edit actions are now enabled. Click on the Wrench icon.
- Make changes to the Title and click Done
 - Verify the page has updated with the changes you made.



10. Go back to the Inbox, Select Edit the Page, and Click Complete to complete the task.



- a. Enter a comment and Complete
11. Revert back to the admin user
12. Impersonate Carlene Avary

a. Notice there is a new message in her inbox. Click "View All (1 New)"

b. Notice there is an Approval Task

The screenshot shows the Adobe Experience Manager inbox interface. At the top, there are buttons for 'Complete', 'Re-assign', 'Open', 'Open Project', and 'View Payload'. Below this is a table titled 'Inbox' with columns: Title, Priority, Description, Assignee, and Project. There are five tasks listed:

- Approval Task**: Priority High. Description: You can opt in for Analytics and Targeting by selecting your configuration and then adding it to your pages.
- Configure Analytics & Targeting**: Priority Medium. Description: It is strongly recommended to use official AEM Security systems. Please review <https://adobe.com/guides>
- Apply the AEM Security Checklist**: Priority High. Description: It is highly recommended for production systems to use an admin user and use the Wizard to configure it.
- Enable Aggregated Usage Statistics Collection**: Priority Medium. Description: You can opt in for aggregated usage statistics collection by enabling it.
- Configure HTTPS**: Priority High. Description: It is highly recommended for production systems to use an admin user and use the Wizard to configure it.

A modal window titled 'Complete Task' is overlaid on the inbox. It has a dropdown menu labeled 'Select Action' with two options: 'Approved' and 'Denied'. The 'Approved' option is selected and highlighted with a cursor. At the bottom of the modal are 'Cancel' and 'Complete' buttons.

13. Click Complete and notice there is a dropdown with Approved and Denied.

If you select Denied, the workflow will step back and create new task for Chuck to re-edit the page with the comments of the Reviewer (Carlene). Otherwise if you select Approved, the workflow will move forward.

14. Complete the task by selecting Approved and add a comment

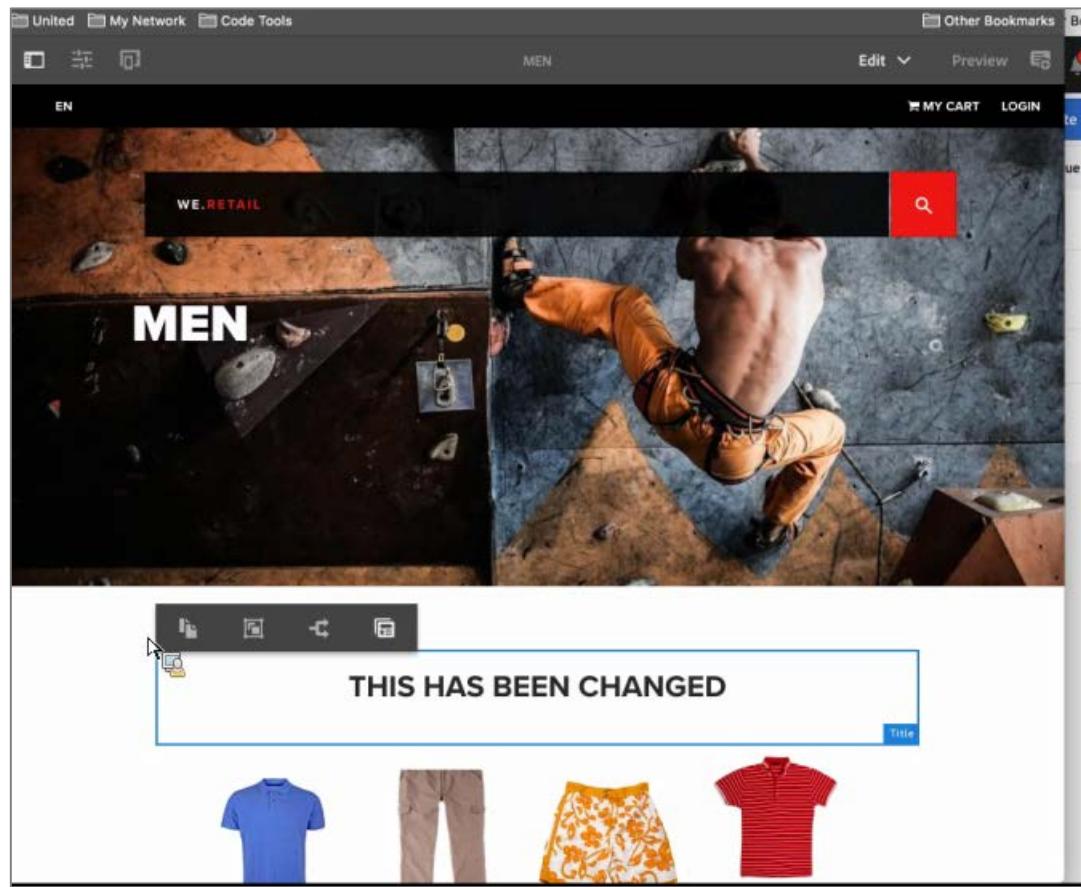
This screenshot shows the same inbox and task list as the previous one. The 'Complete Task' modal is still open, but now the 'Approved' option is selected in the dropdown. A text input field below the dropdown contains the comment 'This is great!'. The 'Complete' button is visible at the bottom right of the modal.



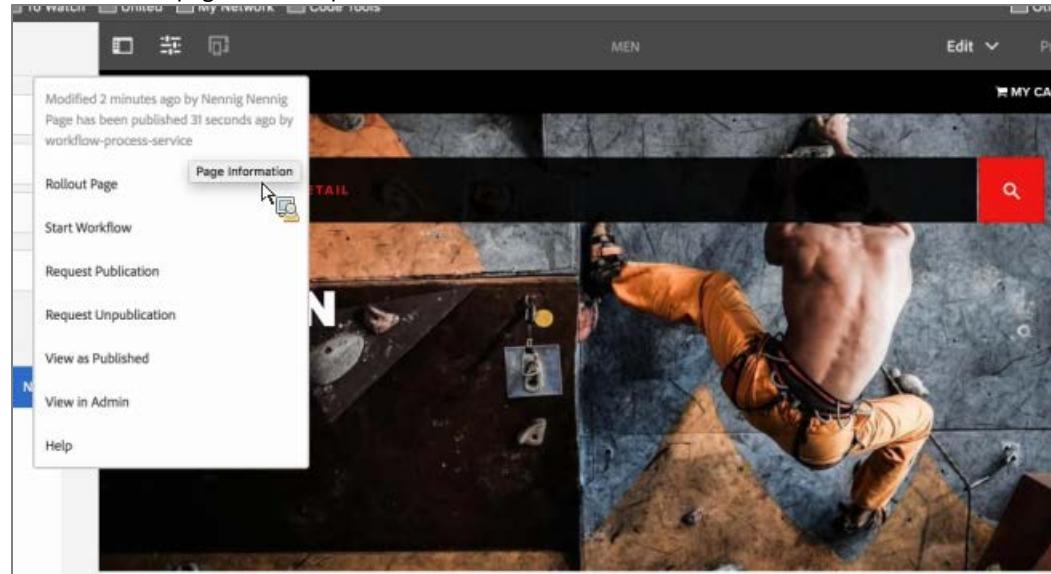
Note: At this point, the workflow runs through the remaining 2 process steps and completes. The first process, removed edit permissions to the editor (Chuck) and the second process published the page.

15. Impersonate Chuck Grant

16. Open the Men page (<http://localhost:4502/editor.html/content/we-retail/language-masters/en/men.html>) and notice you have only the read permissions since the edit permissions have been removed:



- a. Also, notice the page has been published:



17. Observe the log messages in the project-training.log file:

APPENDIX: VLT COMMAND LINE INSTALLATION

Using FileVault

The FileVault tool (VLT) is used to synchronize the content between CRX or Adobe Experience Manager and your local file system. This is achieved as VLT has a client-side code that issues HTTP commands to the JCR and a server-side code that outputs to the file system. VLT is a command line tool and can be used to perform normal repository operations such as check-in, check-out, as well as management and configuration operations.

Commonly Used FileVault Commands

The following are some of the most commonly used VLT commands.

Command	Description
vlt --version	Displays the version of the FileVault tool
vlt co	Performs a VLT check-out of the JCR repository to your local file system
vlt --force	Forces check-out to overwrite local files if they already exist
vlt up	Short for vlt update. Brings changes from the repository into the working directory
vlt ci	Short for vlt commit. Sends changes from your working copy to the repository

Installing and Configuring FileVault

VLT is available in the standard Adobe Experience Manager installation folder in the directory /crx-quickstart/opt/filevault.

Using FileVault to Synchronize Content with Server

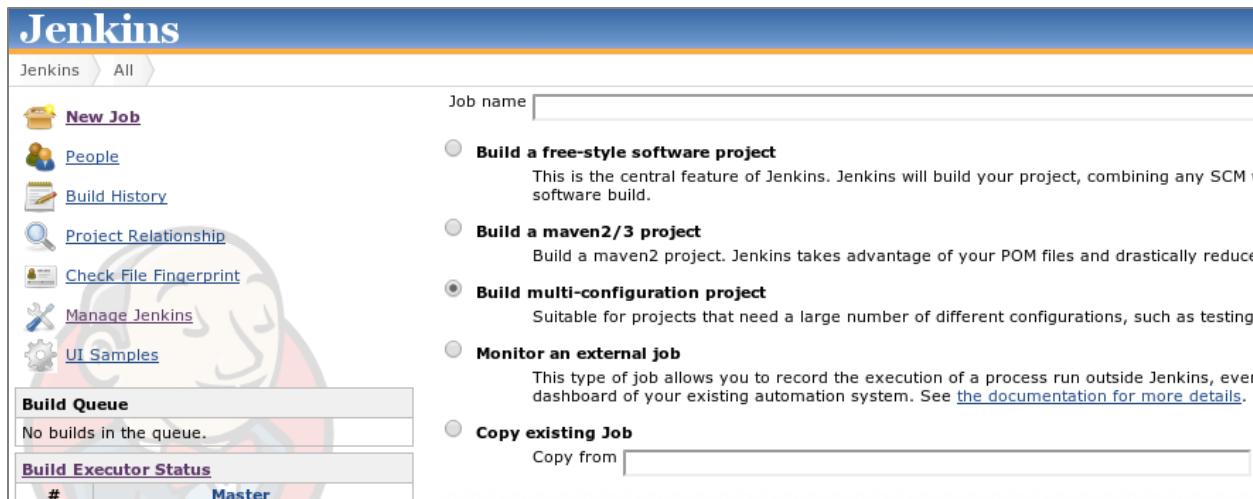
By performing a series of check-outs, updates, and commits, you can synchronize your changes between CRX and your local file system. To understand how content is mapped between the two, take a look at the following table:

CRX/Adobe Experience Manager	Local File System
nt:file	Rendered as files , with the data from jcr:content sub-nodes rendered as the contents of the file.

nt:folder	Rendered as directories
Any other	Rendered as directories , with their set of properties and values recorded in a special file called <code>.content.xml</code> within the directory of the node.

Collaborating with Teams

Most projects are a collaborative effort across various members of a team. Developers commit changes to their code base several times in a day. To make this effort run smoothly, you need to use an automated build and test suite that runs immediately after new deliveries are made. This type of Continuous Integration (CI) is possible using Jenkins.



The screenshot shows the Jenkins dashboard with a sidebar on the left containing links like 'New Job', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', and 'UI Samples'. The main area has a 'Job name' input field and a list of job creation options:

- Build a free-style software project**: This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with a software build.
- Build a maven2/3 project**: Build a maven2 project. Jenkins takes advantage of your POM files and drastically reduces configuration.
- Build multi-configuration project**: Suitable for projects that need a large number of different configurations, such as testing multiple database environments.
- Monitor an external job**: This type of job allows you to record the execution of a process run outside Jenkins, even if it's not part of the Jenkins dashboard. See [the documentation for more details](#).
- Copy existing Job**: Copy from [input field]

You can use Jenkins to create jobs that are triggered as soon as new code is delivered. Members are immediately informed of any build breakages or regressions. By using Jenkins, you can collaborate very closely with your team, and ensure that your core functionality is intact.

Appendix Lab - Generate project files for Eclipse

Overview:

You have configured Eclipse Luna and the plugin to make sure the changes are updated in the Adobe Experience Manager repository. This looks fine if the project was created in Eclipse itself; however, there might be a chance where you have to work on a project that was created using the Maven and command prompt. In this scenario, you must create Eclipse project files for Eclipse using the command prompt. Once that is done, you can import the project in Eclipse and convert it to a Maven project to resolve any dependency. After completing the following steps, you can work on that project in Eclipse without any issues.

Steps:

1. Copy and extract the contents of **training.zip** in a folder under Eclipse workspace as
..\\workspace\\training

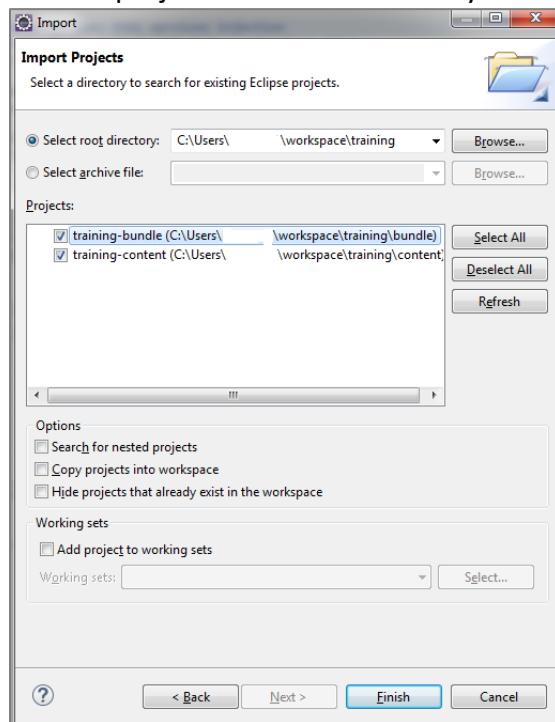
Name	Date modified	Type	Size
bundle	12/22/2015 1:30 PM	File folder	
content	12/22/2015 1:30 PM	File folder	
pom.xml	12/22/2015 1:30 PM	XML Document	10 KB
README.md	12/22/2015 1:30 PM	MD File	2 KB

2. Execute the following command at the project root folder (**workspace\\training**).

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

3. This will generate two files with extensions .project, which can be imported into Eclipse:
 - a. workspace\\training\\bundle\\.project
 - b. workspace\\training\\content\\.project
4. In Eclipse, to import the two project files, we need to do the following:
 - a. Navigate to **File > Import**.
 - b. In the **Import** dialog box, under the **General** option, select **Existing Projects into Workspace**.
 - c. Click **Next**.

- d. Click **Browse**, and then navigate to the **training** directory. Eclipse will find the two project files in this directory tree.



Now, we are ready to work on this project using Eclipse.

APPENDIX LAB: INSTALL AND CONFIGURE VLT ON YOUR SYSTEM

Overview:

You can synchronize code (both Java code and JSP code) in Eclipse workspace with the code in the Adobe Experience Manager JCR. For example, assume that you have application logic in Eclipse that represents a JSP component. You can synchronize the code in Eclipse with code in the Adobe Experience Manager JCR using the vault tool. That is, you can check-in code you write in Eclipse into the Adobe Experience Manager JCR.

Likewise, if you make a change in Adobe Experience Manager using CRXDE Lite, you can check-out the code that results in the code in Eclipse workspace (filesystem) being updated. To synchronize code, you need to configure the vault tool by following the steps below:

Steps:

1. Locate the VLT tool in the directory: **<AEM installation dir>/crx-quickstart/opt/filevault**

There, you will find two compressed files: **filevault.tgz** and **filevault.zip**. Copy the file that is most convenient to use on your system to a suitable directory (see below), and 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).
3. The directory **vault-cli-<version>** can be left in the current location or moved to another location on your system. In either case, ensure that you include the full path to the **vault-cli-<version>/bin** directory in your system path.
4. In order to have a persistent path, add the path to the environment variable (as described earlier), through **Control Panel > System > Advanced System Settings > Environment Variables**. Set the path to the bin folder. For example, C:\Users\Administrator\Desktop\Supported Applications\commons\filevault\vault-cli-3.1.16\bin
5. If you do not have access to the system settings, you can set the path through command line. However, you must note that this setting would

be temporary and you would need to set them each time you start the command prompt.

- e. In Windows, you can use the command:

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

- f. On a Mac, use:

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

6. You can test whether the tool is correctly installed by typing the following in the command line:

```
vlt --version
```

7. Keep your command prompt window open.

8. Additional information about installing and using VLT is available at
https://docs.adobe.com/docs/en/crx/2-3/how_to/how_to_use_the_vlttool.html

Exercise - Use VLT to perform content synchronization

You have made many changes in CRX using CRXDE. Now, you want to check-out that change to your local repository (Eclipse workspace) and vice-versa. To accomplish and understand the process, follow these steps:

9. Verify that VLT is on your \$PATH. Using the `cd` command, change the directory to
training\ui.content\src\main\content\jcr_root

10. Execute the following command:

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

This performs a VLT check-out of the JCR repository to your local file system.

Warning! This takes several minutes.

You will now have file serializations of the JCR content; for example, in **training\ui.content\src\main\content\jcr_root\.content.xml**.

The JCR paths that are checked out are configured in

training\ui.content\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 check-out. They will then be stored in your home directory inside `~/.vault/auth.xml`.

11. Navigate to `training\ui.apps\src\main\content\jcr_root`, and execute the command:

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

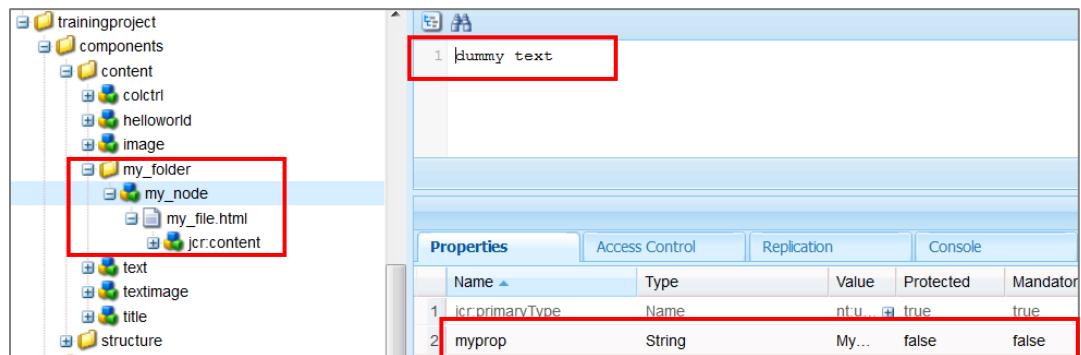


NOTE: Ensure you have navigated to this specific directory (above) in order to execute this important command.

12. In CRXDE Lite, navigate to `/apps/trainingproject/components/content`, and create a node named **my_folder**, of type **sling:Folder**.
13. Inside the folder, add a node named **my_node** of type **nt:unstructured**.
14. With **my_node** selected, in the main area of CRXDE Lite, go to the **Properties** tab (which should be open too, by default).
15. At the bottom of the **Properties** tab, you will see three empty fields at the bottom of the tab. Enter the following properties to the corresponding fields then click **Add**.

Property	Type	Value
myprop	String	Myvalue

16. Verify a new row (#2) is now on the Properties of the node for `my_node`. It includes all the information you added in the previous step.
17. To the same **my_folder** folder, right-click **my_node** and select **Create > Create File** and enter the following filename: **my_file.html** then click **OK**. A new tab `*my_file.html` opens. The “*” indicating you have not made or saved any changes to the file yet.
18. Double-click the file and in the left panel include the following text:
dummy text



19. Click **Save All** in the upper-left corner of CRXDE Lite.
20. On the command line, using the cd command, change the directory to **training\ui.apps\src\main\content\jcr_root**, and execute the following command:

```
vlt up
```

You should see (with paths abbreviated):

```
C: \training\ui.apps\src\main\content\jcr_root>vlt up
Connecting via JCR remoting to http://localhost:4502/crx/server
A apps\trainingproject\components\content\my_folder
A apps\trainingproject\components\content\my_folder\.content.xml (text/xml)
A apps\trainingproject\components\content\my_folder\my_node
A apps\trainingproject\components\content\my_folder\my_node\.content.xml (text/xml)
A apps\trainingproject\components\content\my_folder\my_node\my_file.html (text/html)

C: \training\ui.apps\src\main\content\jcr_root>_
```

21. In your training directory, change directory to **training\ui.apps\src\main\content\jcr_root\apps\trainingproject\components\content\my_folder\my_node**.
22. Open **.content.xml** in a text editor, 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"
        myotherprop="NewValue">
          <my_file.html/>
</jcr:root>

```

23. Commit to CRX by executing the following command from **training\ui.apps\src\main\content\jcr_root**:

```

vlt ci
apps\trainingproject\components\content\my_folder\my_node\.content.xml

```

You should see:

```

Connecting via JCR remoting to http://localhost:4502/crx/server
Collecting commit information...
sending.... apps\trainingproject\components\content\my_folder\my_node\.content.xml
Transmitting file data...
U apps\trainingproject\components\content\my_folder\my_node\.content.xml
done.

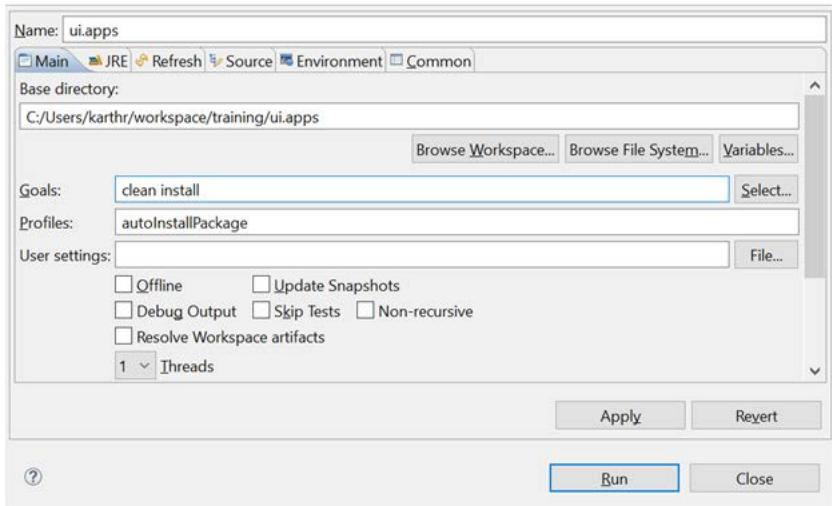
```

24. Verify the change in CRXDE Lite.

Name	Type	Value	Protected	Managed
jcr:primaryType	Name	nt:unstructured	true	true
myotherprop	String	NewValue	false	false
myprop	String	Myvalue	false	false

25. Change the content of **apps/training/components/my_file.html** through a text editor or in eclipse, and commit to CRX not using VLT, but by building the package. To build the package open eclipse and Right-click **training.ui.apps** and click **refresh**.

- Right-click **training.ui.apps** and go to **Run As > Maven build**
- In the Edit Configuration dialog box, enter the Goals as **clean install** and in the Profiles field, enter **autoInstallPackage** and click Apply and Run.



26. Verify the change in CRXDE Lite.

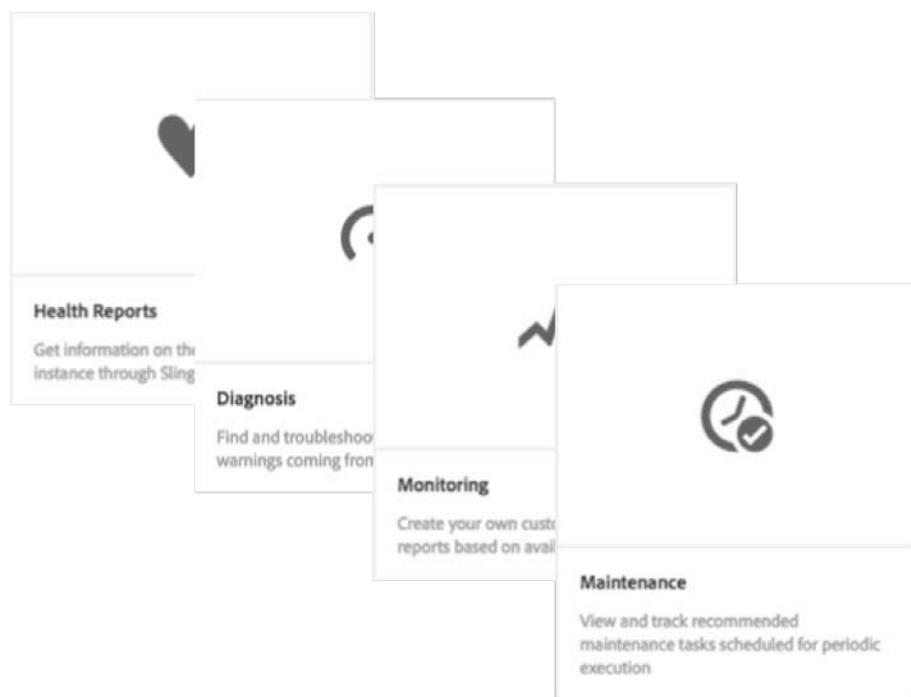
Review:

- Set up VLT and tested the checkout process.

APPENDIX: OPERATIONS DASHBOARD

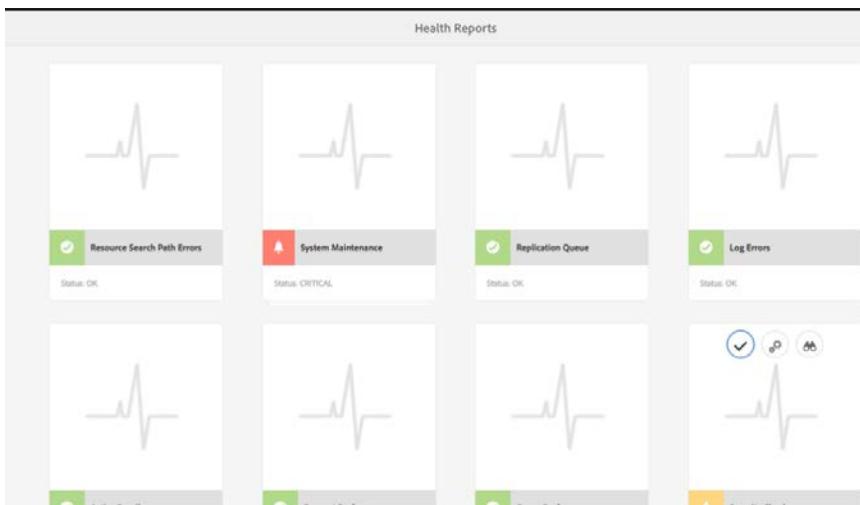
Overview

The Operations Dashboard in Adobe Experience Manager 6.2 helps system operators to monitor the Adobe Experience Manager system health at a glance. The Dashboard also provides auto-generated diagnosis information on relevant aspects of Adobe Experience Manager and allows configuring and running of self-contained maintenance automation to reduce project operations and support cases significantly. The Operations Dashboard can be extended with custom health checks and maintenance tasks. Further, Operations Dashboard data can be accessed from external monitoring tools via JMX. You can access the Dashboard from AEM Home > Tools > Operations or by using OmniSearch by pressing "/" and entering "Operations".



Health Reports

The Health Report system provides information on the health of an AEM instance through Sling Health Checks. This can be done via either OSGI, JMX or HTTP requests (via JSON). It offers measurements and threshold of certain configurable counters and in some cases, will offer information on how to resolve the issue.

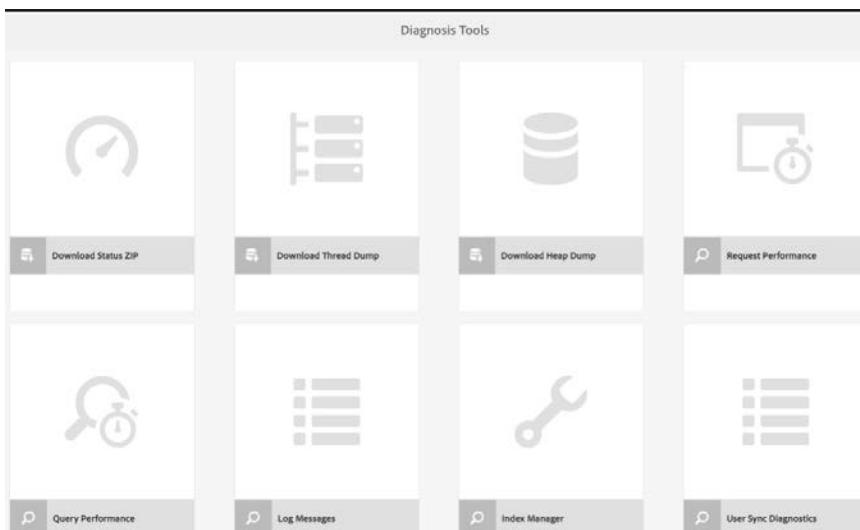


Diagnosis Tools

Diagnosis Tools that can help finding and troubleshooting root causes of the warnings coming from the Health Check Dashboard, as well as providing important debug information for system operators.

Amongst the most important features are:

- A log message analyzer
- The ability to access heap and thread dumps
- Requests and query performance analyzers



Automated Maintenance Tasks

The Automated Maintenance Tasks page is a place where you can view and track recommended maintenance tasks scheduled for periodic execution. The tasks are integrated with the Health Check system and their execution has minimal impact on system performance. The tasks can also be manually executed from the interface.

AEM 6 ships with a default set of automated maintenance tasks, including Version Purge, Workflow Purge, and Revision Clean Up among others,

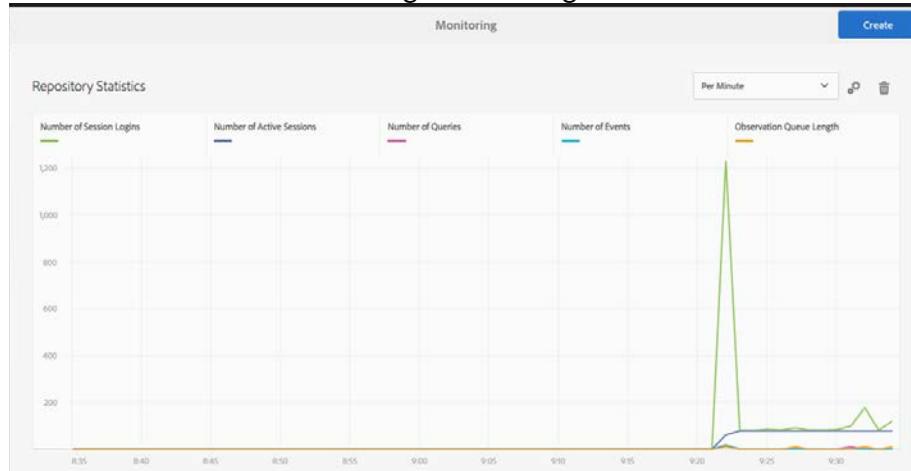
Custom maintenance tasks can be implemented as OSGi services. The maintenance task infrastructure is based on Apache Sling's job handling.

The screenshot shows a 'Maintenance' dashboard with two cards:

- Daily Maintenance Window:** Daily: 2:00 to 5:00, Next: Aug 27 2016 02:00 EDT
- Weekly Maintenance Window:** Weekly: Saturday 1:00 to Saturday 2:00, Next: Aug 27 2016 01:00 EDT

Monitoring

Health Check Dashboard can integrate with Nagios via the Granite JMX Mbeans.

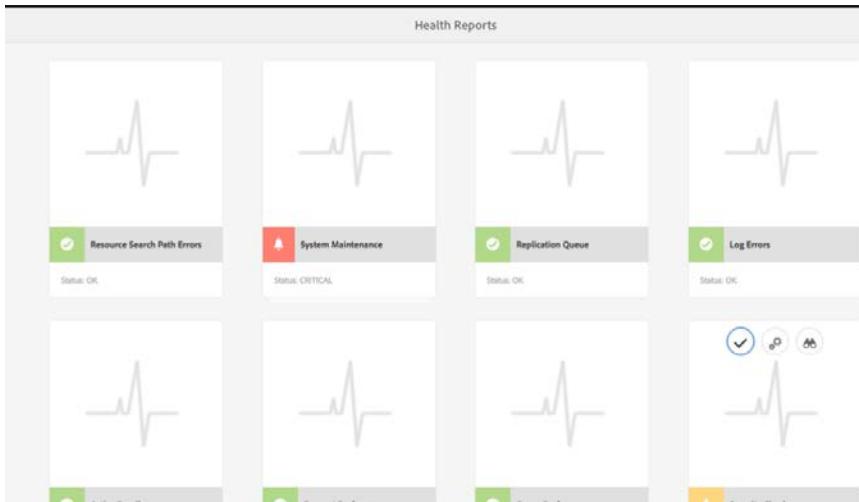


For more information about the Operations Dashboard, see

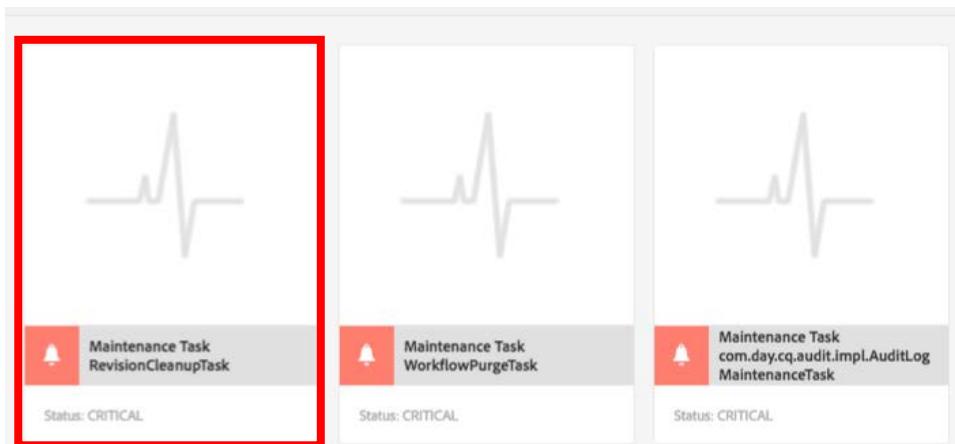
<https://docs.adobe.com/docs/en/aem/6-2/administer/operations/operations-dashboard.html>.

Exercise 1: Diagnosing an issue using the Operations Dashboard

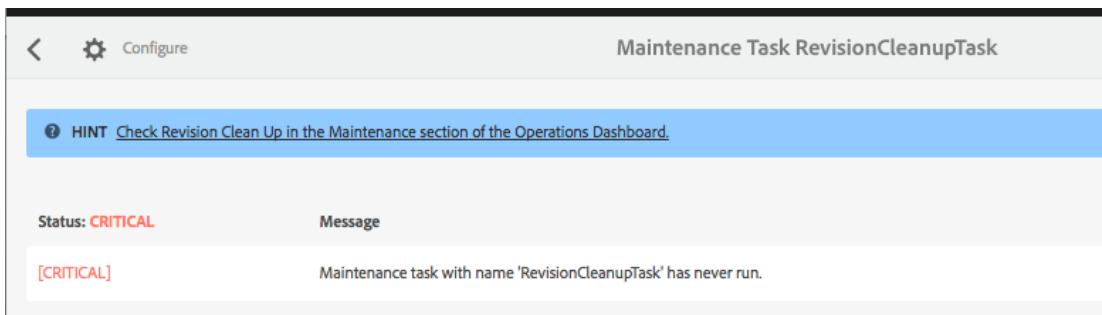
1. Navigate to AEM Home > Tools > Operations. Select Health Reports.



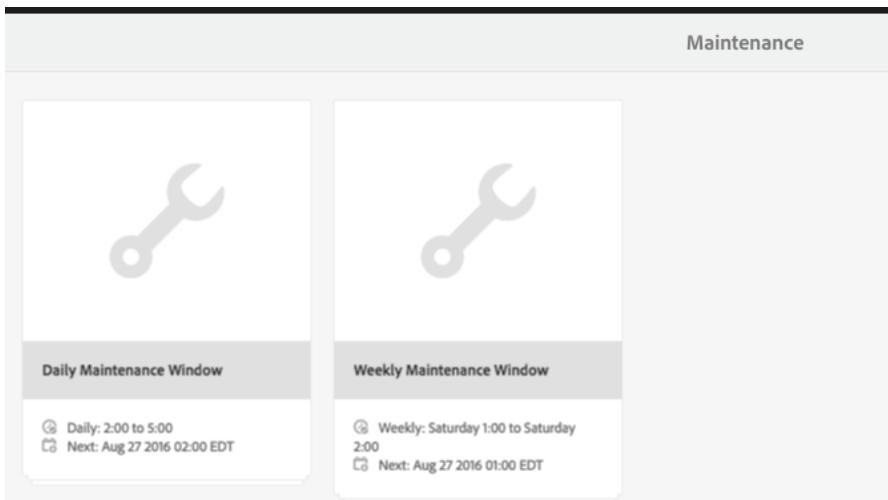
2. Notice that the System Maintenance card is in a CRITICAL state. You can use the Dashboard functionality to diagnose and treat the problem.
3. Double-click on the System Maintenance card to display the next level of information. Notice that all the cards are in a CRITICAL state.



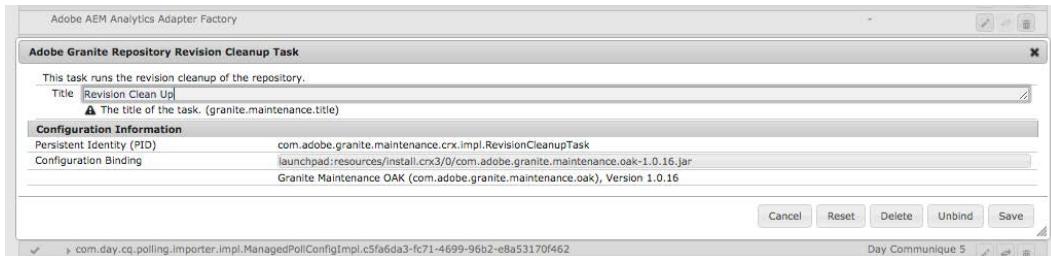
4. Double-click on the RevisionCleanupTask to display details of the problem. Notice that the system gives you a diagnosis and a Hint for solving the problem.



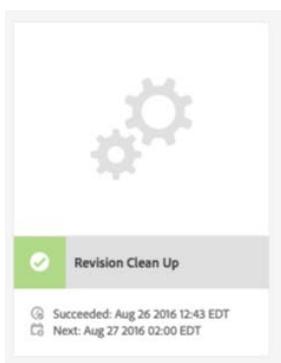
- Click on the Hint, which takes you to the Automated Maintenance Tasks page. Notice there are daily and weekly maintenance tasks.



- Click in the Daily Maintenance Window to see the list of scheduled Daily tasks.
- Hover on the RevisionCleanup Task. Notice that you can Run the task and configure the task.



- To launch the task, click the play button. While the task is running, there will be a yellow indicator and then when it is complete, a green (success) or red (failure) indicator.



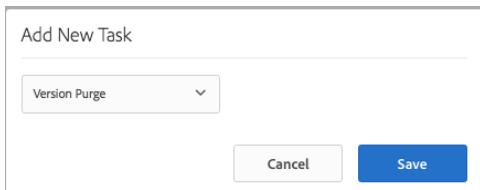
Congratulations! You now know how to use the Operations Dashboard to diagnose and fix an issue.

Exercise 2: Add a New Task to the Maintenance Schedule

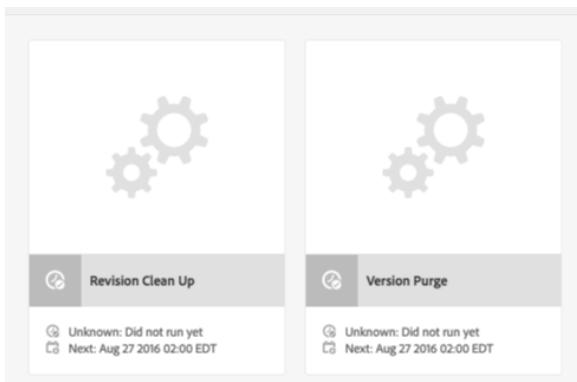
1. Navigate to AEM Home > Tools > Operations > Maintenance > Daily Maintenance.



2. Click on the "+" icon to add a new task.



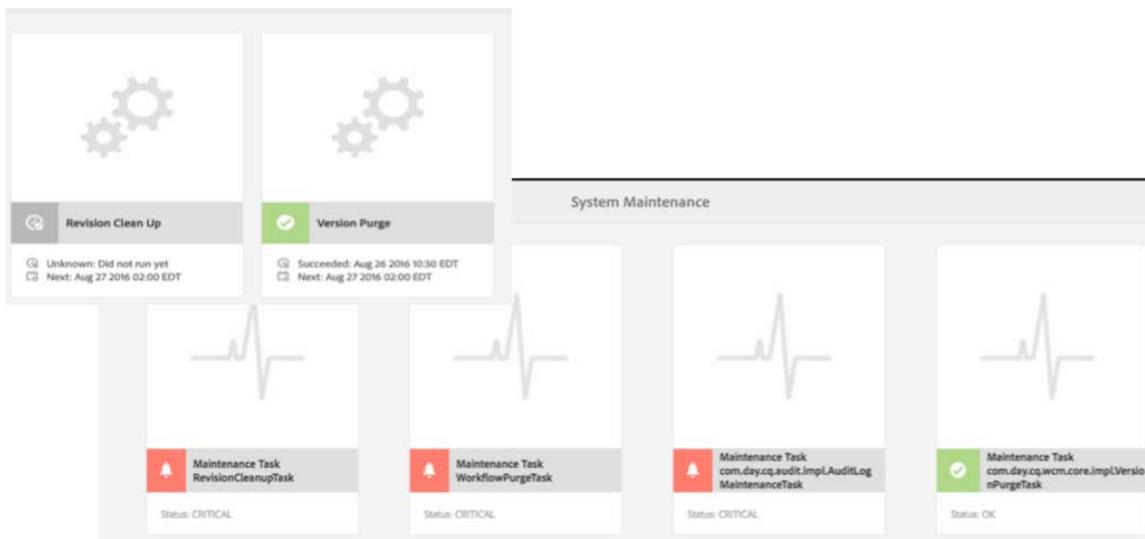
3. Select the **Version Purge** task and click **Done**.
4. A new card will appear in the daily maintenance page.



5. Before you run the Version Purge task, return to the System Maintenance page, AEM Home > Tools > Operations > Health Reports > System Maintenance. Notice that a new CRITICAL task has shown up.



6. Return to the daily maintenance page and run the Version Purge task by clicking on the play button.



Congratulations! You now know how to add a new task to the automated maintenance schedule.

Exercise 3 Modify the Maintenance Schedule

1. Navigate to AEM Home > Tools > Operations > Maintenance.

The screenshot shows the 'Maintenance' section of the AEM interface. It features two main cards: 'Daily Maintenance Window' and 'Weekly Maintenance Window'.
The 'Daily Maintenance Window' card includes:

- A gear icon with a checkmark.
- A wrench icon.
- The title 'Daily Maintenance Window'.
- Details: 'Daily: 2:00 to 5:00' and 'Next: Aug 27 2016 02:00 EDT'.

The 'Weekly Maintenance Window' card includes:

- A wrench icon.
- The title 'Weekly Maintenance Window'.
- Details: 'Weekly: Saturday 1:00 to Saturday 2:00' and 'Next: Aug 27 2016 01:00 EDT'.

2. Hover over the **Daily Maintenance Window** and click the **Configure** button.

The screenshot shows the 'Configure Maintenance Window' dialog box. It contains the following fields:

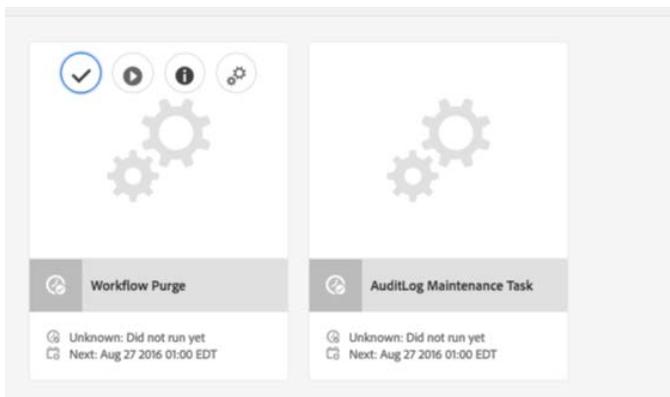
- Name: Daily Maintenance Window
- Recurrence: Daily (radio button selected)
- Start: 03:00
- End: 05:00
- Buttons: Save (blue) and Cancel

3. Modify the **Start** time to 3:00 and Click Save.

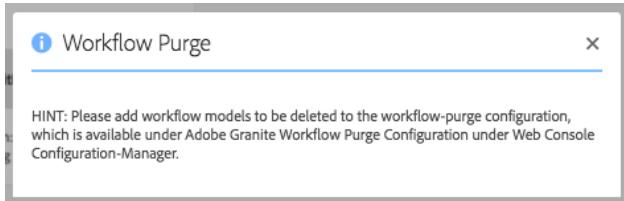
Congratulations! You now know how to modify the Automated Maintenance Schedules.

Exercise 4 Configure the Workflow Purge Task.

1. Navigate to and open the **Weekly Maintenance Window** (**Tools > Operations > Maintenance > Weekly Maintenance window**)

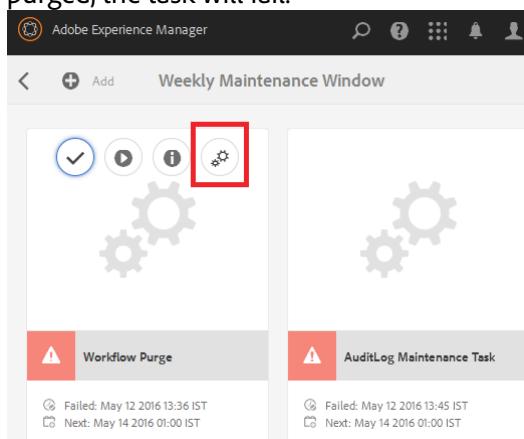


2. Hover over the Workflow Purge task and click **Info**.

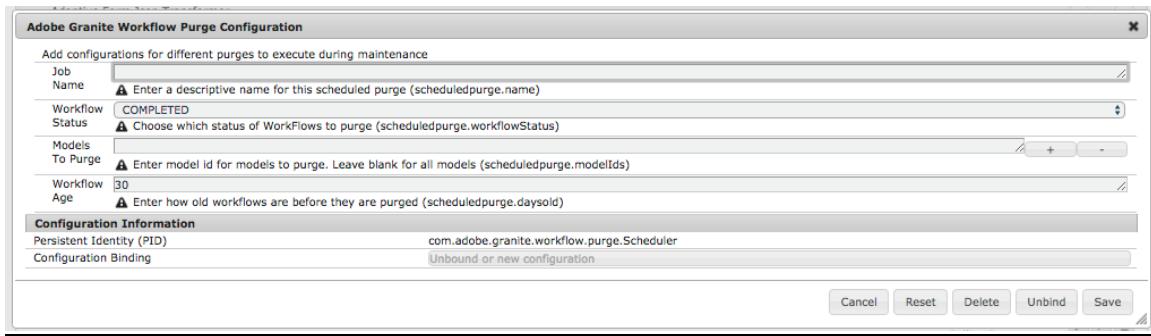


3. Close the Info window.

To configure the Workflow Purge, you must add the workflow models to be purged to the Workflow Purge Scheduler. If you run the task before specifying the workflow models to be purged, the task will fail.



4. Navigate to **AEM Home > Tools > Web Console > Configuration Manager**. Find the Workflow Purge Scheduler: [com.adobe.granite.workflow.purge.Scheduler](http://localhost:4502/system/console/configMgr/com.adobe.granite.workflow.purge.Scheduler).



Notice that the configuration is empty. To configure the Workflow Purge Scheduler, you must create configuration nodes in the repository.

5. Navigate to **AEM Home > Tools > CRXDE Lite** (<http://localhost:4502/crx/de/>)
6. If the `/apps/geometrixx/config.author` folder does not already exist, do the following (otherwise, go to Step 7):
 - a. Right click the `/apps/geometrixx` folder and choose **Create... Create Node**.
 - b. Enter the following values in the **Create Node** dialog:

Name: config.author

Type: sling:Folder

- c. Click **Save All**.

7. Right-click `/apps/geometrixx/config.author` and select **Create > Create Node**.
8. Enter the following values in the **Create Node** dialog and click **OK**:

Name: com.adobe.granite.workflow.purge.Scheduler-WEEKLY

Type: sling:OsgiConfig

9. **Save**.
10. Add the following properties on the Properties tab. As you add each property, click **Save All**. Refer to the following table for the four properties you will add:

Name	Type	VALUE
<code>scheduledpurge.name</code>	String	Weekly Purge
<code>scheduledpurge.workflowStatus</code>	String	COMPLETED
<code>scheduledpurge.modelIds</code>	String[]	""(leave this blank)
<code>scheduledpurge.daysold</code>	Long	28

The screenshot shows the CRXDE Lite interface with the following details:

- Left Sidebar:** Shows a tree view of AEM components under the "apps" folder, including "sem-forms", "commerce", "community-components", "dam", "geometrix", "geometrix-commons", "geometrix-gov", "geometrix-instore", "geometrix-media", "geometrix-outdoors", "geometrix-unlimited-app", "granite", "rep-policy", "settings", and "social".
- Header:** CRXDE | Lite Content Repository Extreme Dev
- Search Bar:** Enter search term to search the repository
- Table:** Properties of the com.adobe.granite.workflow.purge.Scheduler-WET configuration

Name	Type	Value	Protected	Mar
jcr:created	Date	2016-08-26T13:36:11.290-04:00	true	false
jcr:createdBy	String	admin	true	false
jcr:primaryType	Name	sling:OsgiConfig	true	true
scheduledpurge.daysold	Long	28	false	false
scheduledpurge.modelIds	String[]		false	false
scheduledpurge.name	String	Weekly Purge	false	false
scheduledpurge.workflowStatus	String	COMPLETED	false	false

11. Navigate to **AEM Home > Tools > Web Console > Configuration Manager**
[http://localhost:4502/system/console/configMgr/com.adobe.granite.workflow.purge.Scheduler again.](http://localhost:4502/system/console/configMgr/com.adobe.granite.workflow.purge.Scheduler)

The screenshot shows the "Adobe Granite Workflow Maintenance Operations MBean" dialog with the following configuration:

Configuration Information	Value
Persistent Identity (PID)	com.adobe.granite.workflow.purge.Scheduler.d41cfd06-f649-4046-9e96-87c5ed5cccf
Factory Persistent Identifier (Factory PID)	com.adobe.granite.workflow.purge.Scheduler
Configuration Binding	launched:resources/install/19/com.adobe.granite.workflow.core-2.0.12-CQ620-B0002.jar
Adobe Granite Workflow Core (com.adobe.granite.workflow.core), Version 2.0.12.CQ620-B0002	

Buttons at the bottom: Cancel, Reset, Delete, Unbind, Save.

Notice that all values, except the workflow models are filled in. The explanation next to the ModelIds to Purge notes that leaving the ModelIds blank will purge all models.

12. Return to the Workflow Purge task and run the task. You will notice that the card has a green indicator, meaning Success.

The screenshot shows the 'Weekly Maintenance Window' interface in Adobe Experience Manager. It displays two maintenance tasks:

- Workflow Purge**: Status: Succeeded (May 12 2016 19:13 IST), Next: May 14 2016 01:00 IST.
- AuditLog Maintenance Task**: Status: Failed (May 12 2016 13:45 IST), Next: May 14 2016 01:00 IST.

Both the 'Workflow Purge' task icon and its status message are highlighted with a red box.

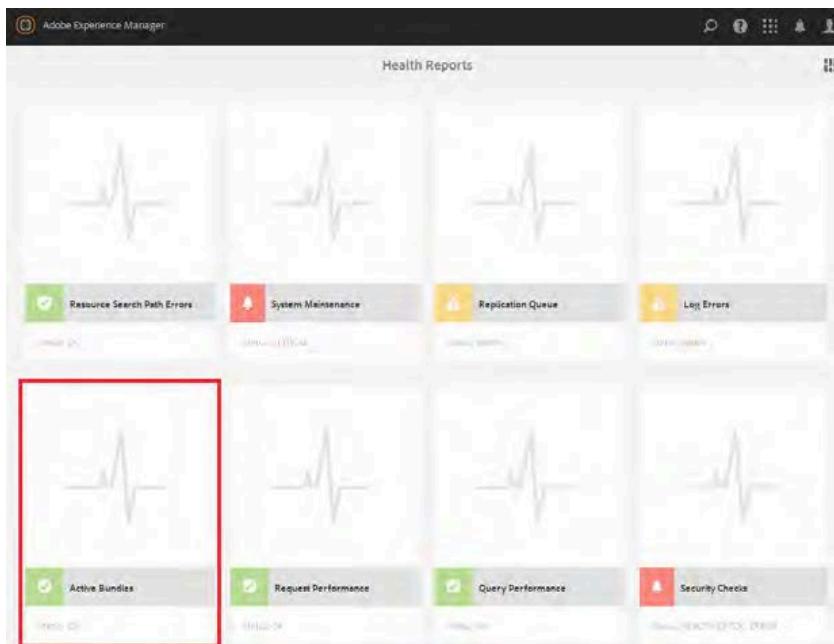
Congratulations, you now know how to configure a purge task.

Extra credit exercise: Successfully run the Audit log Maintenance Task.

- Using the same methodology as Exercise 8.4 and the com.adobe.cq.audit.purge.Scheduler. Configure the Audit Log Maintenance task to run successfully.

Exercise Extra Credit: Configure Health Reports to check Bundle status.

- Navigate to AEM Home > Tools > Operations > Health Reports.
- Check the **Active Bundles** card in the **Health Reports**.



- After the vanilla installation of AEM with the out-of-the-box settings, the status of this card should be **OK** (Green card). Click the card to ensure that there are no inactive or unresolved bundles.

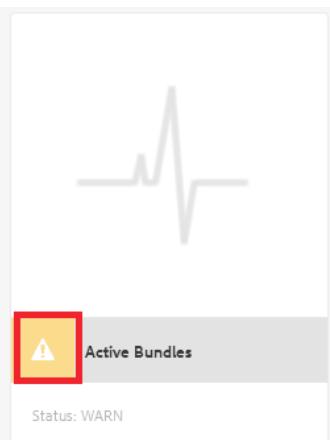


To **test the Health Check**, you will stop a bundle.

- Navigate to **AEM Home > Tools > Web Console**.
- Click the **Stop** button for the **Apache Sling Simple WebDAV Access to Repositories** bundle.

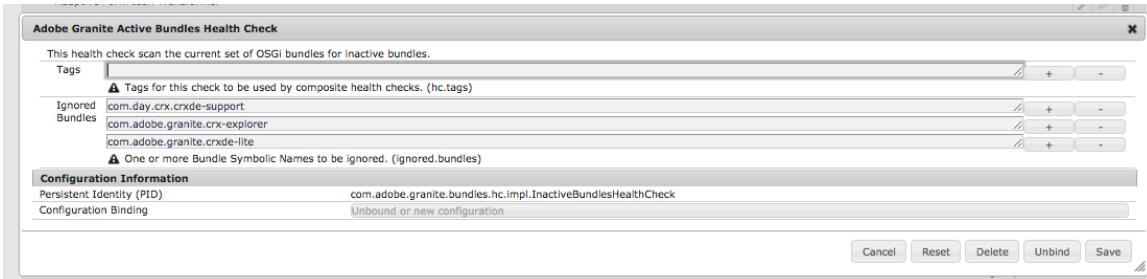
319	Apache Sling Simple WebDAV Access to repositories (org.apache.sling.jcr.webdav)	2.3.4	sling,jcr	Active	
319	Apache Sling Simple WebDAV Access to repositories (org.apache.sling.jcr.webdav)	2.3.4	sling,jcr	Resolved	

6. Return to the Health Reports and refresh the page. You will notice that the Active Bundles card has changed to yellow (WARN).

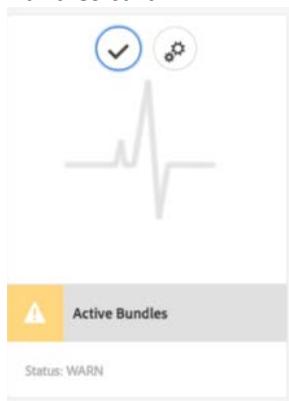


You can also configure the Active Bundles Health Check to monitor the status a specific set of bundles.

7. Navigate to **AEM Home > Tools > Web Console > Configuration Manager**. Find the **Adobe Granite Active Bundles Health Check**.



Pro Tip: There is a short cut to go directly to the **Adobe Granite Active Bundles Health Check** bundle from the Active Bundles Health Check. Click the Configuration button on the Active Bundles card.



8. For testing purposes, you can quickly fill in the form in the Web Console configuration tab, but it is preferable to use a node of type **sling:OsgiConfig** with the appropriate name and properties as described below.

Node:

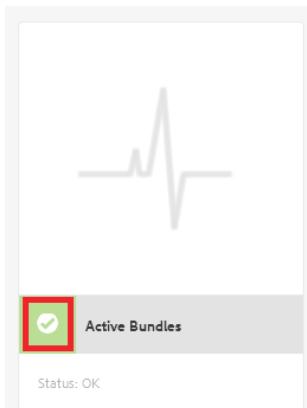
Node Type	sling:OsgiConfig
Name	com.adobe.granite.bundles.hc.impl.InactiveBundlesHealthCheck

Properties:

hc.tags	String[] = "" (leave this blank)
ignored.bundles	String[] = org.apache.sling.jcr.webdav

Name	Type	Value	Protected	Mandatory	Multiple	Auto Create
hc.tags	String		false	false	false	false
ignored.bundles	String	org.apache.sling.jcr.webdav	false	false	false	false

9. Return to the **Health Reports** and you will notice that the **Active Bundles** card has returned to the green **OK** status:



The OK status is reached because you specified that the org.apache.sling.jcr.webdav bundle be ignored.

Congratulations! You successfully used the Operations dashboard to check the status of your AEM instance concerning the 'Active Bundles', and you have learned how to configure the health check bundle associated with this card.

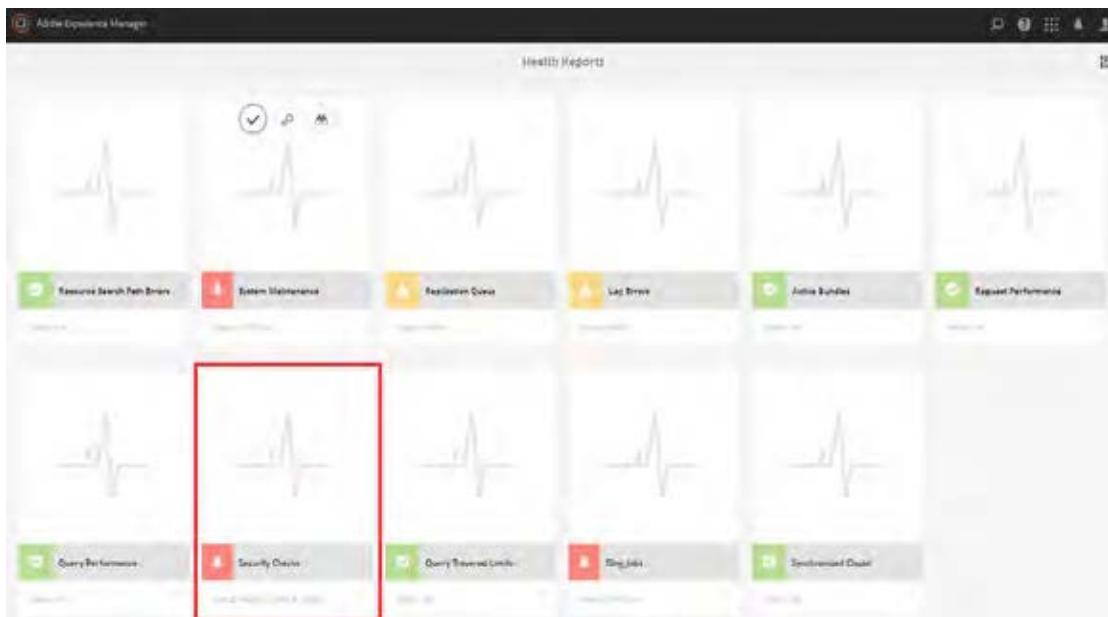
Composite Health Checks

A Composite Health Check's role is to aggregate a number of individual Health Checks sharing a set of common features. For instance, the Security Composite Health Check groups together all the individual health checks performing security checks.

You can create custom composite Health Checks by configuring the Composite Health Check OSGi component.

Exercise 6 Working with Security Checks

1. Navigate to **Tools > Operations > Health Reports > Security Checks**. You will see some cards for a number of key security points. Note the disclaimer as it is important to realize that there is more to security than what you find in the dashboard. For more help on this point, you should consult the online documentation as concerns the Security checklist:
<https://docs.adobe.com/docs/en/aem/6-2/administer/security/security-check-list.html>



Most of the cards in **Security Checklist** dashboard are in yellow **WARN** status.

Security Checks				
> Health Reports Disclaimer				
✓ Deserialization Firewall Attach API Readiness Status: OK	✓ Deserialization Firewall Functional Status: OK	✓ Deserialization Firewall Loaded Status: OK	✓ Authorizable Node Name Generation Status: OK	⚠ CRXDE Support Status: WARN
✓ DavEx Health Check Status: OK	⚠ Default Login Accounts Status: WARN	⚠ Sling Get Servlet Status: WARN	⚠ CQ Dispatcher Configuration Status: WARN	⚠ Example Content Packages Status: WARN

2. Notice that the Default Logins card is one of the Security Checks with **WARN** status.



- Click on the Default Logins card to see a diagnosis and Hint for solutions.

This screenshot shows the detailed view of the 'Default Login Accounts' card from the previous interface. It contains four blue 'HINT' boxes with the following text:

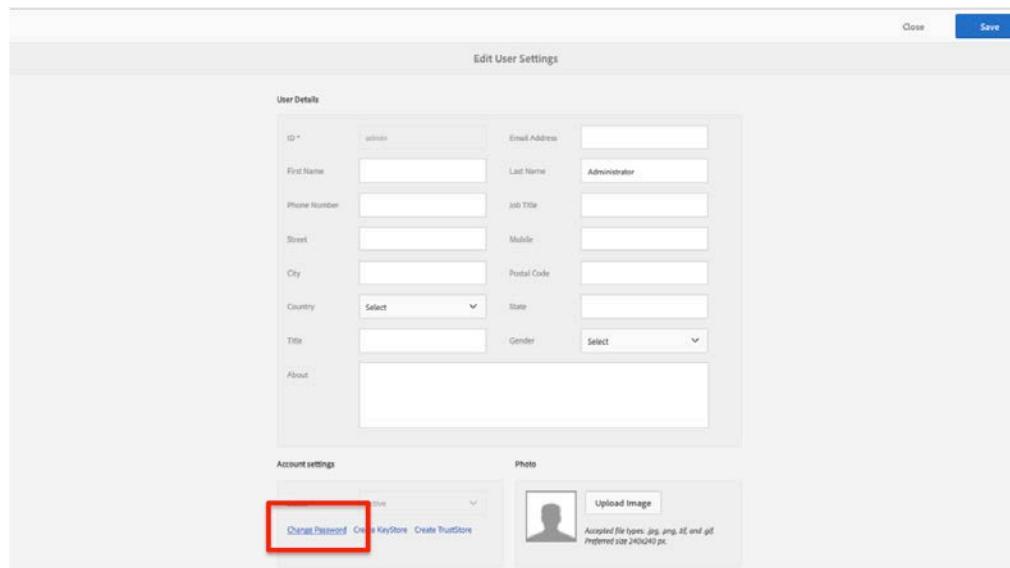
- HINT You can change the admin account password via the User Admin.
- HINT Check the 'Changing the CQ Admin Password' section in the security guidelines.
- HINT You can change the OSGI admin password via the configuration of the Apache Felix OSGI Management Console.
- HINT Check the 'Changing the OSGI Web Console Admin Password' section in the security guidelines.

Below these hints, there's a table with two columns: 'Status' and 'Message'. The 'Status' column has four rows labeled 'WARN', 'WARN', 'INFO', and 'INFO'. The 'Message' column contains the following text corresponding to each status:

- WARN: Login as [admin: admin] succeeded, was expected to fail.
- WARN: Login as [author: author] succeeded, was expected to fail.
- INFO: It is strongly recommended to change the default admin accounts for CQ.
- INFO: The default OSGI console credentials were not changed. It is strongly recommended to change them.

Follow the hints to find a solution for this Security Check.

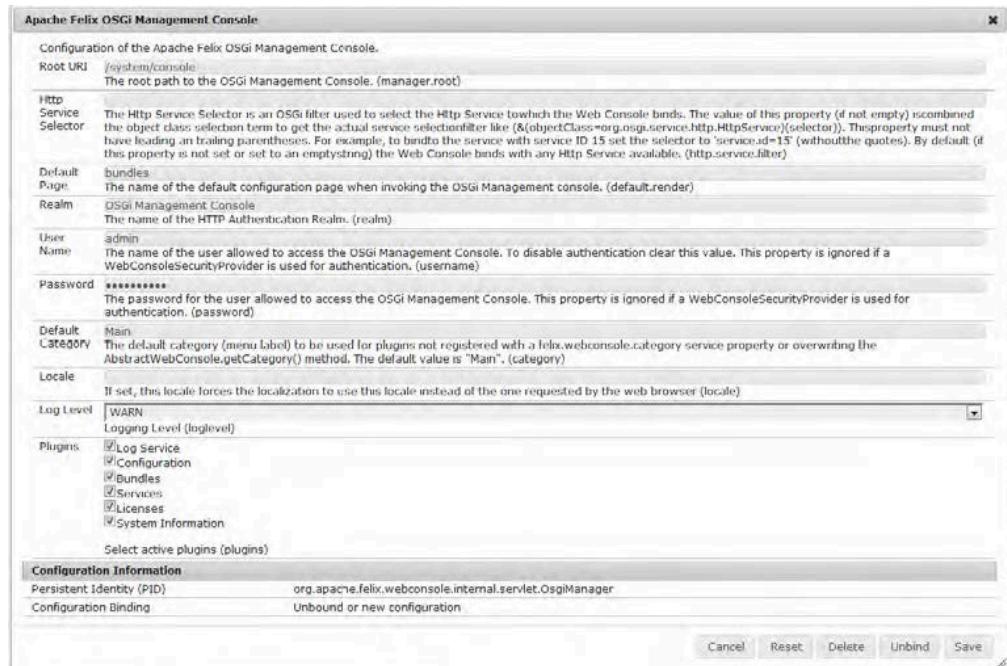
- Change the CRX Password.
 - Navigate to **AEM Home > Tools > Security**.
 - Select the **Administrator** user.



- c. Click the **Change Password** link.
- d. Modify the password.

NOTE: do not forget the password!! This is the new administrator password!!

5. Change the Web Console password
 - a. Navigate to **AEM Home > Tools > Web Console > Configuration Manager**.
 - b. Find the **Apache Felix OSGi Management Console** component and modify the password directly in the console.



Note: this is only item that should be configured directly in the Web Console.

6. Change the password for the default user Author.
 - a. Navigate to **AEM Home > Tools > Security**.
 - b. Select the **Author** user.
 - c. Click the **Change Password** link.
 - d. Modify the password.

The screenshot shows a 'Change Password' dialog box. It has three input fields: 'New password *' (highlighted with a yellow background), 'Retype Password *' (grey background), and 'Your Password *' (white background). At the bottom are 'Cancel' and 'Save' buttons.

7. Return to the Security Checks page in the Dashboard. The Default Logins card is now green (OK status).

The screenshot shows the Adobe Experience Manager interface. At the top, there's a navigation bar with icons for search, help, and user profile. Below it, the page title is "Default Login Accounts". On the left, there's a sidebar with a gear icon labeled "Configure". The main content area displays a table with three rows. The first row has a status column "Status: OK" and a message column "Login as [admin: admin] failed, as expected.". The second row has a status column "Status: OK" and a message column "Login as [author: author] failed, as expected.". The third row has a status column "Status: OK" and a message column "The the OSGI console admin password was changed, as expected.". A red box highlights the first row.

Status	Message
[DEBUG]	Login as [admin: admin] failed, as expected.
[DEBUG]	Login as [author: author] failed, as expected.
[DEBUG]	The the OSGI console admin password was changed, as expected.

Congratulations! You consulted the Security Checks dashboard, addressed some of the points in the Security checklist to change the status of the Default Login's security card, and learned about the Security checklists.

