



Extend and Customize Adobe  
Experience Manager

# student guide



ADOBE COPYRIGHT PROTECTED

©2018 Adobe Systems Incorporated. All rights reserved.

#### Adobe Experience Manager Technical Basics

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.

05-17-2018

# Table of Contents

Introduction.....	8
Objectives.....	8
Fluid Experiences.....	9
AEM Architecture.....	10
Basics of AEM Architecture Stack.....	10
Application-Level Functions.....	14
AEM Installation.....	15
Instances.....	15
Installation Prerequisites.....	16
Graphical and Command-Line Methods to Install AEM.....	17
Exercise 1: Install AEM author and publish instances.....	20
Task 1.1: Install and start an AEM author instance using the graphical method.....	20
Task 1.2: Install and start an AEM publish instance using the graphical method.....	23
Task 1.3: Start an AEM author instance using the command line method.....	26
(Optional) Task 1.4: Start AEM using nosamplecontent and change the admin password.....	27
Authoring Basics.....	30
AEM UI: Key Features.....	30
AEM Sites Pages.....	33
Creating Pages.....	33
Editing Pages.....	33
Exercise 2: Create and edit a page in AEM .....	35
Task 2.1: Create a page.....	35
Task 2.2: Edit a page.....	39
Developer Tools.....	46
CRXDE Lite.....	46
Web Console.....	47
Package Management in AEM .....	48
Exercise 3: Install, create, build, and download a package.....	51
Task 3.1: Install a package.....	51
Task 3.2: Create, build, and download a package.....	57
References.....	61
Introduction.....	62
Objectives.....	62
Working with Maven.....	63
Installing and Configuring Eclipse.....	67

Setting up the AEM Plug-in for Eclipse.....	67
Configuring the AEM Perspective.....	67
Exercise 1: Install and configure Eclipse (Optional).....	69
Task 1: Install Eclipse.....	69
Task 2: Configure Eclipse.....	71
Exercise 2: Install and configure the Eclipse AEM plug-in (Optional).....	73
Exercise 3: Import an existing Maven project.....	76
Task 1: Import an AEM project.....	76
Task 2: Update the POM files for the project.....	79
Configuring the AEM Console.....	82
Building and Deploying a Project Using Maven.....	84
Exercise 4: Build and deploy the project to AEM.....	85
Task 1: Build the AEM project.....	85
Task 2: Deploy the project.....	88
Exercise 5: Configure the AEM Server in Eclipse.....	90
Task 1: Configure the AEM Server .....	90
Task 2: Export content from Eclipse.....	94
References.....	96
Introduction.....	97
Objectives.....	97
OSGi Configurations .....	98
Methods to Create OSGi Configurations.....	98
Adding a New Configuration to the Repository.....	100
Run Modes.....	102
Introduction to Run Modes.....	102
Custom Run Modes.....	102
Setting Run Modes .....	103
Exercise 1: Create custom OSGi configurations and start AEM with a custom run mode.....	105
Task 1.1: Create a custom OSGi configuration node.....	105
Task 1.2: Start AEM with a custom run mode.....	109
Task 1.3: Create a custom OSGi configuration node for a custom run mode .....	111
Answers to Questions.....	113
References.....	114
Introduction.....	115
Objectives.....	115
AEM Logging System.....	116
Types of Log Files in AEM.....	117
Loggers and Writers .....	118
Creating Your Own Loggers and Writers .....	120

Exercise 1: Observe project logger config node .....	121
Exercise 2: Create custom loggers.....	125
Introduction.....	128
Objectives.....	128
OSGi – An Overview.....	129
OSGi Architecture.....	129
Bundles.....	130
Declarative Services .....	136
Annotations in OSGi.....	139
Exercise 1: Implement a bundle activator .....	144
Task 1: Create a new Java class .....	144
Task 2: Deploy the project.....	146
Exercise 2: Create and use a custom service .....	147
Task 1: Create a service and an implementation.....	147
Task 2: Deploy the bundle and expose the service in HTL.....	151
Task 3: Test the service.....	153
Exercise 3: Code OSGi configurations.....	155
Task 1: Update the OSGi component .....	155
Task 2: Deploy the bundle.....	161
Task 3: Test the service.....	161
Introduction.....	164
Objectives.....	164
Apache Sling .....	165
RESTful Architecture.....	166
Understanding Sling Resolution Process.....	168
Basic Steps of Processing Requests.....	169
The Resource Resolver.....	171
Working with Sling Servlets.....	175
Configuring the Default Sling GET Servlet .....	175
Configuring the Sling POST Servlet.....	176
Exercise 1: Create a Sling servlet.....	178
Task 1: Create a servlet with a path .....	178
Task 2: Create a servlet with a resource type .....	181
Sling POST Servlet.....	182
Exercise 2: Add content with Sling post servlet.....	183
Creating System Users.....	189
Exercise 3: Create a system user.....	192
Task 1: Create a service user .....	192
Task 2: Create mapping from the bundle to the user.....	196
Understanding ResourceResolver .....	199

Working with Sling Models.....	201
Injector Specific Annotations .....	205
Exercise 1: Create a Sling Model .....	207
Task 1: Create data for the Sling Model.....	207
Task 2: Adapt the resource to a Sling Model.....	209
Task 3: Adapt the request to a Sling Model .....	214
Sling Model Exporter.....	219
Exercise 2: Extend a core component.....	220
Query Index.....	227
Configuring the Indexes.....	230
Exercise 3: Search resources with queries.....	237
Sling Events .....	242
Exercise 1: Handle OSGi Events.....	245
Task 1: Create a new Java class .....	245
Task 2: Deploy and test the class.....	248
Working with Sling Scheduler.....	250
Sling Jobs.....	251
Scheduling Jobs.....	251
Exercise 2: Create a job consumer for a topic.....	253
Exercise 3: Clean up nodes with a Sling scheduler.....	257
Extra Credit: Add more Cleanup paths .....	263
Event Modeling.....	264
Exercise 4: Create a resource listener.....	267
References.....	273
Polling Importers.....	275
Exercise 1: Create a polling importer.....	276
Task 1: Create a polling importer.....	276
Task 2: Deploy and test the polling importer.....	281
Creating AEM Pages and Assets Programmatically .....	285
Exercise 2: Create AEM pages programmatically.....	286
Task 1: Create a page creator .....	286
Task 2: Deploy and test the page creator .....	290
Exercise 3: Programmatically create an AEM page.....	293
Task 1: Create a CSV page creator.....	293
Task 2: Deploy and test the CSV page creator.....	298
AEM Projects .....	301
Exercise 4: Create a simple project .....	302

Exercise 5: Create a project template .....	305
Executing an Existing Workflow.....	312
Workflow Launchers.....	314
Exercise 6: Execute a workflow .....	316
Exercise 7: Build and test a workflow.....	319
Task 1: Build a workflow .....	319
Task 2: Test the workflow .....	328
Exercise 8: Customize the process step.....	335
Task 1: Create the custom workflow process.....	335
Task 2: Deploy the class.....	337
Task 3: Customize the workflow.....	338
Task 4: Test the workflow.....	341
References and Helpful Links:.....	346
Working with Users, Groups, and Access Control Lists.....	348
Users and Groups .....	348
Permissions and ACLs.....	349
Exercise 1: Create a new user.....	352
Exercise 2: Create a new group.....	355
Exercise 3: Add the user to the group.....	358
Exercise 4: Create a custom AEM project with ACL automation.....	360
Task 1: Create a custom workflow process.....	360
Task 2: Add the workflow process to a project workflow.....	365
Task 3: Increase permissions on the workflow process user.....	368
Task 4: Test the permissions automation with an AEM project.....	369
References and Helpful Links:.....	376
Understanding Testing Frameworks.....	378
The Mockito Framework.....	378
Performing Unit Tests .....	379
Performing Functional Tests .....	380
Jenkins for Continuous Integration .....	382
Exercise 1: Create unit tests using Mockito .....	383
Exercise 2: Create unit tests using Sling Mocks.....	386
Task 1: Observe the sling-mock dependency .....	386
Task 2: Create a unit test with sample data.....	387
Task 3: Run the test.....	390
Exercise 3: Create unit tests using AEM Mocks.....	391
Task 1: Observe the AEM-mock dependency .....	391
Task 2: Create a unit test with sample data.....	392
Task 3: Run the test.....	397

References and Helpful Links:.....	398
------------------------------------	-----

ADOBE COPYRIGHT PROTECTED

# Adobe Experience Manager Technical Basics



## Introduction

Adobe Experience Manager (AEM) is a content management system that helps you build and manage online content. It is a Java-based web application that is built using different technologies and has a user-friendly and flexible User Interface (UI). This helps customers build robust and scalable applications.

In order to use AEM capabilities effectively, you should first understand the architecture, UI, basic authoring features, and the administrative consoles of AEM.

## Objectives

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

- Explain fluid experiences in AEM
- Explain the architecture of AEM
- Explain different types of AEM instances
- Install and run AEM instances by using the graphical method
- Install and run AEM instances by using the command line method
- Navigate through the AEM UI
- Create and edit pages
- Explain the features and UI elements of the AEM developer tools
- Install a package in AEM
- Create, build, and download a package

## Fluid Experiences

AEM enables organizations to create seamless digital experiences for their customers. It helps design, anticipate, and deliver rapidly adaptable experiences across web, mobile, in-store, and any end point in the customer journey.

Customers use offline and online touch points to engage with an organization's products and services. It is important that organizations provide a fluid experience across multiple touch points. This increases customer engagement through various channels.

AEM helps you deliver these fluid experiences by supporting true omnichannel experiences across owned, earned, and paid channels. Content goes beyond the traditional "web" form and is available in a multitude of end points – screens, dynamic adaptive forms, documents, email, social, apps, and more.

The Content Services feature in AEM enables you to reuse web content in mobile app, single page applications and custom applications.

You can integrate AEM with Adobe Target to create personalized and targeted content. You can integrate AEM with Adobe Analytics to understand how visitors to a page have interacted with that page. This provides information about what you need to test and optimize to enhance customer experience.

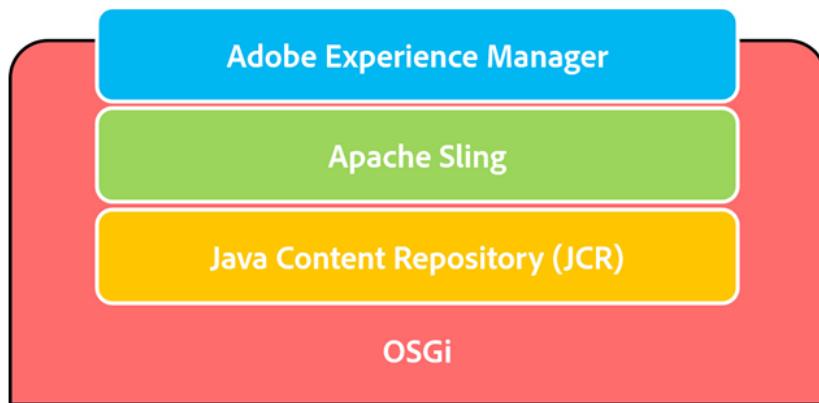
## AEM Architecture

AEM is a web-based client-server system, made up of several infrastructure-level and application-level functions. These functions are used to build relevant applications.

### Basics of AEM Architecture Stack

AEM architecture stack is based on technologies such as OSGi, Java Content Repository (JCR), and Apache Sling.

The following diagram depicts a high-level view of the AEM architecture stack:



## Granite Platform

Granite is a general-purpose platform for building robust scalable applications. It is Adobe's open web stack, and forms the technical foundation on which AEM is built. Granite supports an *open architecture*, which is based on both *open standards* (JCR and OSGi) and *open source projects* (Apache Sling and Apache Jackrabbit).



**Note:** Granite is an open development project within Adobe but not an open source project.

---

Technically, at the core, Granite provides:

- **An application launcher** for a standalone Java or Web application archive for deployment in the existing servlet containers or application servers.
- An **OSGi Framework** into which all applications are deployed.

- **OSGi Compendium Services** to support building applications, such as Log Service, Http Service, Event Admin Service, Configuration Admin Service, Declarative Services, and Metatype Service.
- A comprehensive **Logging Framework** providing various logging APIs, such as SLF4J, Log4F, Apache Commons Logging, and OSGi Log Service.
- A repository based on **Apache Jackrabbit Oak** and **JSR-283**.
- The **Apache Sling Web Framework**.

## Granite UI

The consoles in the main navigation, tools, and editors of AEM are built using the Granite UI. The Granite UI provides a foundational UI framework for:

- UI widgets
- Extensible and plug-in-based admin UI

It also adheres to the following requirements:

- Mobile first (designing an online experience for mobile before designing it for the desktop)
- Extensible
- Easy to override



**Note:** The Granite UI is based on Coral 3, which is Adobe's universal UI across all products.

---

## OSGi Framework

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

---

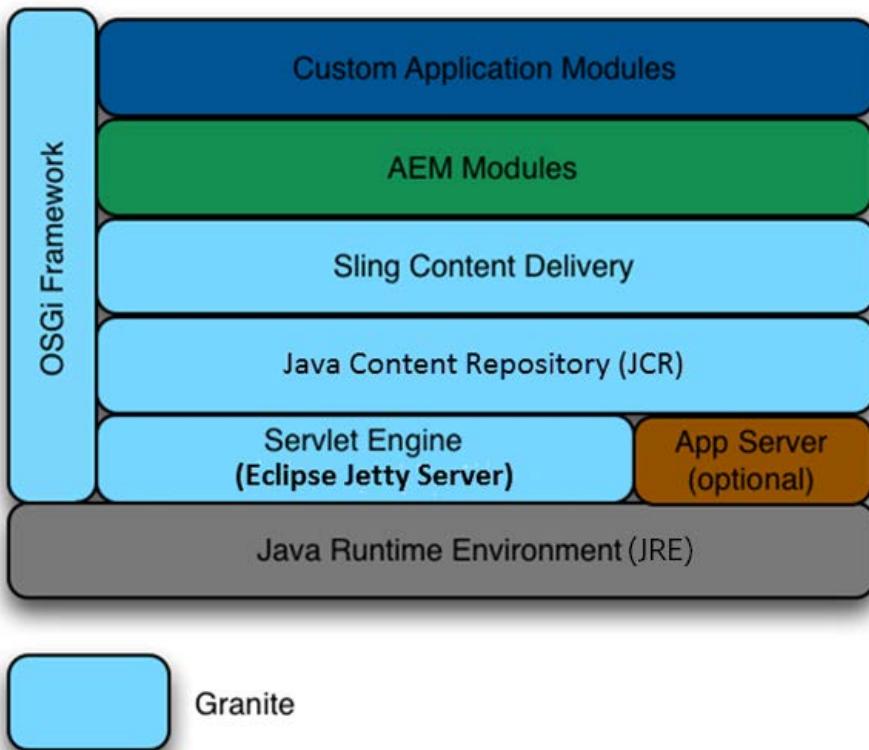


**Note:** OSGi used to stand for Open Service Gateway Initiative, but that name has been discontinued, and it is now officially no longer an abbreviation. It is just known in the industry as *OSGi*.

---

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 by 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.

The following diagram depicts the AEM platform foundation of Granite and the OSGi framework:



## Apache Felix

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

## JCR

The JCR, specifically Java Specification Request-283 (JSR- 283), is a database that supports structured and unstructured content, versioning, and observation. In other words, it is a database that resembles a file system.



**Note:** The Adobe implementation of JSR-283 was known as the Content Repository eXtreme (CRX). Hence, you may see *CRX* in some tools and interfaces in AEM. However, the CRX as a feature is being phased out. In its place, AEM uses the Granite platform and Apache Jackrabbit Oak.

All data pertaining to AEM, such as HTML, HTML Template Language (HTL), CSS, JavaScript/Java, images, and videos are stored in the JCR object database. JCR is built with Apache Jackrabbit Oak, an open-source project.

AEM also works with other JCR repositories, such as Apache Jackrabbit 2.x, and with a number of non-JCR data stores through connectors. Therefore, it is capable of pulling content from its built-in Oak repository and any JCR-compliant source, such as a third-party repository (for example, Jackrabbit 2.x) or a connector that exposes the legacy storage through the JCR.

The advantages of using JCR are:

- It provides a generic application data store for structured and unstructured content. While file systems provide excellent storage for unstructured and hierarchical content, the databases provide storage for structured data. This way, JCR provides the best of both the data storage architectures.
- It 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. This means that this title property is defined in the jcr namespace.
- It provides one interface to interact with text and binary data. This helps in easy access and management of data in comparison to storing it in multiple places.

## Apache Sling

Apache Sling is a web application framework for content-centric applications and uses a JCR, such as Apache Jackrabbit Oak, to store and manage content.

The key features of Apache Sling include:

- It is Apache open source.
- It is based on REST principles and helps build applications as a series of OSGi bundles.
- It is resource-oriented (every resource has a URI) and maps to JCR nodes.

A request URL is first resolved to a resource, and then based on the resource, Apache Sling selects the Servlet or script to handle that request. Servlets and scripts are handled as resources and are accessible by a resource path. This means every script, Servlet, filter, and error handler is available from the Resource Resolver just like normal content—providing data to be rendered on request.

## Application-Level Functions

At the application-level, AEM has the following functions:

- AEM Sites (Web Experience Management, Digital Signage (Screens), e-Commerce, and Online Communities)
- AEM Assets (Digital Asset Management, Assets Sharing and Dynamic Media Delivery)
- AEM Forms (Online Forms Management & Dynamic Customer Communications)
- Managed Services (Adobe-hosted operations for AEM deployments)
- Content Services
- Multi-Site Manager
- Workflows

The above application functions/modules sit on top of the Granite platform. They share the same infrastructure and UI framework. These modules are encapsulated within the OSGi container and are very tightly integrated with each other.

## AEM Installation

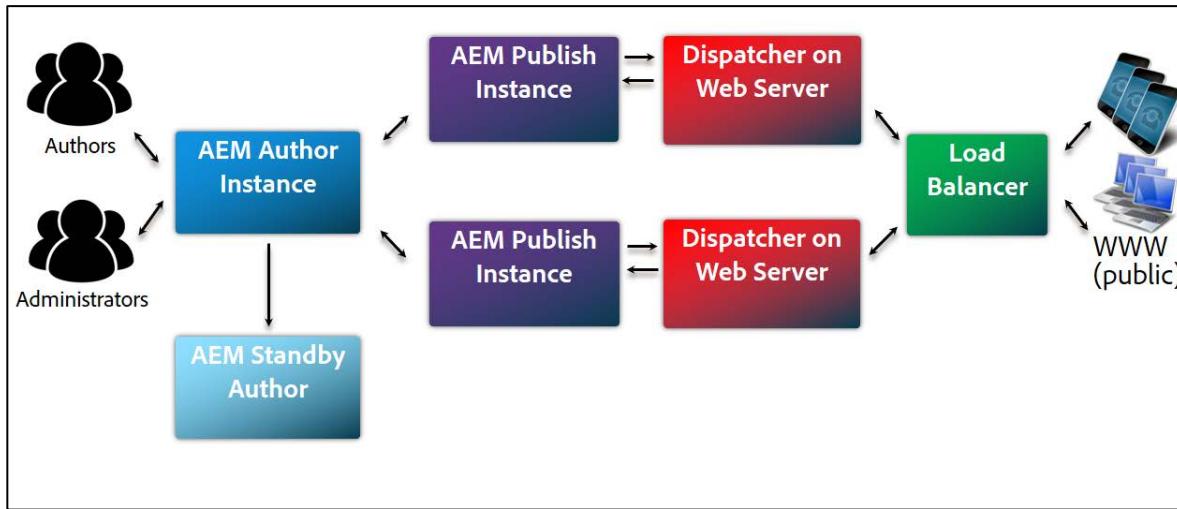
AEM runs on most operating systems that support the Java platform. All client interactions with AEM are done through a web browser.

## Instances

In AEM terminology, an *instance* is a copy of AEM running on a server. AEM installations usually involve running at least two instances:

- **Author:** An AEM instance used to create, upload, and edit content and administer the website. After the content is ready to go live, it is replicated to the publish instance.
- **Publish:** An AEM instance that serves the published content to the public.

The following graphic depicts a typical AEM implementation:



**Note:** The Dispatcher is a static web server, such as Apache httpd and Microsoft IIS, augmented with the AEM Dispatcher module. It caches webpages produced by the publish instance to improve performance.

## Installation Prerequisites

To install AEM, you need:

- AEM installation and startup JAR file (also known as the *quickstart* file)
- A valid AEM license key properties file
- JDK version 1.8
- Approximately 4 GB of free space per instance
- Minimum 4 GB of RAM

During installation, you will notice that the JAR file creates a root folder on your system called crx-quickstart.



**Note:** After installation is complete, the *quickstart* file is referred to as the AEM startup file.



**Note:** Windows users may need to ensure their system's environment variables are set appropriately to run Java 1.8 before installing AEM. Open a command prompt and type `java -version` to ensure Java is set up properly. When you run the command, it should show the following result

`(java version "1.8.0_144"):`

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\rhoades>java -version
java version "1.8.0_144"
Java(TM) SE Runtime Environment (build 1.8.0_144-b01)
Java HotSpot(TM) 64-Bit Server VM (build 25.144-b01, mixed mode)
```

## Graphical and Command-Line Methods to Install AEM

There are many ways of installing and starting an AEM instance, two of which are—graphical and command line.

### Graphical Method

This method involves using the \*.jar file to start an AEM 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 to start a publish instance.

The installation will take approximately 5-7 minutes the first time, depending on your system's capabilities.

A dialog box similar to the following (also known as the GUI) will pop up:



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

### Command Line Method

When you use the command line method to install and start, you can provide additional performance-tuning parameters to the Java Virtual Machine (JVM) and perform other administrative tasks. On Windows, MacOS X, or \*x, you can increase the Java heap size during the installation, which improves performance.

- **Using the Command Line to Install and Start an AEM Author Instance**

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

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

The AEM quickstart installer will show all the available command-line options without starting the server.

In addition, you need to tune the JVM used for running AEM. Tuning the JVM is an important and delicate task and requires a more realistic environment in terms of resources (hardware and the operating system) and workload (content and requests). You can start your instance (author or publish) by 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 to which 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 AEM, 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 AEM consumes a lot of resources).

Example:

```
java -Xms512m -Xmx1024m -jar aem-author-4502.jar
```

- **Using the Command Line to Install and Start an AEM Publish Instance**

If you want to install and start your AEM publish instance 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
```

- **Using the Command Line to Start AEM with the *nosamplecontent* Run Mode**

A run mode is a collection of AEM configuration parameters that allow you to tune your AEM instance for a specific purpose. *nosamplecontent* is a predefined run mode available in AEM.



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

Use *nosamplecontent* in your command line the first time you install an AEM instance to install it without any sample content (which includes the reference site content):

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

Note the use of the `-r author` parameter. This tells AEM to set this instance as the author run mode. The `-gui` option turns on the GUI mode that shows the AEM icon window on your system.

Therefore, the instructions you provided here specify two run modes: author and *nosamplecontent*. The syntax to specify multiple run modes is:

```
-r runmode1, runmode2, ...
```

You may use this option to install on a production environment, which does not require this reference site content. Also, note the *nosamplecontent* option is only available on the first installation of the instance.

Run modes are covered in an additional training course in more depth (OSGi Configurations & Run Modes).

## Additional Method to Start AEM: Batch Files

After you install AEM using a graphical or command line method, you can use the built-in batch files in the `crx-quickstart` directory to start AEM. You can also configure the start up and additional options using these batch files.

Navigate to your install directory and then to the `crx-quickstart` directory that was created as part of your installation. Within `\crx-quickstart\bin` subdirectory, there are batch files (on Windows systems) available for you to start or stop AEM, and get status information on the server. The server parameters provided in these batch files are documented using comments or commented-out as appropriate in the case of options that you may want to set for your instances.

## Exercise 1: Install AEM author and publish instances

**Scenario:** As a developer or administrator, you need to install and start/stop a development instance of AEM on a local machine, using both the quickstart JAR file and command line method.

### Task 1.1: Install and start an AEM author instance using the graphical method

In this task, you will install and start your AEM author instance on port 4502.



**Note:** If you are attending a v/ILT class using ReadyTech, steps 1 through 3 were completed for you.

Skip ahead to step 4.

To install an AEM author instance:

1. Create a folder structure on your file system where you will store, install, and start your AEM author instance. For example, in:
  - a. Windows: **C:/adobe/AEM/author**
  - b. MacOS X: **/Applications/adobe/AEM/author** or \*x: **/opt/adobe/AEM/author**
2. Copy the **aem-quickstart-6.4.0.jar** and **license.properties** files, from the location provided by your instructor, to your newly created directory.
3. Rename the **aem-quickstart-6.4.0.jar** file to **aem-author-4502.jar**:
  - aem = Application
  - author = Web Content Management (WCM) mode AEM will run in (in this case, author)
  - 4502 = Port AEM will run in

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

You can, therefore, control the way AEM is installed by defining properties in a filename.

- Double-click the **aem-author-4502.jar** file (located at C:\adobe\AEM\author in Windows, if you are using ReadyTech). Installation will take approximately 5–7 minutes depending on your system's capabilities.



**Note:** When running for the first time, the quickstart \*.jar will notice that it has to install AEM.

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, AEM 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](#)

After the AEM Author instance has started successfully, the start-up screen (the GUI) will change to something similar to the following:



You have now completed the installation of AEM.



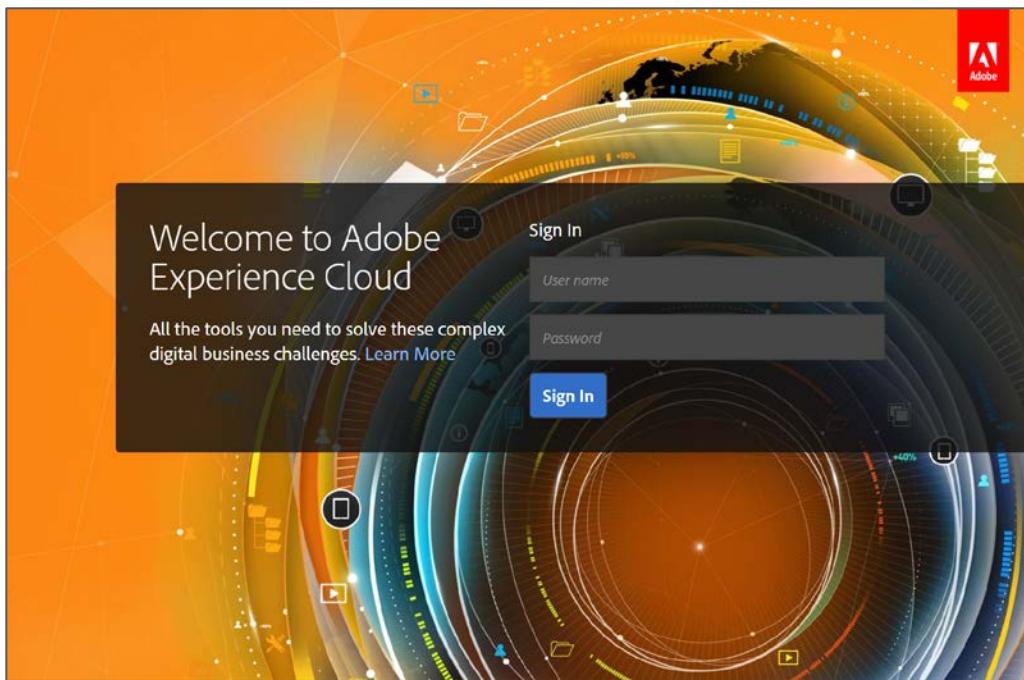
**Note:** To stop the AEM instance, click the ON/OFF toggle button in the GUI.

Now, each time you want to run your AEM author instance, follow the same procedure in step 4 to start it. However, this time, the startup will be as fast as one minute or less, as the initial installation task has been performed.

In addition, after AEM starts, your default browser will automatically open to AEM's start URL (where the port number is the one you defined on installation). For example:

<http://localhost:4502>

A sign in screen is displayed as shown below:



5. Enter the user name and password, and click **Sign In**. If you are using a ReadyTech machine or local installation, use the following credentials to sign in:

User name: **admin**

Password: **admin**



**Note:** Notice, a crx-quickstart directory is also created on your computer as 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

## Task 1.2: Install and start an AEM publish instance using the graphical method

In this task, you will install and start an AEM publish instance on port 4503. The publish instance is a separate run mode where your published content resides for access on the web.



**Note:** If you are attending a v/ILT class using ReadyTech, steps 1 through 3 were completed for you.

Skip ahead to step 4.

To install an AEM publish instance:

1. Create a folder structure on your file system where you will store, install, and start your AEM publish instance. For example, in:
  - a. Windows: Create **C:/adobe/AEM/publish**
  - b. MacOS X: Create **/Applications/adobe/AEM/publish or \*x: /opt/adobe/AEM/publish**
2. Copy the **aem-quickstart-6.4.0.jar** and **license.properties** files, from the location provided by your instructor, to your newly created directory.
3. Rename the **aem-quickstart-6.4.0.jar** file to **aem-publish-4503.jar**:
  - aem = Application
  - publish = Web Content Management (WCM) mode AEM will run in (in this case, publish)
  - 4503 = Port AEM 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

 **Note:** If you have multiple author and 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, ...

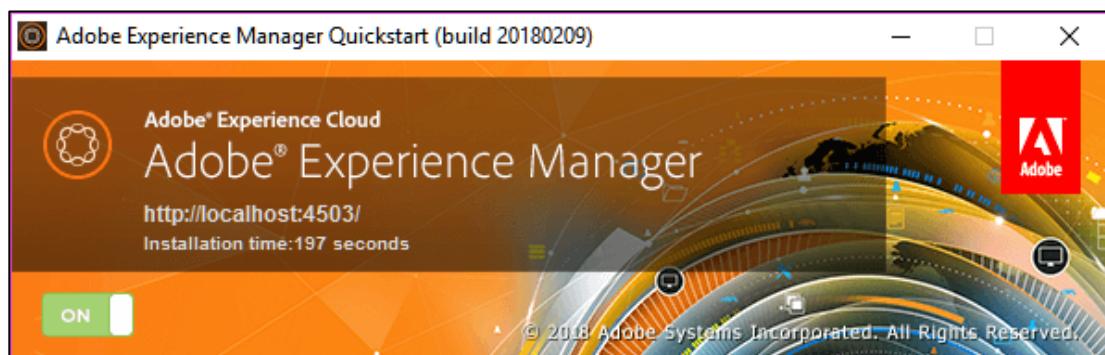
And, your publish instances would be:

- 4503, 4505, 4507, ...

- 
4. Double-click the **aem-publish-4503.jar** file (located at C:\adobe\AEM\publish in Windows, if you are using ReadyTech). Installation will take approximately 5–7 minutes depending on your system's capabilities.

After the initial installation, each time you start an AEM instance (author or publish), it will take 1–2 minutes.

After the AEM publish instance has started successfully on port 4503, the start-up screen (the GUI) will change to something like the following:



The AEM We.Retail page opens in a new tab in your default browser (where the port number is the one you defined on installation). For example, <http://localhost:4503>



**Note:** We.Retail is a reference implementation that illustrates the recommended way of setting up an online presence with AEM. 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.

**Tip:** You do not need to manually sign in as the publish instance loads the We.Retail reference site immediately.

You have now successfully installed and started both AEM author and AEM publish instances on localhost. To start the AEM instance in future, double-click the renamed \*.jar file (in Windows).

## Task 1.3: Start an AEM author instance using the command line method

In this task, you will start and stop an AEM author instance using the command line method.



**Note:** You already have an author instance and a publish instance running. To stop your author instance, click the ON/OFF toggle button in the GUI window.



To start an instance:

1. Open a command prompt in the directory where your quickstart \*.jar file is located to run the author instance. If you are using ReadyTech, this is **C:\adobe\AEM\author**.



**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 (Open command window here) to open that directory in a command-line window.

2. Prior to the installation, you may want to know which parameters are available to configure the AEM quickstart. Enter the following command to display a complete list of optional parameters you can use to install and start AEM (this command will not install and start AEM):

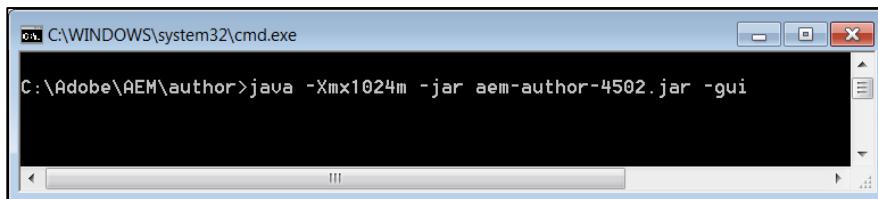
```
java -jar aem-author-4502.jar -h
```



**Tip:** The command may not work if your quickstart file is named differently. If you are using a ReadyTech instance for this training, the filename should be the same as shown in the command above. If you are not using ReadyTech, your quickstart filename may differ, so anytime you run the commands, ensure you use the exact filename.

3. Start your author instance again by allocating 1024 MB to the JVM by using this command:

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



Your AEM instance should start up again, and this time with a command window available to view details of the startup. In addition, the GUI window will also be available for you to shut down AEM.

4. Instead of using the GUI to stop your AEM author instance, use the command window. For Windows, type **CTRL+C** in the command window to stop your AEM author instance.

### (Optional) Task 1.4: Start AEM using nosamplecontent and change the admin password

In this task, you will install AEM without the We.Retail site using the nosamplecontent run mode and change the admin password.

To begin installation:

1. Create a separate sibling directory called **nosamplecontent** on your machine in the same location as your author and publish directories.

In Windows (and also if you are using ReadyTech), this would be:

C:\adobe\AEM\nosamplecontent

2. Copy the **quickstart \*.jar** file and the **license.properties** file from your author directory and paste it into this new directory.

3. Rename your quickstart file to **aem-author-4504.jar**.

4. Open a command prompt in the directory where your **quickstart \*.jar** file is copied.

5. In the command prompt, run the following command:

```
java -jar aem-author-4504.jar -r author, nosamplecontent
```

This installs another author instance with the following instructions:

- With the author and nosamplecontent run modes (that is, an author environment without the We.Retail sample site)
- Running on port 4504 (as specified by the filename you changed)

As this is the first time you are installing an instance of AEM using the command line, a prompt should appear asking you to enter the admin password:

```
java -jar aem-author-4504.jar -r author, nosamplecontent
C:\adobe\AEM\nosamplecontent>java -jar aem-author-4504.jar -r author, nosamplecontent
Loading quickstart properties: default
Loading quickstart properties: instance
Low-memory action set to fork
Using 64bit VM settings, min.heap=1024MB, min permgen=256MB, default fork arguments=[-Xmx1024m,
The JVM reports a heap size of 3559 MB, meets our expectation of 1024 MB +/- 20
Setting properties from filename 'C:/adobe/AEM/nosamplecontent/aem-author-4504.jar'
Option '-quickstart.server.port' set to '4504' from filename aem-author-4504.jar
Setting 'sling.run.modes' to 'author,,nosamplecontent' from command line.
The admin password for this instance hasn't been defined yet.
Please enter the desired admin password:
Please re-enter password:
```

6. Enter a password of your choice for the **admin** user.
7. Re-enter the password to confirm it. Then, wait for your new author instance to install and start.

---

**Tip:** As you did not use the `-gui` option, no *startup* window will appear on your computer to show the progress as AEM installs and loads. Therefore, you will need to wait for your new browser tab `localhost:4504` to appear and signal that AEM has started.

---

8. After the initial installation and startup of AEM, enter your new admin password (instead of `admin/admin`) to confirm that you can log in and that the We.Retail site does *not* appear:




---

**Note:** If you did *not* initially install AEM using the command line, but used the `*.jar` file, the admin password is set for you. The default sign in using this method is:

- User: admin
  - Password: admin
-

- To confirm that We.Retail site does not appear in the AEM navigation, click **Sites** on the default Navigation page. Notice that We.Retail does not appear when you click it (it should only show options such as Screens, Campaigns and Community Sites):

The image contains two screenshots of the Adobe Experience Manager (AEM) interface. The top screenshot shows the 'Navigation' page with a dark header and a sidebar on the left. In the center, there are two icons: 'Projects' (a bar chart) and 'Sites' (a document with a square icon). A red box highlights the 'Sites' icon. The bottom screenshot shows the 'Sites' section of the navigation menu. It has a dark header with the AEM logo and a sidebar on the left containing 'Screens', 'Campaigns', and 'Community Sites'. The 'Screens' item is expanded, showing its sub-items.

- In your command window, type **CTRL+C** to shut down your additional AEM author instance on port 4504 (applicable for Windows only). Be patient as this may take up to 1 or 2 minutes.



**Note:** You need to use this method to stop AEM because you are not using the -gui parameter. Therefore, the GUI window is not available for you to stop AEM.

- Start your author instance on port 4502 again, either using the JAR file method described in Task 1.1 or using the command line method described in Task 1.3.

## Authoring Basics

As you learned in the previous section, the author instance allows you to create, update and review content before publishing it. These authoring functions are made available to you through the AEM UI.

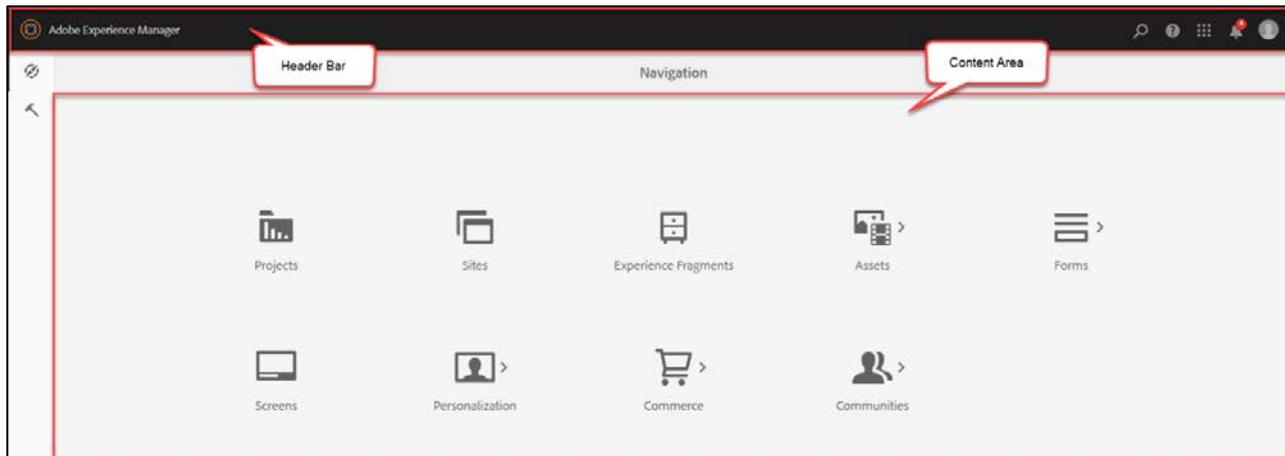
## AEM UI: Key Features

The AEM UI combines the advantages of a web interface with the fluidity and responsiveness that is usually associated with desktop applications. It is touch-optimized for authoring across desktop and mobile devices.

The following table compares the touch and mouse actions, which you can use in AEM:

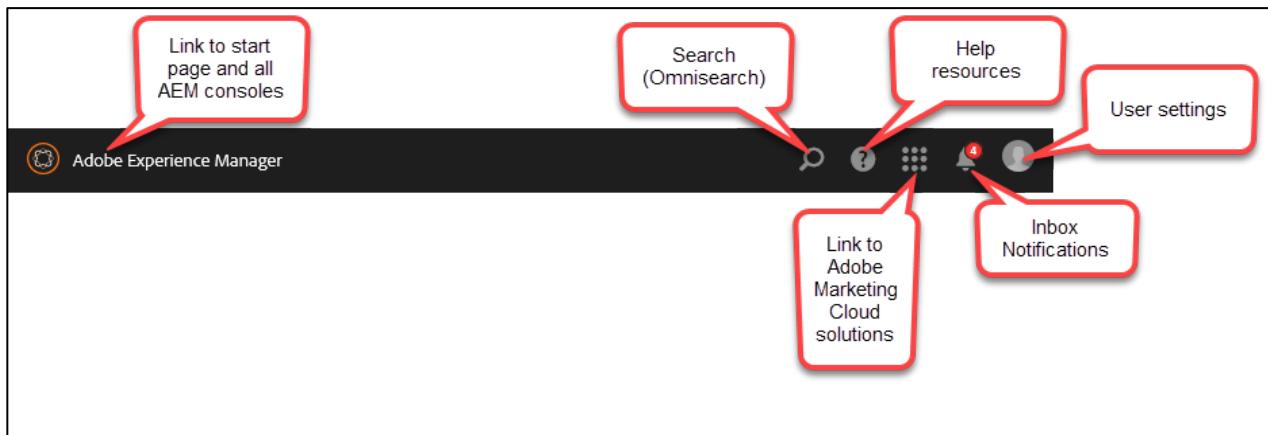
Device UI Actions	Desktop UI Actions
Tap	Click
Touch-and-hold	Double-click
Swipe	Hover

When you start your author instance and sign in to AEM, you are presented with a start screen that includes the header bar and the content area.



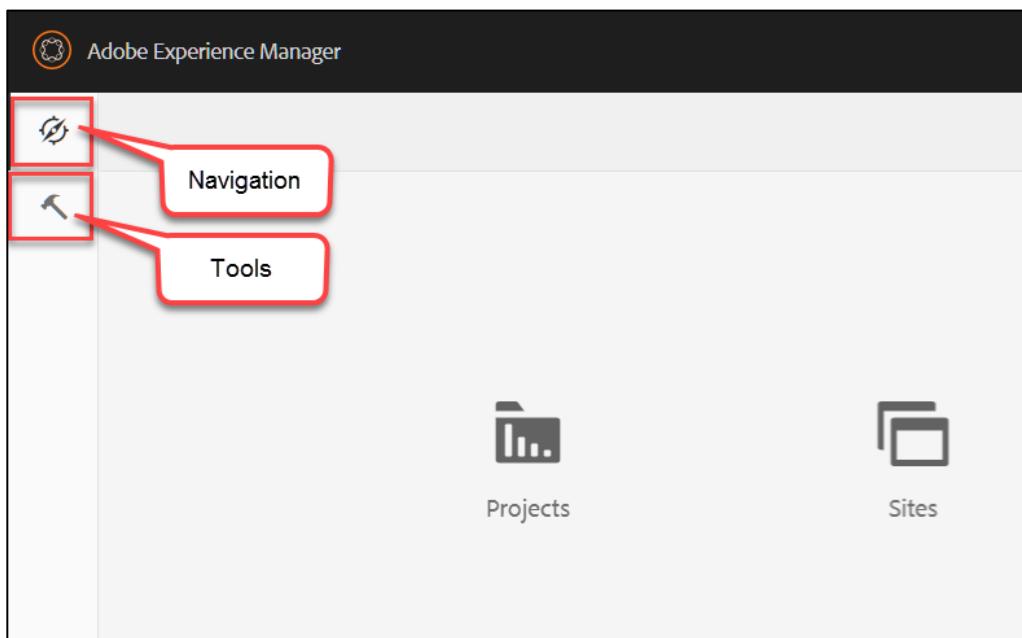
The content area, by default, displays all applications and capabilities of AEM, such as Projects, Sites, Experience Fragments and Assets.

The header bar displays the default options as shown below and changes depending on the process or item you have selected:



## Navigation and Tools

There are two main sections in the AEM UI: Navigation and Tools.



- **Navigation**

The Navigation section is represented by a compass icon in the AEM UI. By default, when you first sign in, the Navigation section loads in the content area. All applications of AEM, such as Projects, Sites, Experience Fragments, Assets, and Forms, are available in this section. This is also the main section relevant to AEM authors to manage and build content for AEM Sites or leverage assets.



**Note:** The applications within the Navigation section are arranged like folders in a hierarchy. The applications indicated with a carat symbol, as shown in the following diagram, have child sub-folders available, such as **Assets > Files**:



- **Tools**

The Tools section is represented by a hammer icon in the AEM UI. This section displays all AEM administrative consoles, developer tools, and other technical consoles available. AEM developers and administrators use this section to develop and administer websites, digital assets, and other aspects of the content repository.

# AEM Sites Pages

A page in AEM is similar to a webpage and contains **components** such as text, images and videos. These pages are created from **templates** that define the structure of the page—including where each component has to be placed.

## Creating Pages

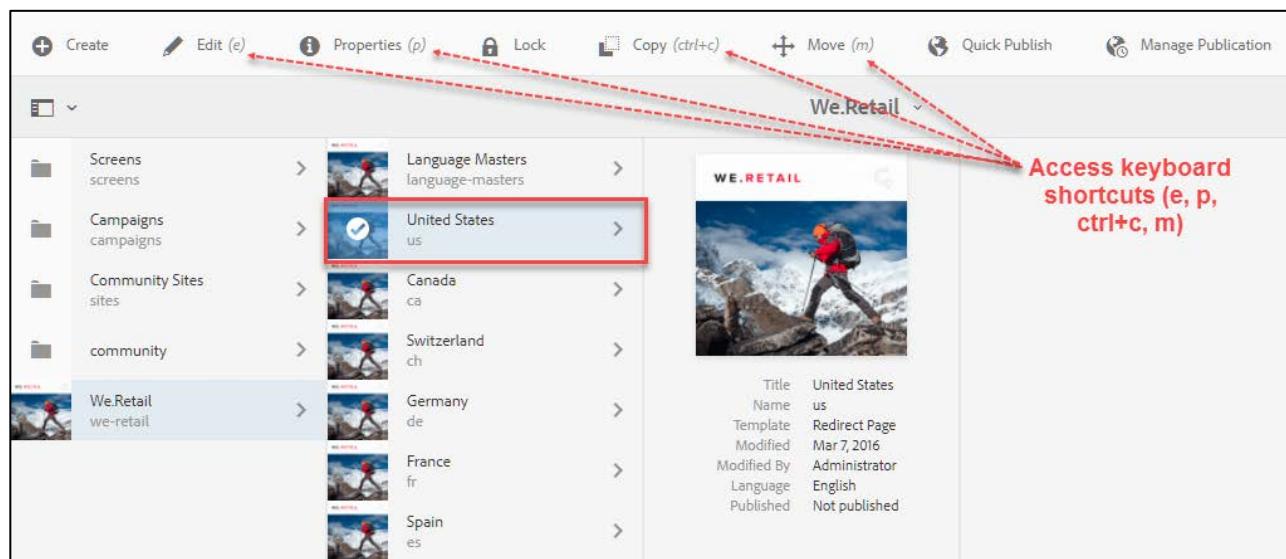
To create a page in AEM Sites, you need to follow a simple wizard where the page template is specified, and the page is given a title and a name. The title is displayed to the user and the name is used to generate the page URL (and can be derived from the title).

## Editing Pages

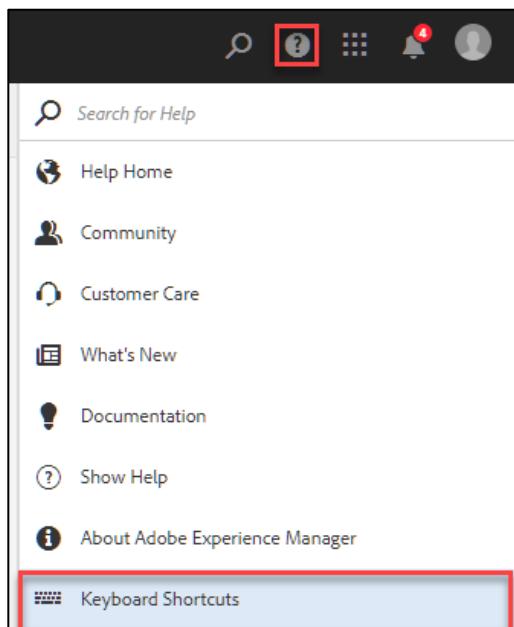
After you create a page, you can edit and add content to it. You can add content by dragging the components available in the side panel onto your page in the page editor.

### AEM Keyboard shortcuts

When you select an object in AEM such as a page, the header bar changes to show a menu of options to work with the object. To aid in editing and managing pages, there are a collection of keyboard shortcuts you can use to quickly access common tasks and functions, as shown below. By default, each user's keyboard shortcuts are enabled.



You can customize or enable/disable shortcuts by navigating to **Help > Keyboard Shortcuts**:



---

 **Note:** For accessibility reasons, shortcuts may be disabled if they conflict with screen readers.

---

## Reference Site: We.Retail

We.Retail is an example retail outdoor equipment reference site that comes with AEM. The purpose of We.Retail is to show the recommended way of setting up an online presence with AEM Sites. This site is built with the following best practices of AEM:

- Localized site structure, with language masters live-copied into country-specific sites
- Content fragments
- Core components
- Responsive layout for all pages
- All editable templates
- HTML Template Language (HTL) for all components
- eCommerce capabilities with product catalog
- AEM Communities sites for site visitors

---

 **Note:** While We.Retail illustrates a retail vertical, only the product catalog and cart features are retail specific. The site is set up in a way that it can be applied to any vertical.

---

## Exercise 2: Create and edit a page in AEM

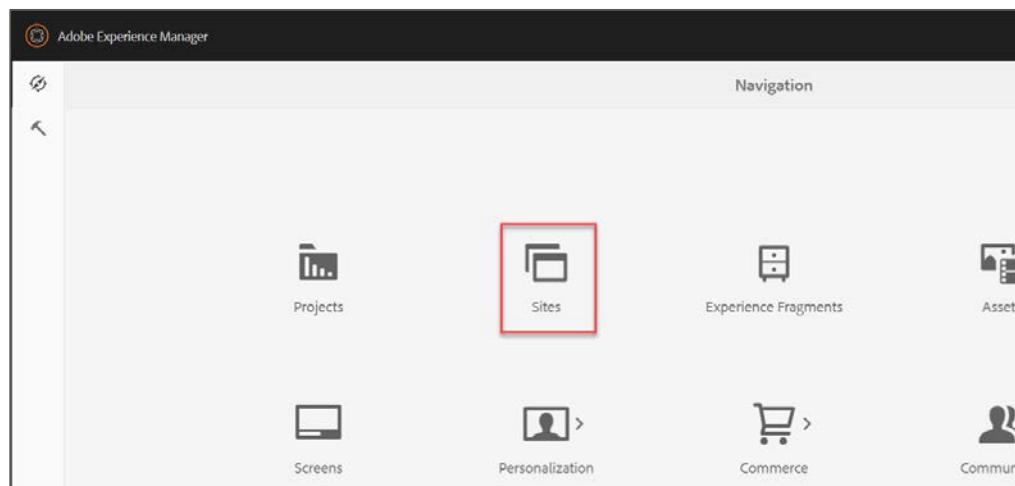
**Scenario:** As an author, you need create and edit a page in AEM Sites under the built-in We.Retail site hierarchy.

### Task 2.1: Create a page

In this task, you will create a demo page using the content template available in AEM.

To create a new page:

1. Ensure you have started and logged on to your AEM author instance on port 4502.
2. The **Navigation** page is displayed by default in the content area. Click **Sites** as shown below:

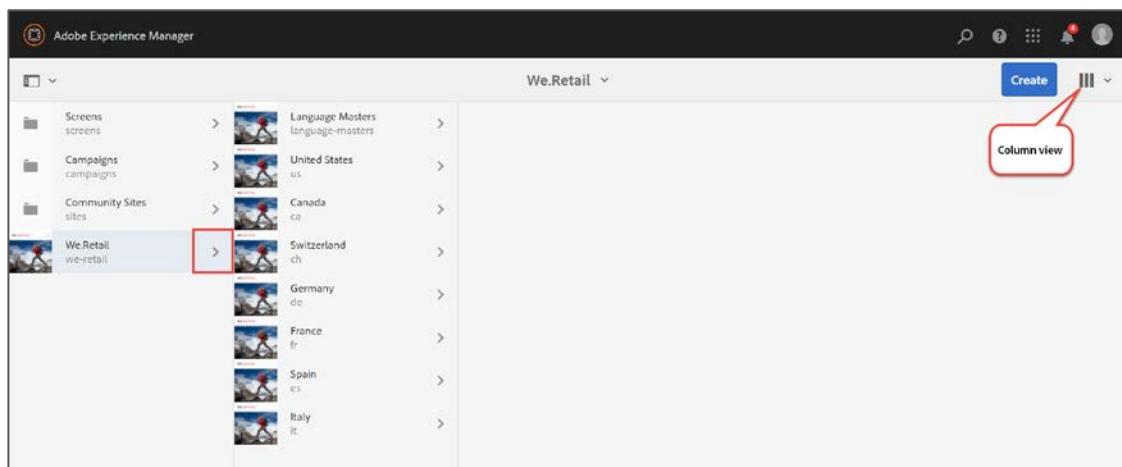


The column view is displayed.

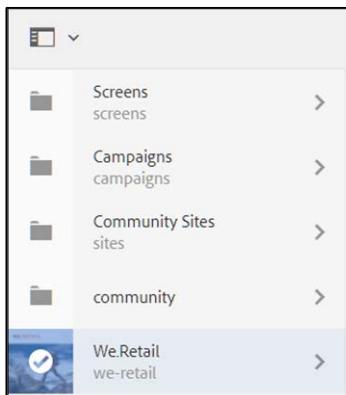


**Note:** You may see the **Product Navigation** tutorial dialog guiding you through the navigation. You may click **Next** to proceed through the tutorial and learn the basic AEM UI elements and navigation, or you may click **Close** to hide the tutorial.

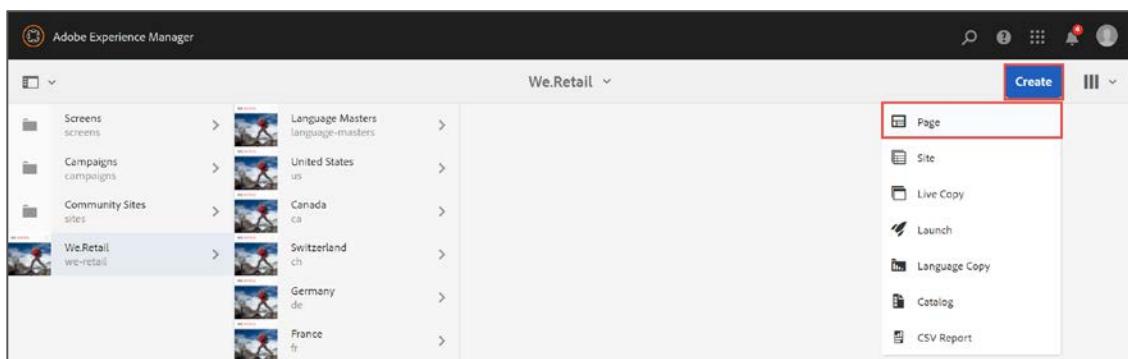
3. In the column view, click the right-pointing arrow next to We.Retail as shown:



**Tip:** If you see a check mark on the We.Retail thumbnail as shown below, it means you have *selected* We.Retail for editing and/or managing the page. Click the thumbnail again to clear the selection and click the right-pointing arrow instead.

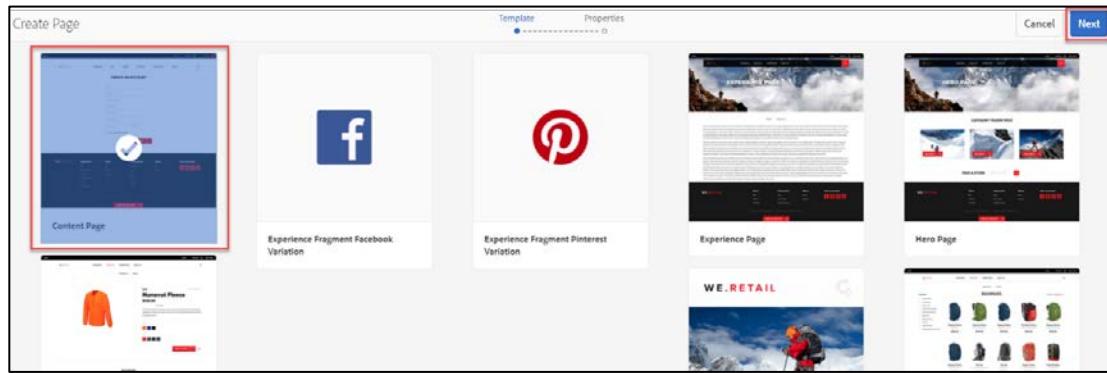


4. Click **Create > Page** as shown:

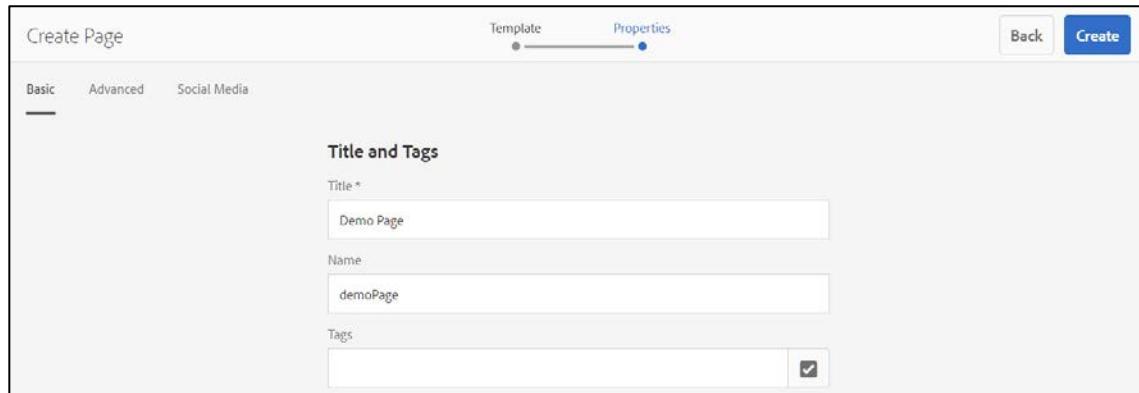


After you click **Page**, a wizard appears where you need to select a template for your page.

- Click the **Content Page** template to select it, and click **Next** as shown:



- In the **Properties** step of the page creation wizard, enter the following values for the corresponding fields:
  - Title: Demo Page**
  - Name: demoPage**



- Click **Create** in the top-right corner to create the page. A **Success** dialog box is displayed with a message that your page has been created.

8. Click **Done**. The new page appears as a child page of We.Retail. The page name (**Demo Page**) appears in the column view. You can also see the page title (**demoPage**) below the page name.

The screenshot shows the Adobe Experience Manager (AEM) interface. On the left, there is a navigation tree with categories like Screens, Campaigns, Community Sites, and a folder named 'community'. Under 'community', a folder named 'We.Retail' is selected and highlighted with a blue background. To the right of the tree, there is a grid view showing various language variants of the 'We.Retail' site. One row in this grid is highlighted with a red border, showing the details for a page named 'Demo Page' (with the internal name 'demoPage'). The properties for this page are listed on the far right:

	Title	Name
Template	Demo Page	demoPage
Modified	Content Page	
Modified By	3 minutes ago	
Language	Administrator	
Published	English	
	Not published	

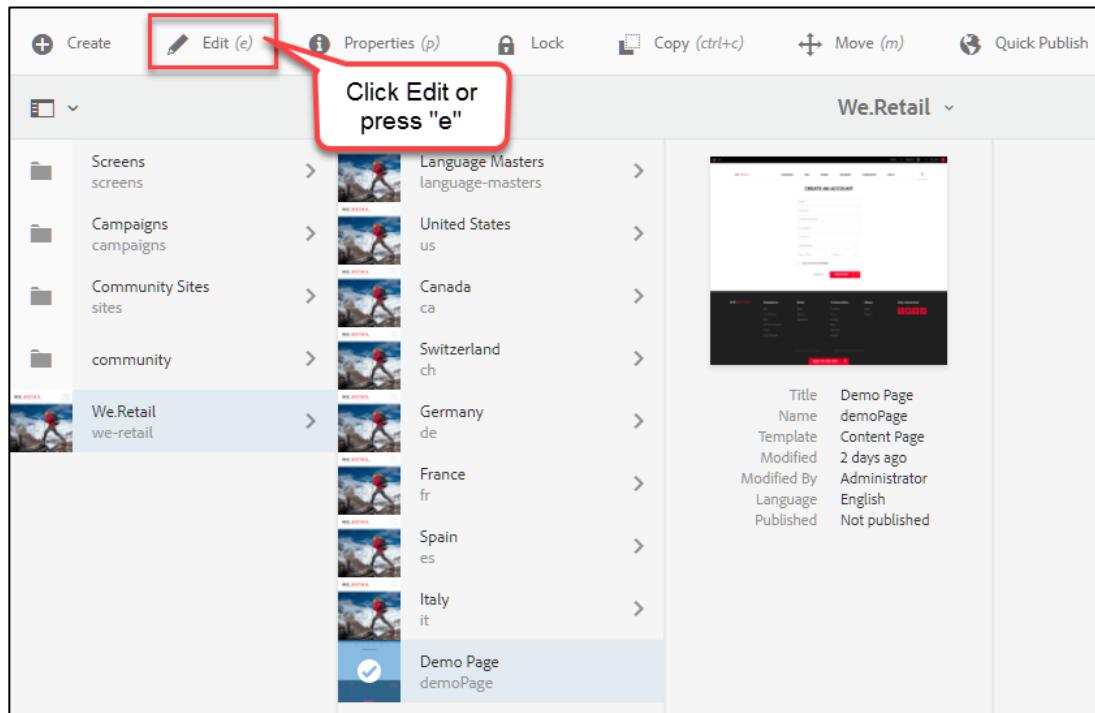
You can create subpages of any page using the same process.

## Task 2.2: Edit a page

In this task, you will edit the page you just created by adding text and image components.

To edit the page:

1. Click the **Demo Page** (thumbnail) you just created to select it, and click **Edit** from the actions bar. Be patient as it may take a few minutes to load the AEM page editor the first time.



---

 **Note:** You will notice the shortcut keys in the header bar. After you have selected a page, you may use the keyboard shortcuts such as **e** to edit the page, **p** to view page properties, and **Ctrl+c** to copy the page.

---

- When you first open the page in edit mode, you will most likely see a dialog box guiding you through the different modes. If you want to come back to this tutorial later to learn more about the AEM editor, clear the **Don't show this again** checkbox and then click **Close** as shown:

## Modes

The experience within the editor can be viewed in different modes. Most importantly, there is:

- Preview:** To navigate the site and see how the page looks once published.
- Edit:** To make changes to the experience.

The **Ctrl-Shift-M** keyboard shortcut allows you to quickly switch between preview and last selected mode.

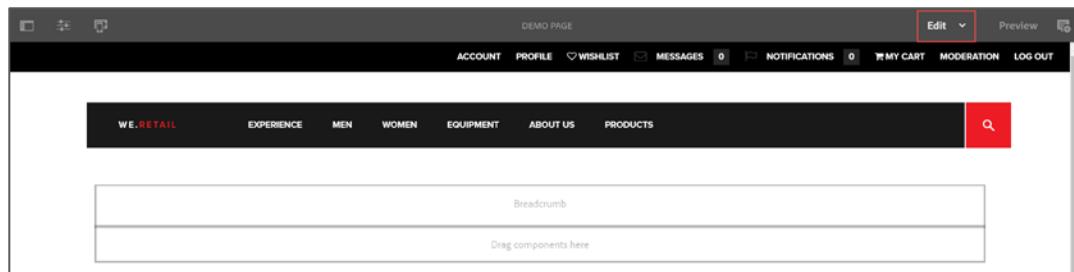
**i** Mind, to avoid accidentally navigating away from the page while editing, all links are disabled in Edit mode.

Don't show this again

• • • • •

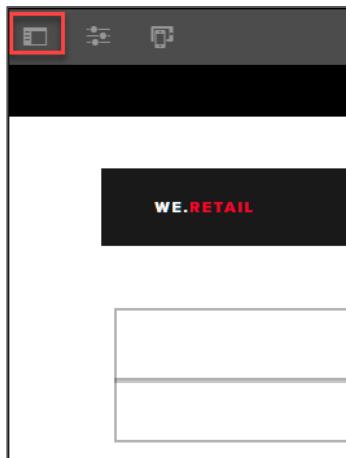
**Close** **Next**

Ensure the page is open in edit mode by checking in top-right corner to see the **Edit** mode highlighted:



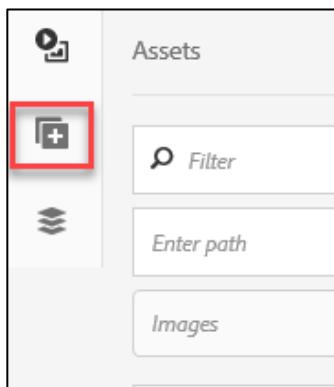
To add a text component to the page:

3. Click the Toggle Side Panel icon in the top-left corner as shown:

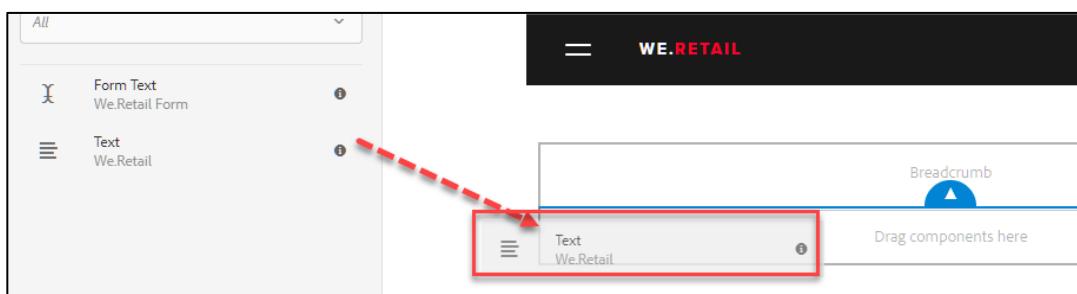


The side panel is displayed.

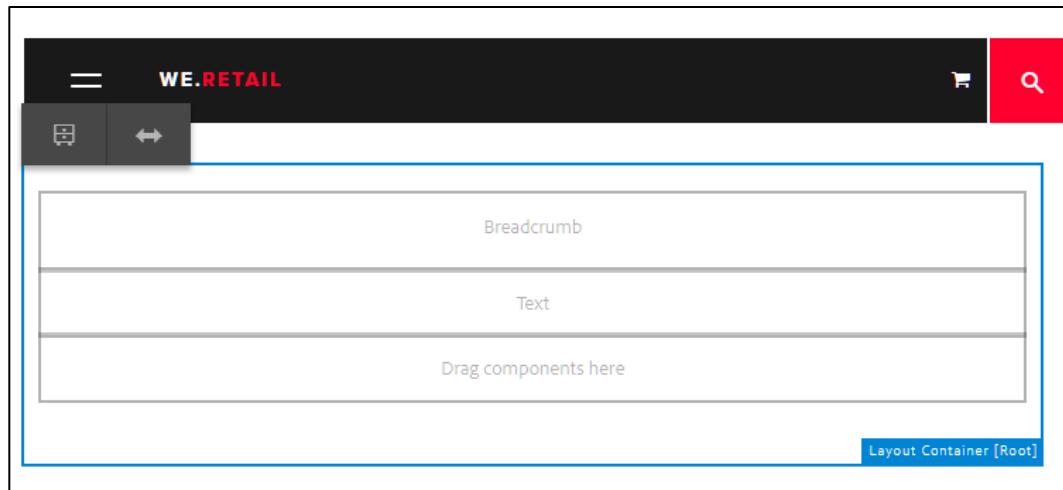
4. Click the Components icon as shown:



5. In the **Filter** field, enter **text** to search for the **Text** component, and press Enter. The search yields results that contain the word 'text'.
6. Drag the **Text (We.Retail)** component into the **Drag components here** box as shown:

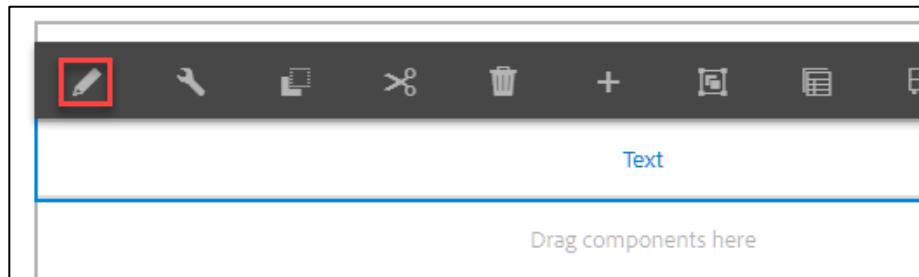


Your editor should look like the following:



**Tip:** If the ordering is not correct, just drag the text component again to the **Drag components here** box.

7. Click the **Text** component, and then click Edit (pencil icon) from the component toolbar as shown. A text editor opens in the same tab.



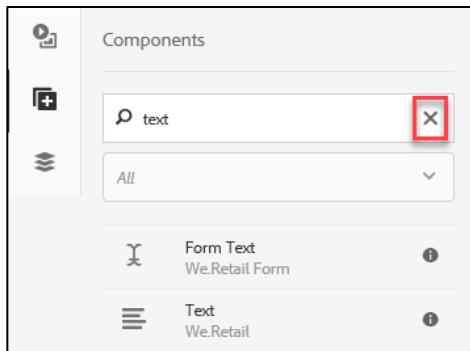
8. Add sample text of your choice in the text editing form that appears, and click Done (checkmark icon) to save the changes. The text is added to the page.



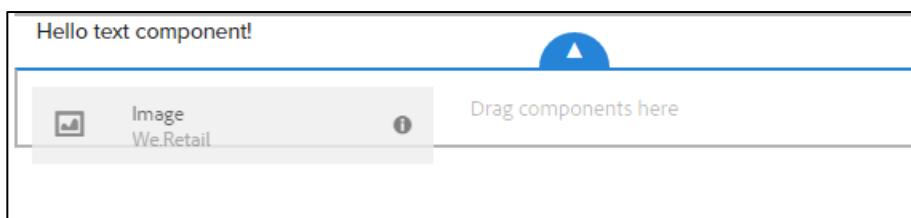
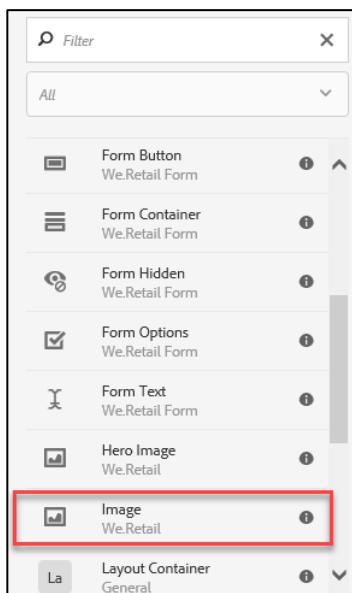
You have now added the first component to your page!

To add the image component to your page:

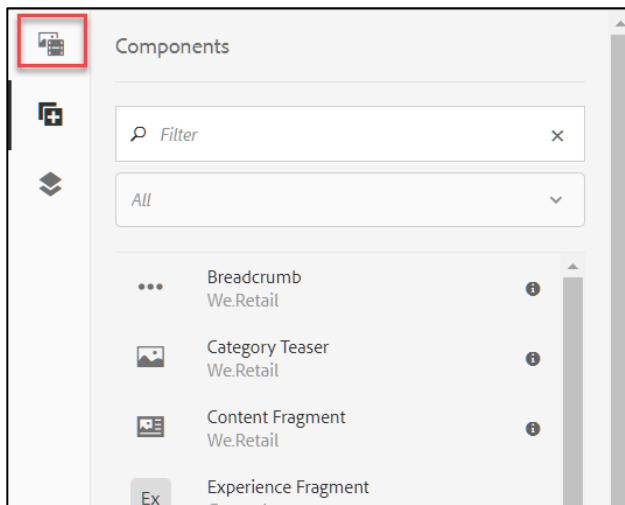
9. In the side panel, click **x** to clear the search as shown:



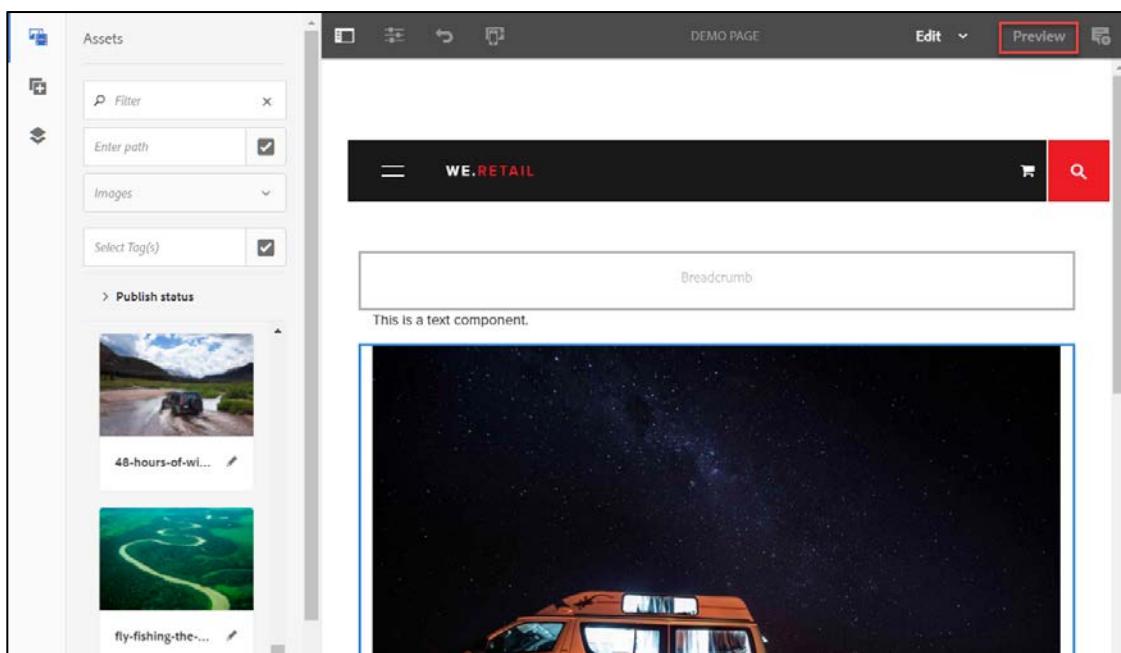
10. Scroll down and drag the **Image (We.Retail)** component onto the **Drag components here** box as shown:



- Click the Assets icon in the side panel as shown:



- Drag any image from the **Assets** tab on the **Image** component. AEM will add the image to the page.
- Click **Preview** in the top-right corner to preview the changes to your page:



- Press **CTRL+SHIFT+M** to go back to the edit mode. You may use this method to toggle between **Preview** and **Edit** mode, as you may need to switch often when adding and testing page content.  
Note that this keyboard shortcut is AEM specific and does not depend on the operating system or the browser. In other words, this shortcut is universal to AEM.

The only variation to this is the use of ⌘. The key ⌘ is specific to macOS, and is equivalent to CTRL in Windows.

---



**Note:** At this time, your page has been edited, but is not yet live. In order to push changes to content to the web, your page has to be published (or activated) to a publish instance of AEM.

---

# Developer Tools

The three most common developer tools available in AEM for developers and administrators are:

1. CRXDE Lite
2. Web Console
3. Package Manager

## CRXDE Lite

CRXDE Lite is embedded into AEM and allows you to perform common development and administration tasks within the browser. It is a light Integrated Development Environment (IDE) for quick access to the JCR. 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:** CRXDE Lite is primarily used by front-end developers who develop components and client libraries in AEM. DE in CRXDE stands for Development Environment.

You can access CRXDE Lite directly by navigating to <http://localhost:4502/crx/de/index.jsp> or by navigating to Tools > CRXDE Lite within AEM.

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.

You can use CRXDE Lite for:

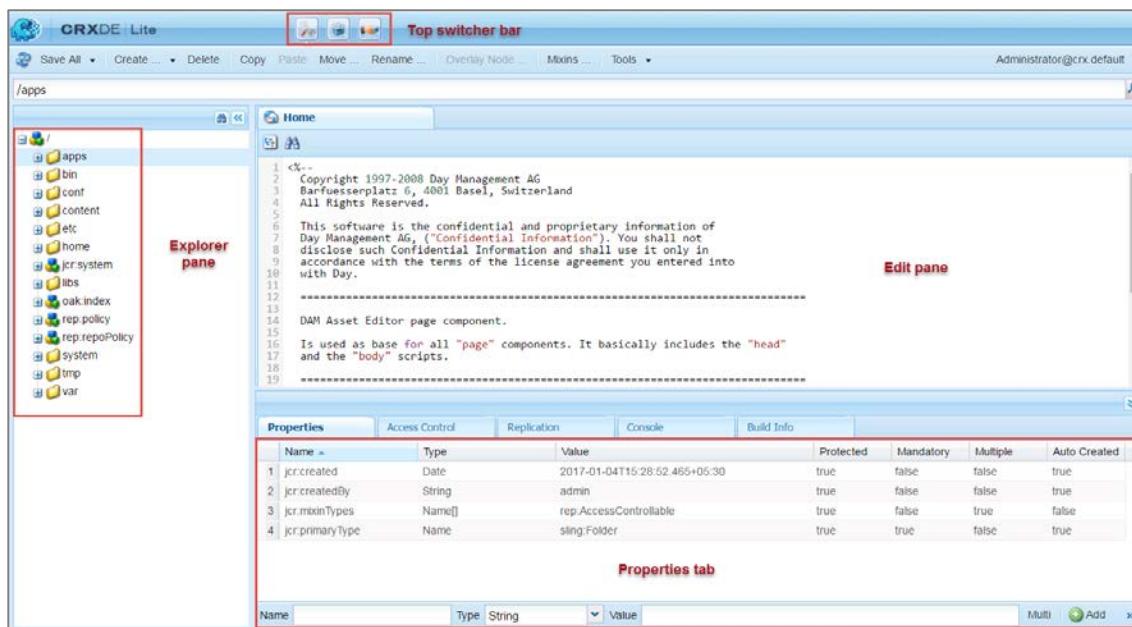
- Code/content validation
- Product training
- Operations debugging
- Overlays from /libs

The CRXDE Lite UI contains:

- **Top switcher bar:** Enables you to quickly switch between CRXDE Lite, Package Manager, and Package Share
- **Explorer pane:** Displays a folder tree structure 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, and creating a file.
- **Edit pane:** Allows you to edit the code. Double-click a file, such as a .jsp or a .html file, in the **Explorer** pane to display its content. 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.



## Web Console

The Web Console in AEM is based on the Apache Felix Web Management Console, and is used to manage OSGi 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 within AEM.



**Note:** If you use the navigation in AEM, the Web Console and CRXDE Lite always open in a new tab in your browser.

The Web Console contains a selection of tabs for maintaining OSGi bundles. 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 AEM.
- **Configuration**- Used for configuring OSGi bundle, and is the underlying mechanism for configuring AEM parameters.



## Package Management in AEM

A package is a \*.zip file that holds AEM repository content in the form of a file-system serialization called *Vault* serialization. Vault is provided by Apache Jackrabbit.

Packages provide an easy-to-use-and-edit representation of files, such as pages, assets, and folders. They also enable you to import and export repository content from one instance or environment to another.

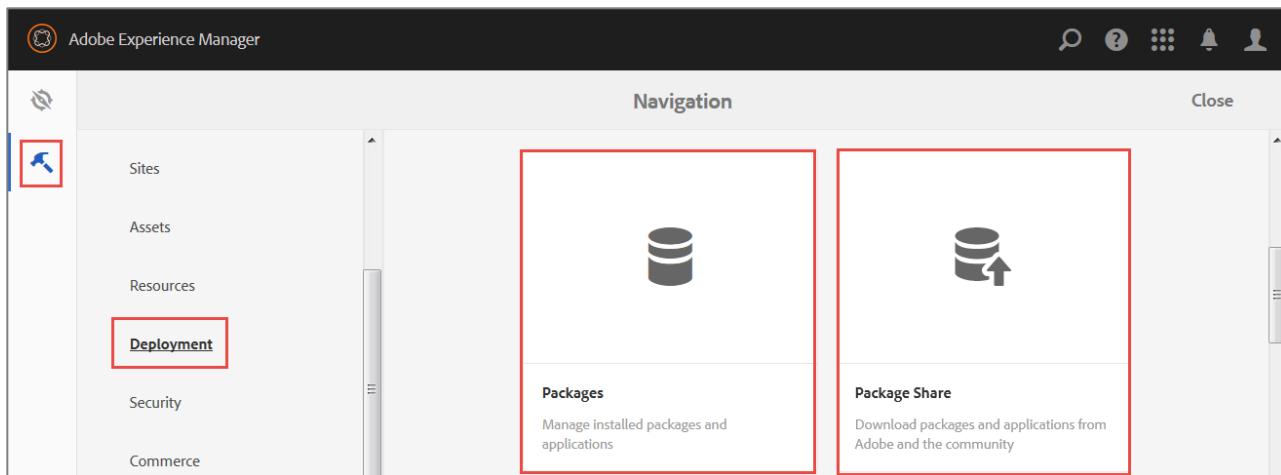
A package can contain:

- 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

You can 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 by using Package Manager or Package Share. You can access these options from the AEM UI by navigating to **Tools > Deployment > Packages/ Package Share** as shown, or through the following link: <http://localhost:4502/crx/packmgr/index.jsp>:



## Package Manager

Package Manager is used to import or export content on your instance, transfer content between instances, and back-up repository content. With Package Manager, you can perform the following common tasks:

- Create, build, and download content packages
- Upload, validate, and install packages
- Modify existing packages
- View package information

You can also use filters to create a package containing page content or project-related content.

### Creating and Building New Packages

When creating packages using Package Manager, you can apply rules and filters to determine the content a 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 the File System

You can download a package by clicking the download link. This link is displayed when the package details are expanded. After downloading the package, you can unzip the contents of the package to your local system.

Typically, an unzipped (extracted) content package contains the following folders:

- **jcr\_root**: Contains files and folders that are serialized nodes and properties from the JCR.
- **META-INF**: Contains metadata regarding node definitions and the filter.xml file that gives directions to Vault about the paths to include.

### Package Share

Package Share is a centralized server where public packages are made available. These packages may include hotfixes, new functionality, updates or documentation. 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:

- AEM packages provided by Adobe (For example, new functionalities such as updated core components, hotfixes and service packs)
- Shared packages provided by other organizations and made public by Adobe

You can access this console through the following link: <http://localhost:4502/crx/packageshare/index.html>

You can also directly access Package Share (without using CRXDE Lite) through the following link (after signing in using your Adobe ID): <https://www.adobeaecloud.com/content/packageshare.html>

# Exercise 3: Install, create, build, and download a package

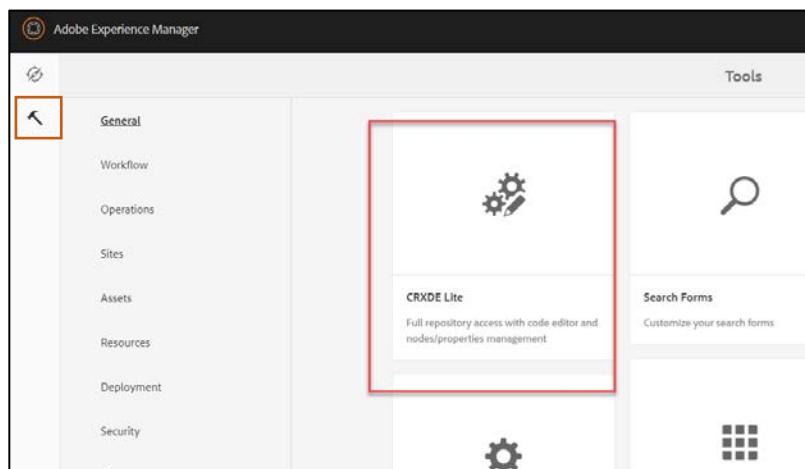
**Scenario:** As an AEM developer or administrator/development operations, you need to own all package management tasks including validating, installing, creating, building and downloading packages in AEM.

## Task 3.1: Install a package

In this task, you will validate and install a package using the Package Manager.

To validate a package:

1. Ensure that you are logged on to your AEM author instance on port 4502 (<http://localhost:4502>).
2. Navigate to Tools > CRXDE Lite in your AEM author instance as shown:

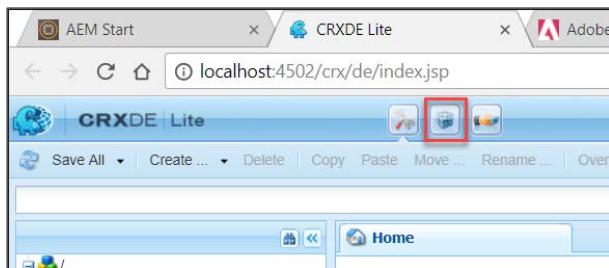


The **CRXDE Lite** console loads in a new browser tab.



**Tip:** Bookmark the **CRXDE Lite** URL (<http://localhost:4502/crx/de/index.jsp>) in your browser to access this tool, as you will use it often in your training as well as in your role as an AEM developer or administrator.

3. In the CRXDE Lite console, click the **Package** icon as shown:



This will take you to the AEM Package Manager tool.

---

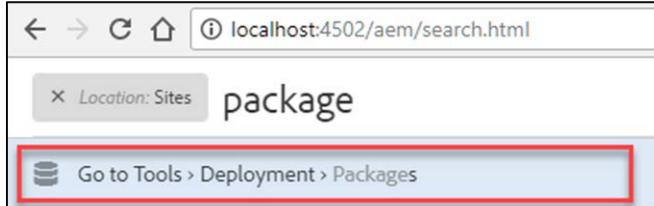
**Tip:** Bookmark the **Package Manager** URL (<http://localhost:4502/crx/packmgr/index.jsp>) in your browser in addition to CRXDE Lite.

---

---

**Tip:** Other ways to navigate to **Package Manager** include:

- Use Tools > Deployment > Packages within AEM.
- Type / within AEM to use the AEM Omnisearch feature. Type **package** and click the result:

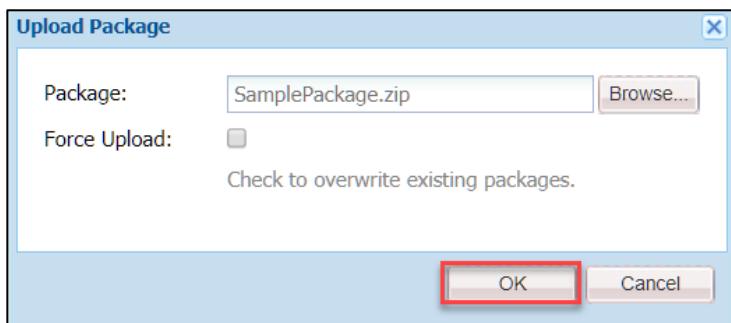


4. Click **Upload Package**.



5. In the **Upload Package** dialog box, click **Browse** and select the **SamplePackage.zip** package from the **Exercise\_Files** (in the \Module 4 – Package Manager folder) provided to you.

6. Click **Open**, and then click **OK** as shown:



Your uploaded package is now available in AEM Package Manager as shown:

SamplePackage.zip  
Build: 1 | Last built 2016/04/14 | admin

Edit Build Install Download Share More ▾

Package: SamplePackage  
Download: SamplePackage.zip (5.5 MB)  
Group: training  
Filters: /content/dam/assets-from-sample-package

7. Click **More > Validate** as shown:

SamplePackage.zip  
Build: 1 | Last built 2016/04/14 | admin

Edit Build Install Download Share More ▾

Package: SamplePackage  
Download: SamplePackage.zip (5.5 MB)  
Group: training  
Filters: /content/dam/assets-from-sample-package

we.retail.community.apps-1.11.84.zip  
Version: 1.11.84 | Last installed Mar 7 | admin  
Profiles used for exporting We.Retail Community Site Apps.

More ▾

- Delete
- Coverage
- Contents
- Rewrap
- Other Versions
- Uninstall
- Test Install
- Validate
- Replicate

- Leave all options selected and click **Validate** in the dialog box. In the **Activity Log** below, notice there are no issues with your package:

The screenshot shows the 'Activity Log' window with the following text:  
**Validate Package: /etc/packages/training/SamplePackage.zip**  
Thu Mar 08 2018 15:53:49 GMT-0800 (Pacific Standard Time)  
Validating content package  
  
No unsatisfied OSGi package imports. No overlay rebase warnings. No ACL warnings.  
Package validated in 0ms.

**Note:** As a best practice, you should validate all packages before installing to check for any warnings. While this validation tool is covered in depth in other training courses, this utility will notify you of any issues that impact overlaid JCR resources in /apps, any unsatisfied bundles, and any ACL (Access Control Lists) conflicts. In other words, this will prevent any problems with OSGi bundles that are unable to start after installing a package, any impacts on permissions (ACLs), and files in /libs that impact overlaid files in /apps.

To install your package:

- Click **Install** as shown:



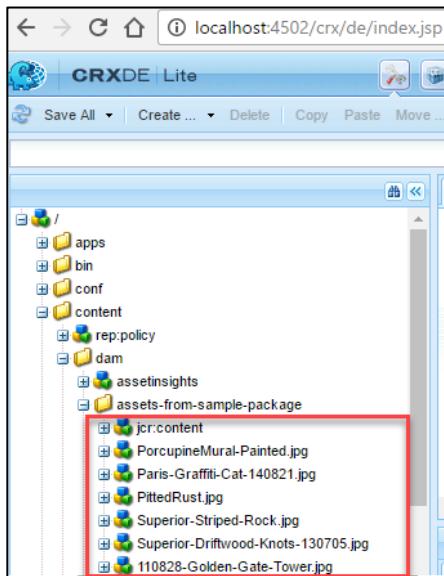
The **Install Package** dialog box appears.

- Ignore the **Advanced Settings** area and click **Install**.
- Check the **Activity Log**. You can see the content was added from the package. In the activity log, note that the contents of the package consist of a set of image assets being added to /content/dam/assets-from-sample-package in the JCR repository.
- Click the Develop icon as shown:



This takes you back to **CRXDE Lite**.

13. Navigate to the **/content/dam/assets-from-sample-package** folder in the **Explorer pane** (at left). The package contents have been added to the JCR in this location as shown:



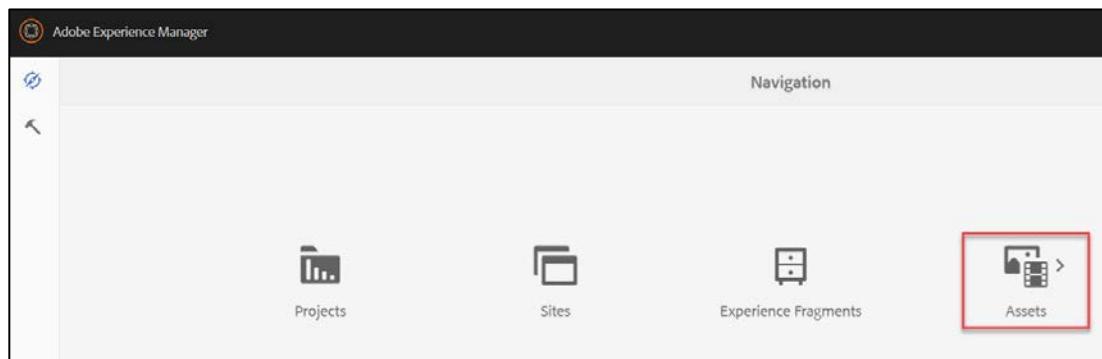
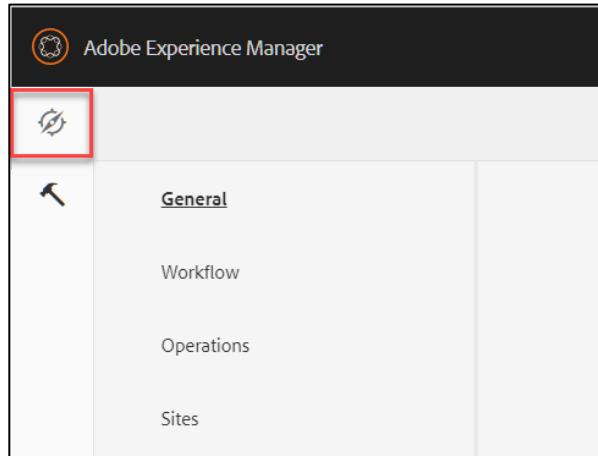
14. Click one of the nodes (such as **Superior-Striped-Rock.jpg**). Note that in the **Properties** tab, properties of that node are shown. In addition, each file has a child **jcr:content** node. So, at a basic level, all content in the JCR consists of nodes and properties.

Properties		
Name	Type	Value
1 jcr.baseVersion	Reference	30496f6
2 jcr.created	Date	2018-03
3 jcr.createdBy	String	admin
4 jcr.isCheckedOut	Boolean	true
5 jcr.mixinTypes	Name[]	mix:vers

The screenshot shows the CRXDE Lite interface with the 'Properties' tab selected for the 'Superior-Striped-Rock.jpg' node. A red box labeled 'Node' points to the node in the tree view on the left. A red box labeled 'Properties' points to the table in the main pane. The table lists the following properties:

Name	Type	Value
1 jcr.baseVersion	Reference	30496f6
2 jcr.created	Date	2018-03
3 jcr.createdBy	String	admin
4 jcr.isCheckedOut	Boolean	true
5 jcr.mixinTypes	Name[]	mix:vers

15. In your other browser tab, navigate back to the AEM start (<http://localhost:4502>).
16. Click the Navigation icon, then click **Assets** as shown:



17. Click the **Files** folder. You can now view the set of images in the **Assets from Sample Package** folder that were installed through the package.



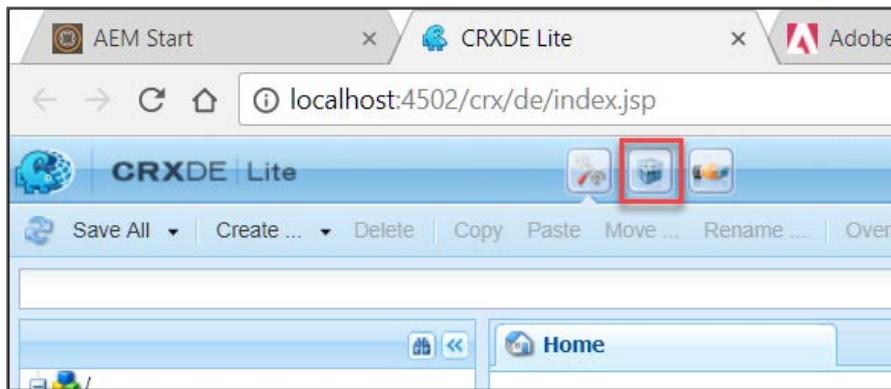
**Note:** You may see a **Product Navigation** tutorial dialog guiding you through the navigation. You may click **Next** to proceed through the tutorial and learn the basic AEM user interface elements and navigation, or you may click **Close** to hide the tutorial.

## Task 3.2: Create, build, and download a package

In this task, you will use the Package Manager to package We.Retail content for distribution to another environment.

To create a package:

1. In your CRXDE Lite browser tab, click the **Package Manager** icon as shown:



The **Package Manager** screen appears.

2. Click **Create Package** as shown:



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

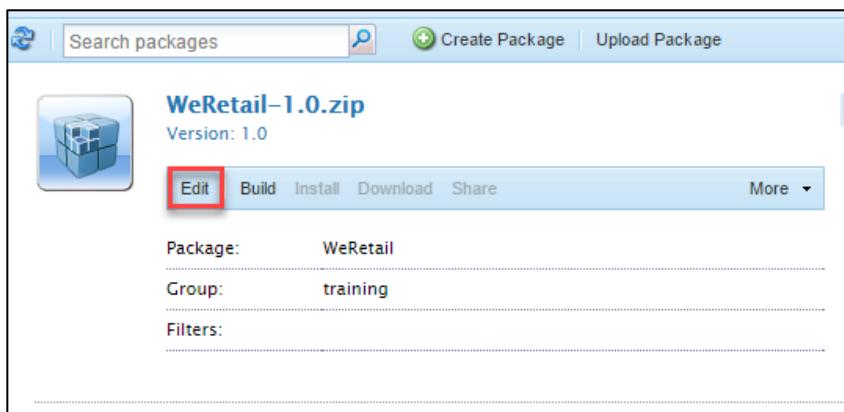
**Package Name:** We.Retail

**Version:** 1.0

**Group:** training

4. Click OK. The **We.Retail-1.0.zip** package is created.

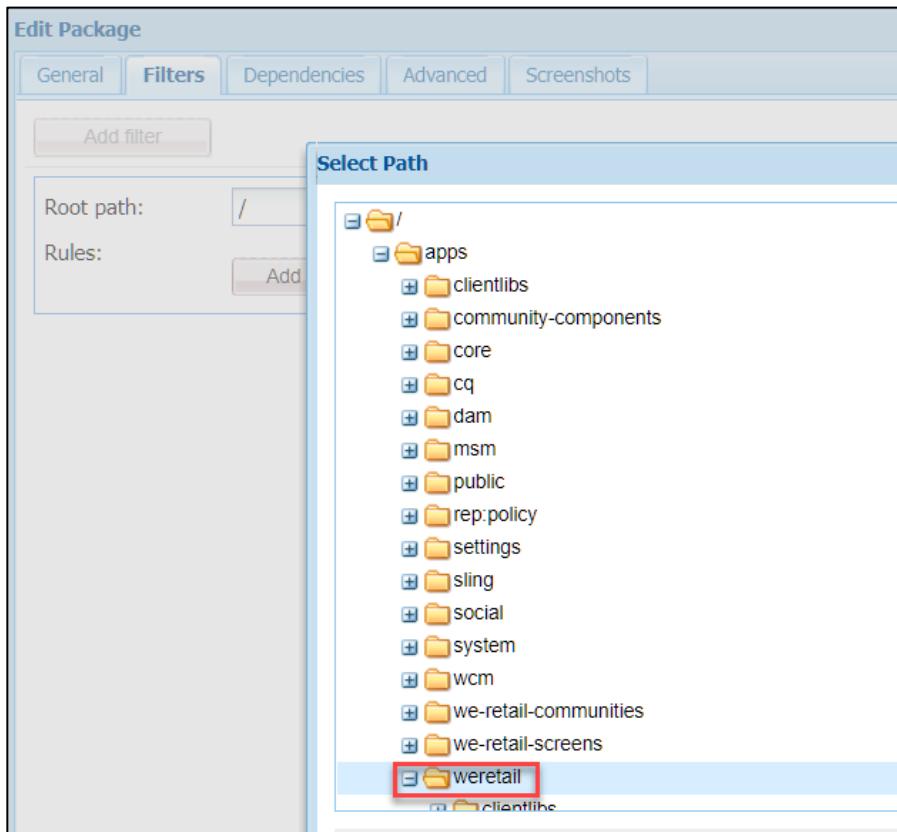
5. Click **Edit** on the newly created package as shown:



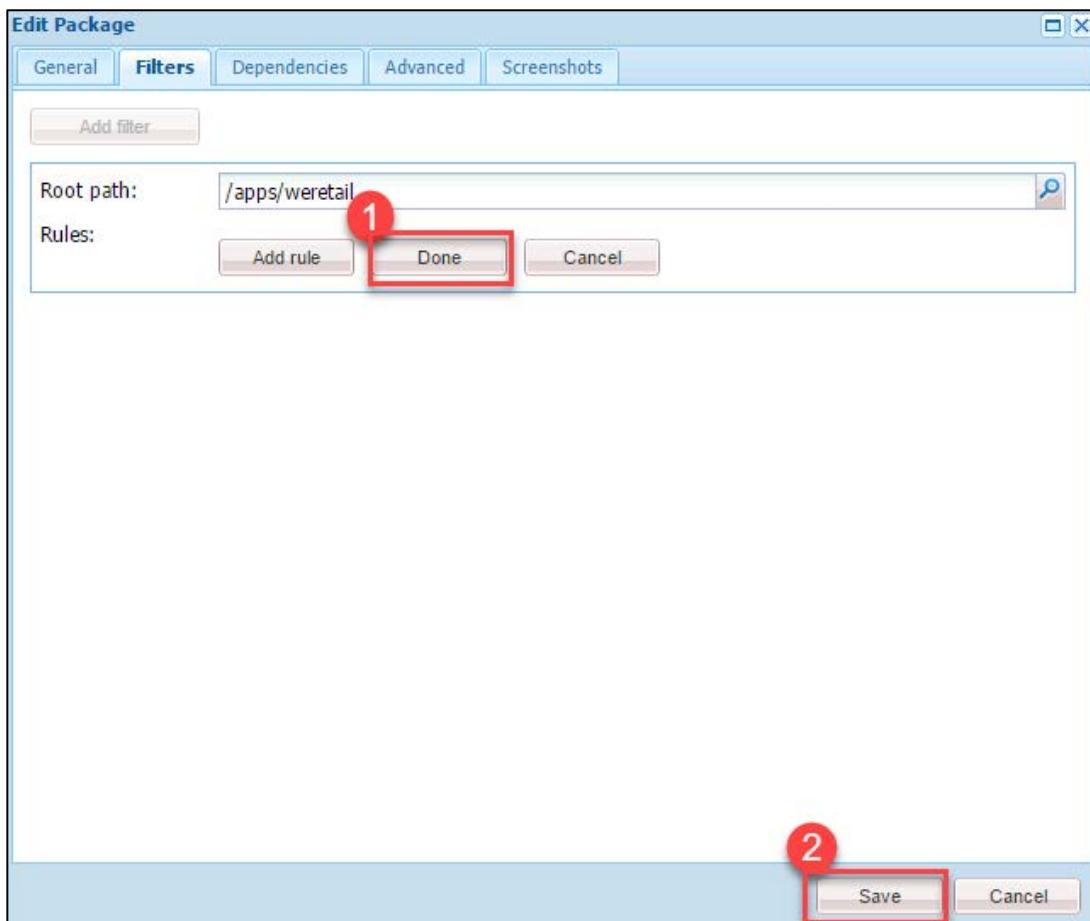
6. To add filters to the package, click the **Filters** tab, and click **Add Filter**.

 **Note:** Filters are the mechanism used to add content to packages. Here, you specify the paths that contain the content from the JCR you want to include in a package. Before adding filters, your package is completely empty. You may also restrict filetypes added to a package using filter rules, such as excluding all \*.txt files.

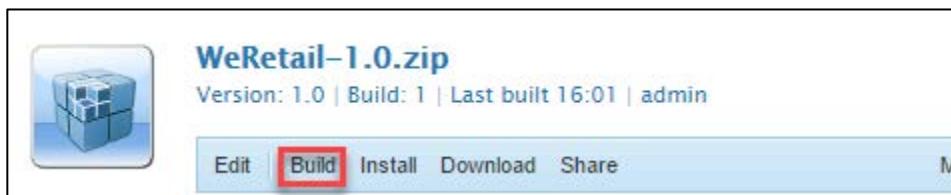
7. For the **Root** path, browse and select the **weretail** project folder under **apps**, and click **OK**.



8. Click **Done**, and then click **Save**, as shown:



9. Click **Build** to build the package as shown:



 **Note:** As the package is being built, the activity log is running at bottom, showing a series of "A" actions in the log. "A" denotes a node is being added to the content package.

10. Click **Build** again in the confirmation dialog box. The package is now ready for download.
11. Click the **Download** link to download a copy of the package to your computer. The package is download to your computer's default download folder for the browser.



**Note:** You should now see a list of all packages in your instance organized by group on the left-hand menu. Note that there are now two packages in the **training** group – the first package that you installed in Task 3.1 and the second package you just created for We.Retail. Therefore, you may use the group name to categorize and organize your packages in AEM.

The screenshot shows a list of package groups on the left side of the AEM interface. The 'Groups' heading is at the top. Below it, a list of groups is shown, each with a count of packages in parentheses. The 'All packages (163)' group is highlighted with a blue bar. The 'training (2)' group is highlighted with a red box. Other groups listed include 'my\_packages', 'Adobe (72)', 'com.adobe.cq.inbox (1)', 'day (88)', 'fd', 'rep:policy', and 'sling'.

Groups	Count
All packages	(163)
my_packages	
Adobe	(72)
com.adobe.cq.inbox	(1)
day	(88)
fd	
rep:policy	
sling	
training	(2)

# References

AEM 6.4 Help Content (main page): <https://helpx.adobe.com/support/experience-manager/6-4.html>

Architecture:

- OSGi: <http://www.osgi.org>
- JCR 2.0 Specification: [http://www.day.com/specs/jcr/2.0/2\\_Introduction.html](http://www.day.com/specs/jcr/2.0/2_Introduction.html)
- JSR-283: <https://jcp.org/en/jsr/detail?id=283>
- Apache Sling: <https://sling.apache.org/>
- Apache Felix: <https://felix.apache.org/>
- Granite UI (based on Coral 3): <https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/touch-ui-concepts.html>

Installation:

- Official list of supported platforms for AEM: <https://helpx.adobe.com/experience-manager/6-4/sites/deploying/using/technical-requirements.html>
- Local Machine Operating System Requirements for AEM: <https://helpx.adobe.com/experience-manager/6-4/sites/deploying/using/deploy.html#GettingStarted>

Authoring:

- Authoring Pages: <https://helpx.adobe.com/experience-manager/6-4/sites/authoring/using/page-authoring.html>
- You can download the latest release of the We.Retail site using the following link:  
<https://github.com/Adobe-Marketing-Cloud/aem-sample-we-retail/releases>

Developer Tools:

- Vault FS on Apache Jackrabbit Documentation: <https://jackrabbit.apache.org/filevault/vaultfs.html>
- Package Share Direct URL (No need to access through CRXDE Lite):  
<https://www.adobeaecloud.com/content/packageshare.html>
- Bookmark/Favorite these sites for a local author installation (development environment):
  - CRXDE Lite: <http://localhost:4502/crx/de/index.jsp>
  - Web Console: <http://localhost:4502/system/console>
  - Package Manager: <http://localhost:4502/crx/packmgr/index.jsp>

# Configuring AEM Development Environment



## Introduction

You can use Maven and Eclipse to configure Adobe Experience Manager (AEM) development environment. Maven builds and deploys bundles to Java Content Repository (JCR), and Eclipse helps developers edit code, build Open Services Gateway Initiative (OSGi) bundles, and perform unit tests. These third-party tools help a developer to comprehend the development effort easily and accurately.

Maven is the recommended build management tool for AEM projects. The following steps are required to develop an AEM project in Eclipse:

- Install Eclipse
- Install Eclipse AEM plugin
- Set up your AEM project based on Maven
- Build and deploy your AEM project

## Objectives

After completing this course, you will be able to:

- Describe Maven
- Install and configure Eclipse
- Install and configure the Eclipse AEM plugin
- Set up AEM for Eclipse
- Build and deploy an AEM project

# Working with Maven

Maven helps build and manage Java-based projects. Maven specifies how software is built and describes the dependencies of the software. Maven has a central repository from which Maven can dynamically download Java libraries and plugins and store them in a local cache. This cache can also be updated by the developers 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.

**Building your AEM project based on Maven offers you the following benefits:**

- Integration Development Environment (IDE)-agnostic development environment
- Usage of Maven Archetypes and Artifacts provided by Adobe
- Usage of Apache Sling and Apache Felix tool sets for Maven-based development setups
- Ease of project import into an IDE
- Easy integration with Continuous Integration Systems

## Contents of a POM File

POM files are XML files that contain the identity and the structure of the project, build configuration, and other dependencies. When executing a task or goal, Maven looks for the POM in the current directory. Maven reads the POM, gets the needed configuration information, then executes the goal. A typical POM file includes:

- General project information: This section includes the project name, website URL, and the organization name. It can also include a list of developers, 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 section includes the profiles that can be used 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: This section includes the interdependencies between projects and their POM settings inherited from parent POMs. These interdependencies include:
  - GroupId
  - ArtifactId
  - Version
  - Dependencies



## Super POM

A Super POM is:

- Extension of Maven project PMOs
- Stored in \${M2\_HOME}/lib/maven-xxx-uber.jar in a file named pom-4.0.0.xml in the org.apache.maven.project package
- Part of the Maven installation

The default Super POM defines a single remote Maven repository with an ID of central. This is the central repository that all Maven clients are configured to read from by default. A custom **settings.xml** file can override this setting.

- Snapshot versions: This section includes snapshot versions used for projects under active development. Snapshot versions are indicated by the string '-SNAPSHOT'. 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: This section includes the properties from another part of the POM file, Java system properties, or implicit environment variables. It supports three environment variables:
  - env
  - project
  - settings
- Plugins: This section includes plugins that provide functionalities, such as compiling source, packaging a WAR file, or running JUnit tests. These plugins are retrieved from the Maven repository. If a new functionality is added to a plugin, you can use it by updating the version number of the plugin in a single POM configuration file.

## UberJar

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

The UberJar file contains:

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

## Using the UberJar File

To use the UberJar file, you must add the following elements to the 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 the Maven repository manager that you are using 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. For AEM projects, you need to install Eclipse and the following plugins:

- Maven Integration for Eclipse (M2E)
- AEM plug-in for Eclipse

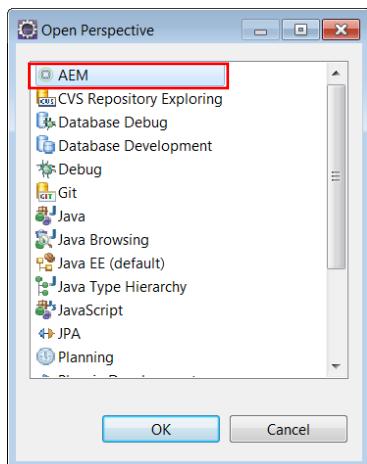
## Setting up the AEM Plug-in for Eclipse

AEM plug-in has the following benefits:

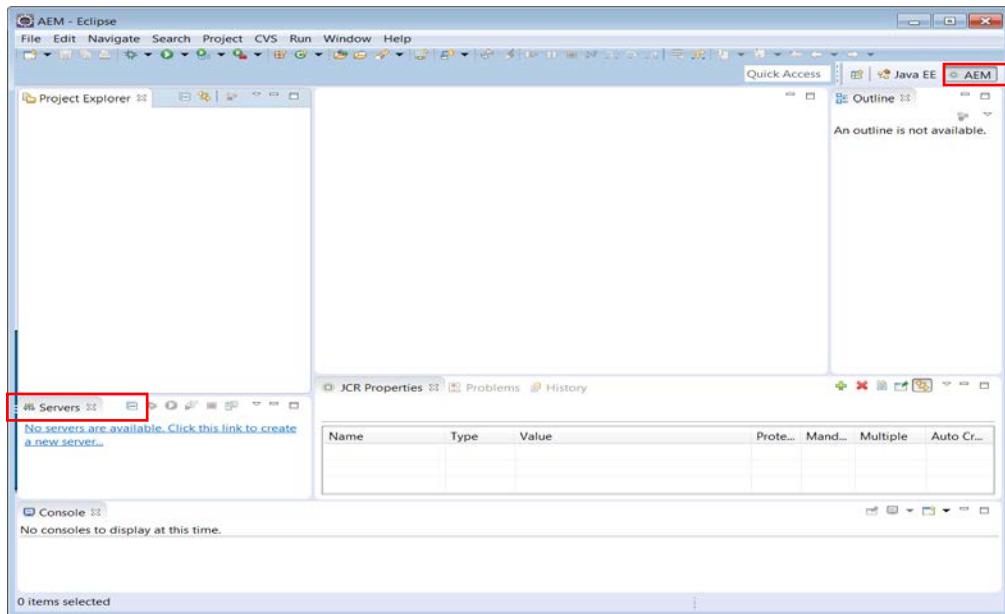
- Synchronizes both content and OSGi bundles
- Supports debugging and code hot-swapping
- Includes a project creation wizard to simplify bootstrapping of AEM projects
- Integrates with AEM instances seamlessly through Eclipse Server Connector
- Easy JCR properties edition

## Configuring the AEM Perspective

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



Using AEM Perspective, you can configure an AEM Server to which Eclipse will connect.



The AEM perspective enables you to add and modify nodes and properties in your AEM project through the AEM Console and the JCR properties view.

# Exercise 1: Install and configure Eclipse (Optional)

In this exercise, you will install and configure Eclipse.

This exercise includes two tasks:

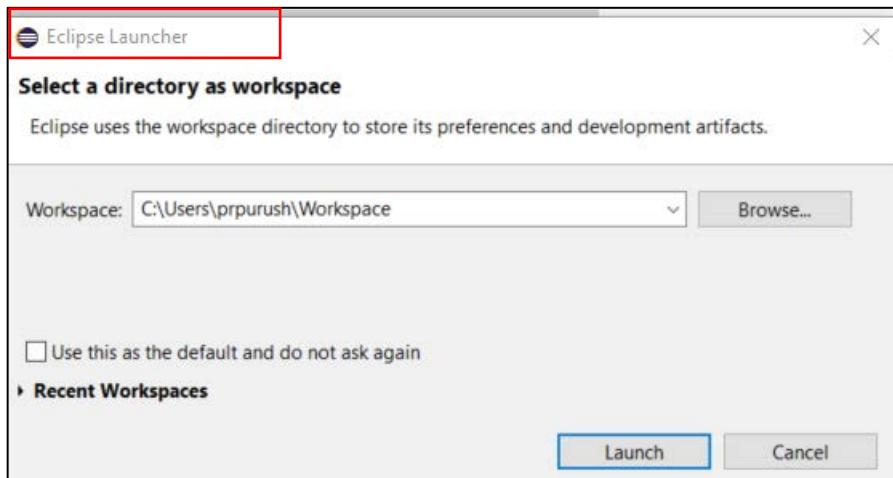
1. Install Eclipse
2. Configure Eclipse



Note: You can skip this exercise if you use a ReadyTech environment because AEM is already installed as part of the image.

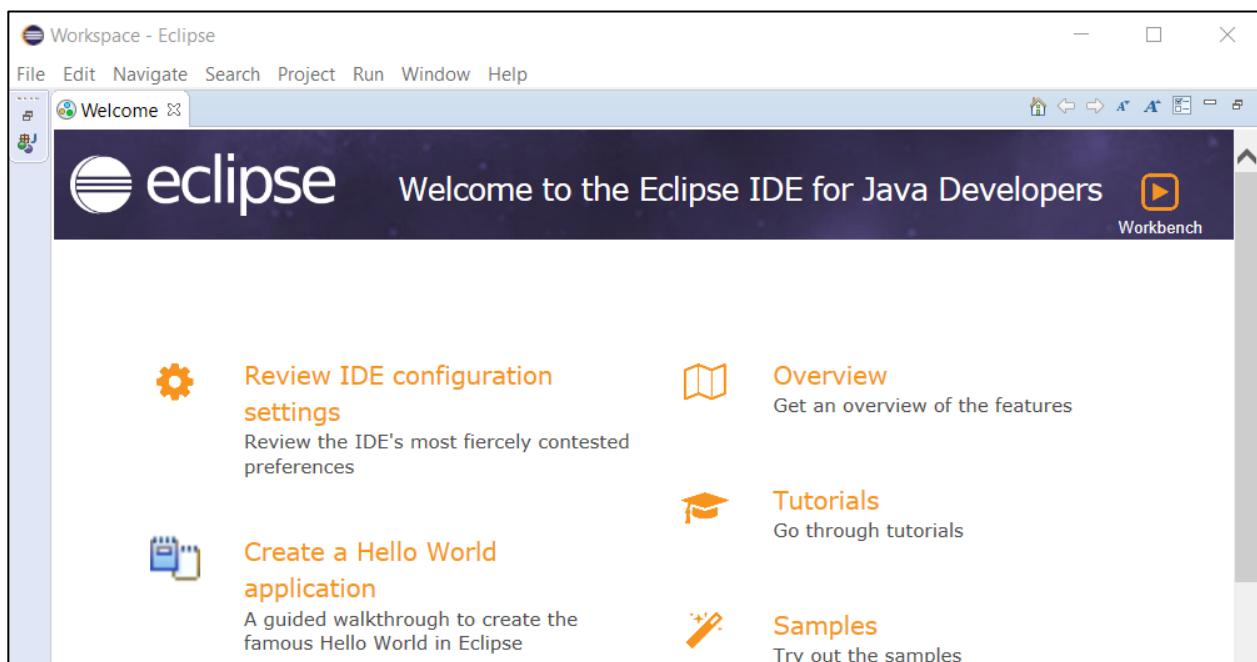
## Task 1: Install Eclipse

1. Extract files from the **Exercises\_Files.zip** file to the destination directory of your choice.
2. Navigate to the directory where you extracted the contents of the Eclipse installation zip file. For example, navigate to **C:\Program Files\Eclipse\** on Windows or **Applications/Eclipse** on Mac. On Mac, you may need to use the sudo command to navigate to the Eclipse directory.
3. Double-click **eclipse.exe** (or **eclipse.app**) to start Eclipse. The Eclipse Launcher opens, as shown:



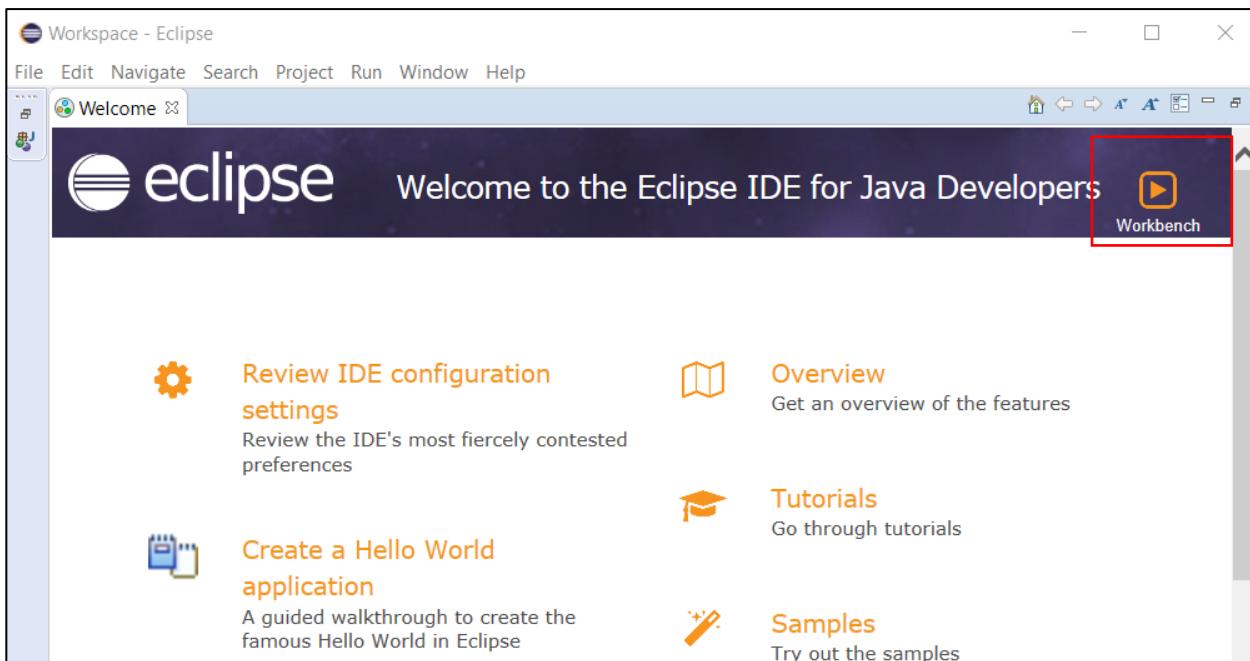
+

4. Accept the default workspace, and click **Launch**. The Eclipse Development Environment opens, as shown:

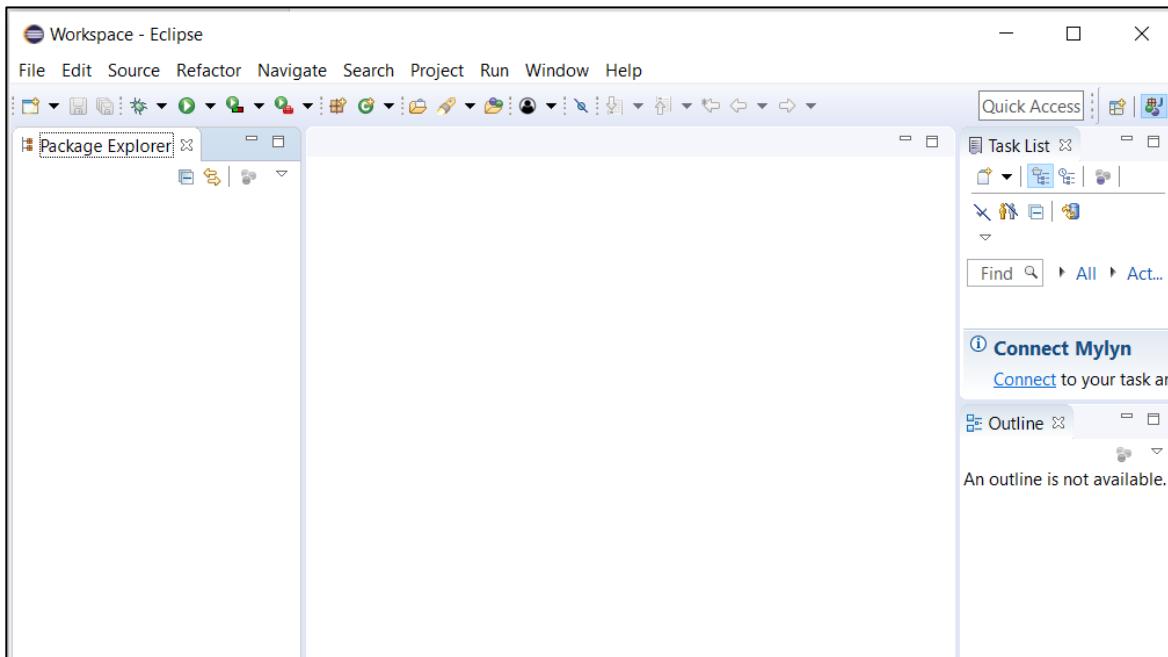


## Task 2: Configure Eclipse

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

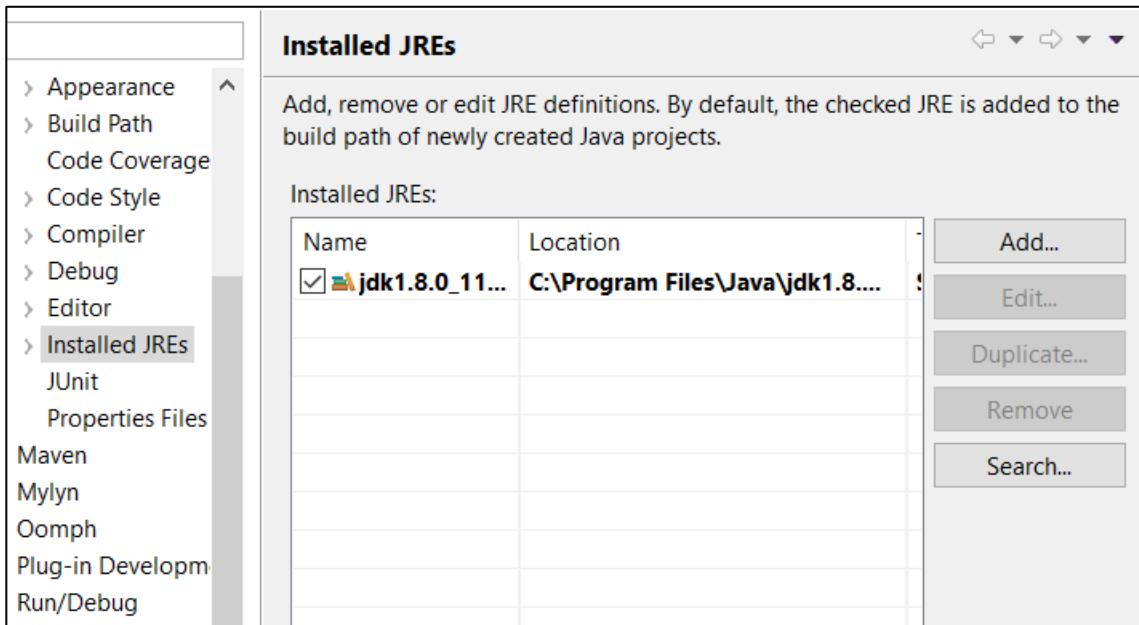


The Workbench opens, as shown:



2. Verify Eclipse's JRE is set to a JDK by clicking **Window > Preferences > Java > Installed JREs**.

You should see a path similar to the one given in the following screenshot. Otherwise, you must provide the correct directory path to your JDK.



## Exercise 2: Install and configure the Eclipse AEM plug-in (Optional)

In this exercise, you will install and configure the Eclipse AEM plug-in.



Note: You can skip this exercise if you use a ReadyTech environment because AEM is already installed as part of the image.

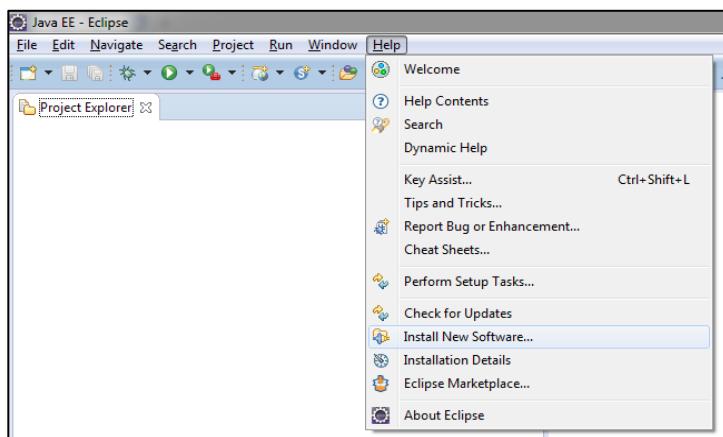
1. Open the URL <https://eclipse.adobe.com/aem/dev-tools/> or use the files provided in USB contents.

**Note:** There are two ways to install the plug-in:

- **Online:** You will provide the link to install the plug-in in Eclipse.
- **Offline:** You will provide the downloaded plug-in in Eclipse.

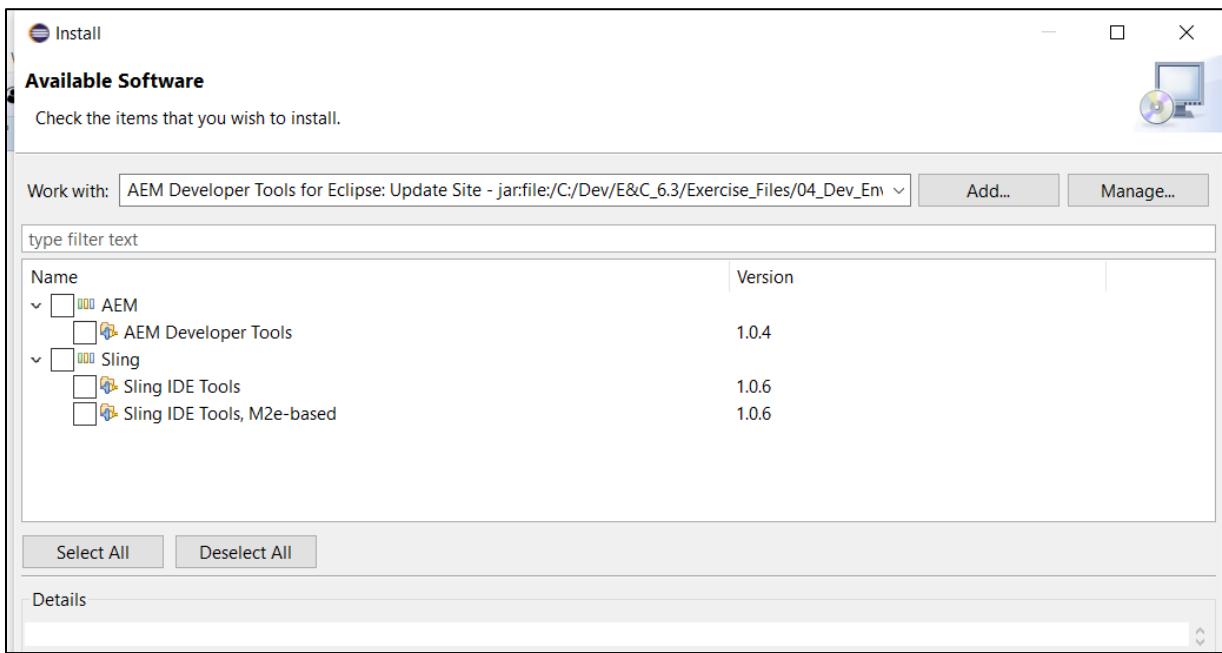
You will perform the offline method in this exercise.

2. Double-click `eclipse.exe` (or `eclipse.app`) to start Eclipse. The Eclipse Development Environment opens.
3. Select **Help > Install New Software**, as shown. The **Install** window opens.

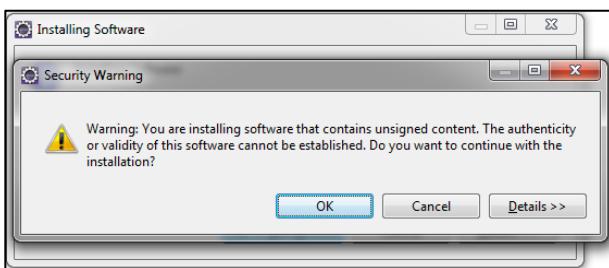


4. Click **Add**. The **Add Repository** dialog box opens.

5. Click **Archive**.
6. Navigate to the repository archive and select the zip file (**com.adobe.granite.ide.p2update-1.1.0.zip**) provided for the plug-in from **Exercise\_Files/05\_Dev\_Environment/**.
7. Click **Open**. The location of the repository is added.
8. Click **OK**. The **Available Software** window with the items that you want to install opens, as shown:



9. Click **Next**. The **Review Licenses** screen opens.
10. Click the **I accept the terms of the license agreements** radio button and click **Finish**. The **Installing Software** dialog box with a progress bar opens. The installation should take a minute or two.
11. If a **Security Warning** window pops up, as shown, click **OK** to continue the installation. The **Installing Software** dialog box will re-open until the installation is completed.



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

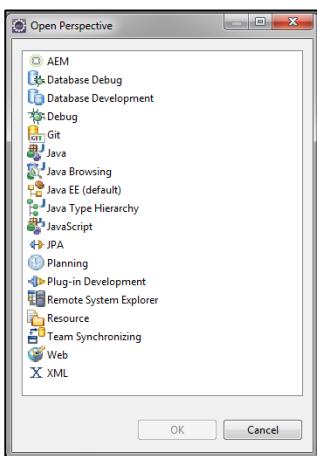


**Note:** This process takes about a minute. If you see a dialog box that asks if you want to keep the current location of your Eclipse install, click **OK**. Eclipse opens.

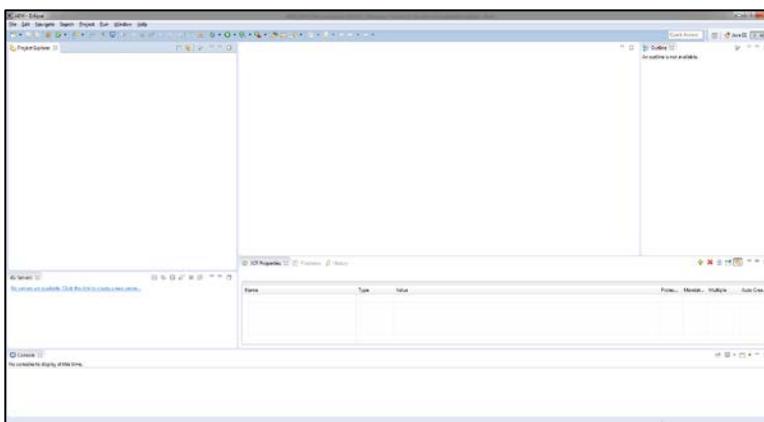
13. Click **Workbench** in the upper-right corner again. The **Workspace** opens.
14. Notice the new **AEM perspective** available in Eclipse. To check this, click the **Open Perspective** icon, as shown:



15. Notice how the **AEM** perspective is at the top, as shown:



16. Select **AEM**, and click **OK**. You will see a new perspective opened in Eclipse, as shown:



Note: Keep Eclipse open for the next exercise.

## Exercise 3: Import an existing Maven project

In this exercise, you will import an existing Maven project that is built using the Maven Archetypes 13. This exercise includes two tasks:

1. Create an AEM project
2. Resolve uber-jar dependencies



**NOTE:** Alternatively, you could use the AEM wizarding tool in Eclipse. The project we are importing below has no special features outside the standard Archetype 13. We are merely using a precreated project for time related purposes.

### Task 1: Import an AEM project

1. Extract the zip file named **training-starter-project.zip** from Exercise\_Files\05\_Dev\_Environment. The **training-starter-project** file gets extracted.
2. Double-click **training-starter-project** file to open it. You should see a **training** folder with several folders and files inside it, as shown:



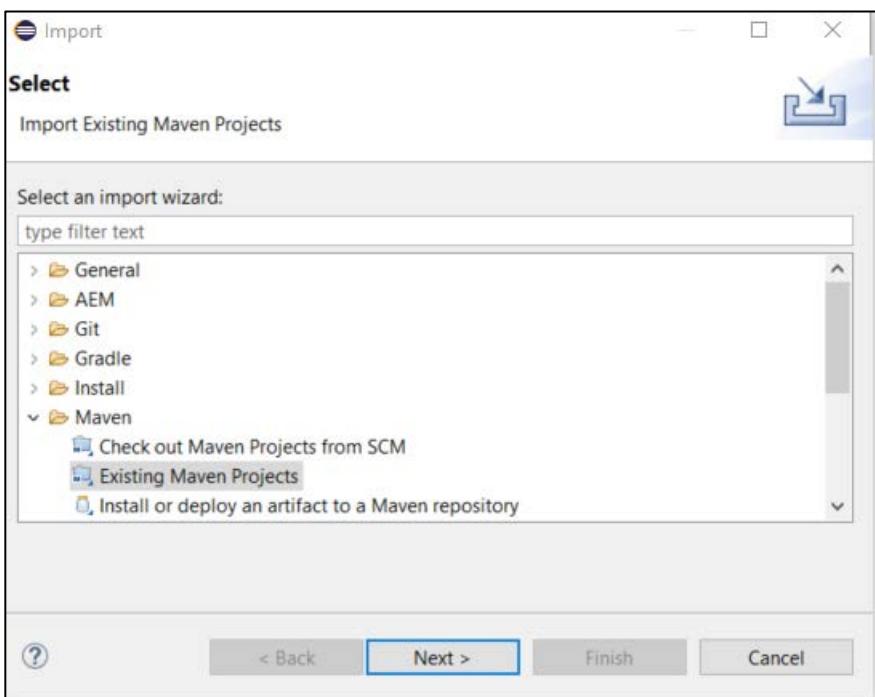
**Note:** Please ensure you have received the **Exercise\_Files.zip** file from your instructor, copy it to your ReadyTech instance local drive, and extract it. You will need this file to carry out the steps.

Exercise_Files > training	
Name	Date modified
.settings	4/5/2018 4:42 PM
core	4/5/2018 4:42 PM
it.launcher	4/5/2018 4:42 PM
it.tests	4/5/2018 4:42 PM
ui.apps	4/5/2018 4:42 PM
ui.content	4/5/2018 4:42 PM
.gitignore	4/5/2018 4:38 PM
.project	4/5/2018 4:38 PM
pom.xml	4/5/2018 4:38 PM
README.md	4/5/2018 4:38 PM

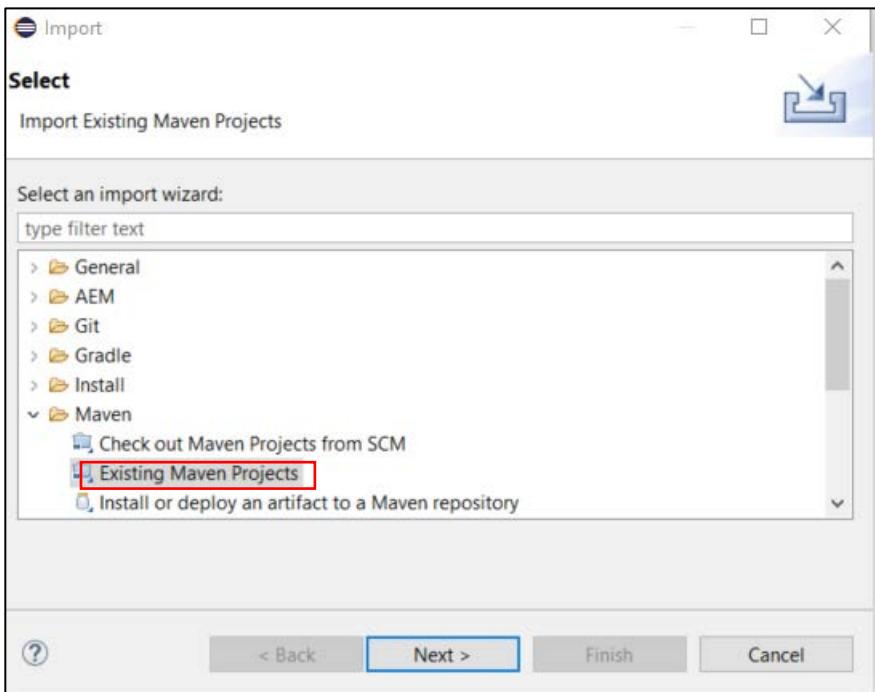
3. Right-click the **Eclipse Jee Oxygen** icon, and select **Open** from the list. The **Eclipse Launcher** opens.
4. Click **Launch**. The eclipse-workspace – Eclipse window opens.



5. Click **File > Import**. The **Import** window opens, as shown:

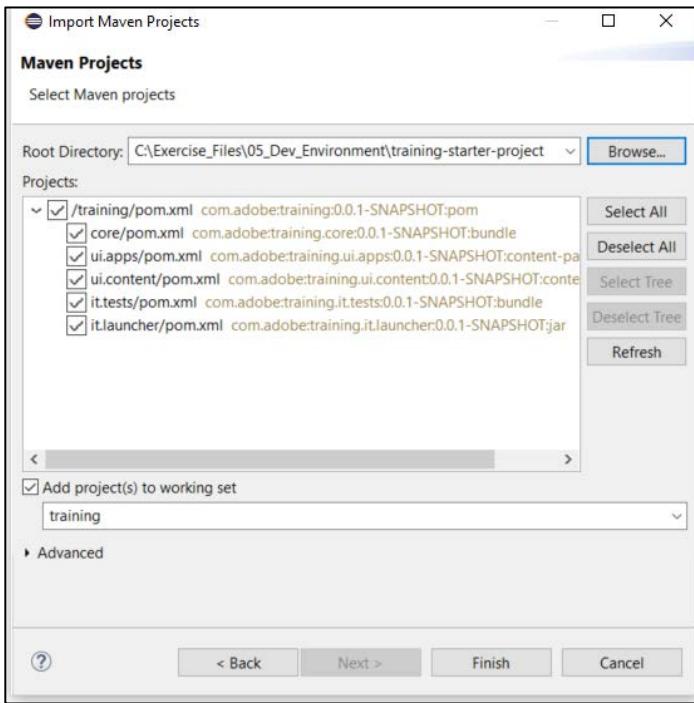


6. Click the small arrow symbol beside **Maven** to expand it.  
7. Select **Existing Maven Projects**, and click **Next**, as shown. The **Import Maven Projects** window opens.

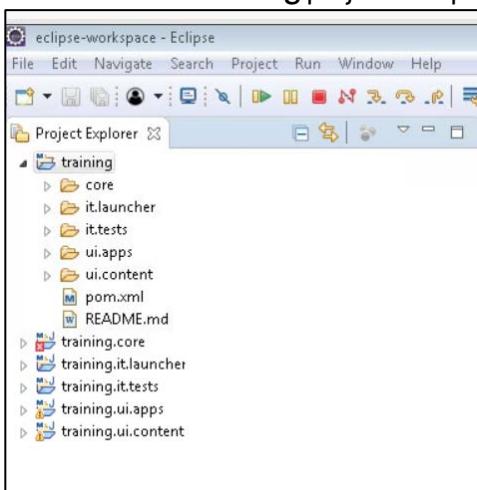


---

8. Click the **Browse** button beside the **Root Directory** field to browse to C:\Exercise\_Files\05\_Dev\_Environment\training-starter-project, and select all the POM files, as shown:



9. Ensure that the **Add Project(s) to working set** checkbox is selected.  
10. Click **Finish**. The **training** project is imported in the **Project Explorer**, as shown:



## Task 2: Update the POM files for the project

Now that you have a basic project from the archetype, you will be able to update the POM files to support AEM 6.4 and a few dependencies that you will use later in this course.

In the **Project Explorer**, find and replace the contents of:

1. **training > it.tests > pom.xml** with Exercise\_Files/05\_Dev\_Environment/POM\_Files/it-tests\_pom.xml, and save the changes.
2. **training.core** (parent POM) > **pom.xml** with Exercise\_Files/Files/05\_Dev\_Environment/POM\_Files/pom.xml, and save the changes.
3. **training > core > pom.xml** with Exercise\_Files/05\_Dev\_Environment/POM\_Files/core\_pom.xml, and save the changes.
4. **training > ui.apps > pom.xml** with Exercise\_Files/05\_Dev\_Environment/POM\_Files/ui-apps\_pom.xml, and save the changes.
5. **training > ui.content > pom.xml** with Exercise\_Files/05\_Dev\_Environment/POM\_Files/ui-content\_pom.xml, and save the changes.

Note: The updated POM files have the following changes to support our project:

- The uber-jar was updated from 6.3 to 6.4, which lets retrieve all the correct dependencies for AEM.
- A new dependency for core component development is added, as shown:

Parent POM:

```
<!-- Core Components 2.0 -->
<dependency>
    <groupId>com.adobe.cq</groupId>
    <artifactId>core.wcm.components.core</artifactId>
    <version>2.0.4</version>
    <scope>provided</scope>
</dependency>
<!-- ~~~~~
<!-- Core Components 2.0 -->
```



Core POM:

```
<!-- Core Components dependency -->
<!-- ~~~~~
<dependency>
<groupId>com.adobe.cq</groupId>
<artifactId>core.wcm.components.core</artifactId>
</dependency>
<!-- ~~~~~
<!-- Core Components dependency -->
```

- A new dependency for Google Gson for json objects is added, as shown:

Parent POM:

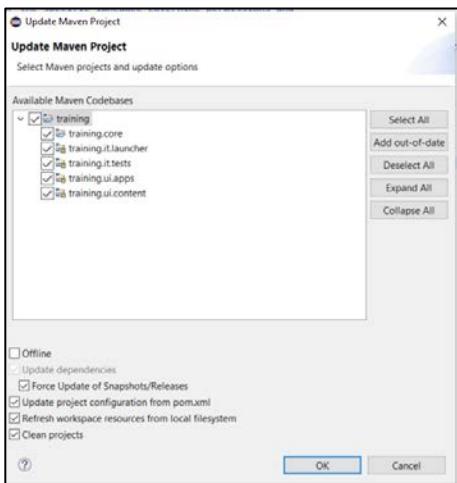
```
<!-- Dependency for Google Gson http://search.maven.org/#artifactdetails%7Ccom.google.code.gson%7Cgson%7C2.3.1%7C
<dependency>
<groupId>com.google.code.gson</groupId>
<artifactId>gson</artifactId>
<version>2.3.1</version>
<scope>provided</scope>
</dependency>
```

Core POM:

```
<!-- Dependency for Google Gson http://search.maven.org/#artifactdetails%7Ccom.google.code.gson%7Cgson%7C2.3.1%7C
<dependency>
<groupId>com.google.code.gson</groupId>
<artifactId>gson</artifactId>
</dependency>
```

6. Right-click **training**, and select **Maven > Update Project**. The **Update Maven Project** window opens.
7. Verify your codebase is selected as **training**.

8. Select the **Force Update of Snapshots/Releases** checkbox, and click **OK**, as shown. Your project is now updated.

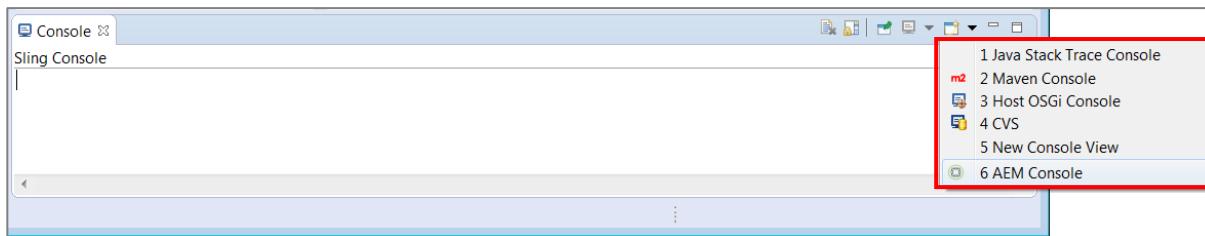


9. Ignore any errors. You will fix them later.

## Configuring the AEM Console

The AEM Console displays the progress of repository synchronization. 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, on the **Console** tab, click the **Open Console** icon and select **AEM Console**. If the **Console** tab is not visible, select **Window > Show View > Console**, click **Open Console**, and select **AEM Console**, as shown:

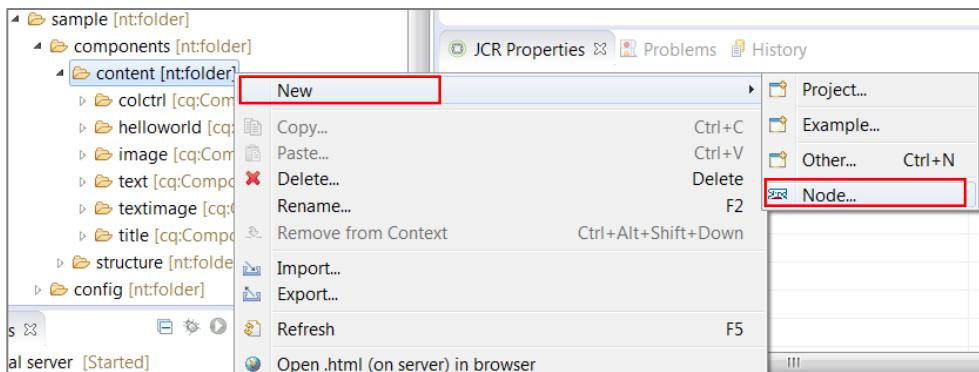


## Working with JCR

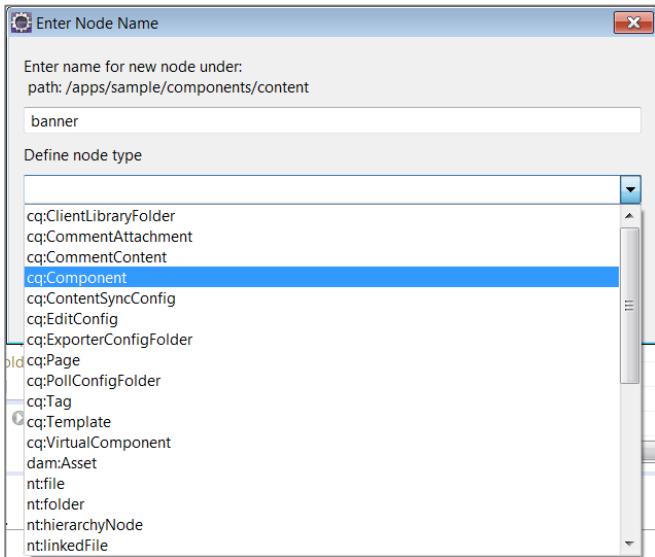
With the AEM perspective, you can add a new node to the JCR and add or edit properties of the node.

Adding a New Node:

1. Using Eclipse, create new JCR nodes in your project:



2. Give a name for the node, and select the type of node:



3. Click a node in Project Explorer to check its properties in the JCR Properties view, as shown:

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

#### Adding or Editing a Node Property:

1. To add a property, select the **Insert a property** icon from the toolbar.
2. Add the details of the property, and press Enter to save the changes.
3. Right-click the property, and select **Show** in editor to view the underlying vault file in the editor window.

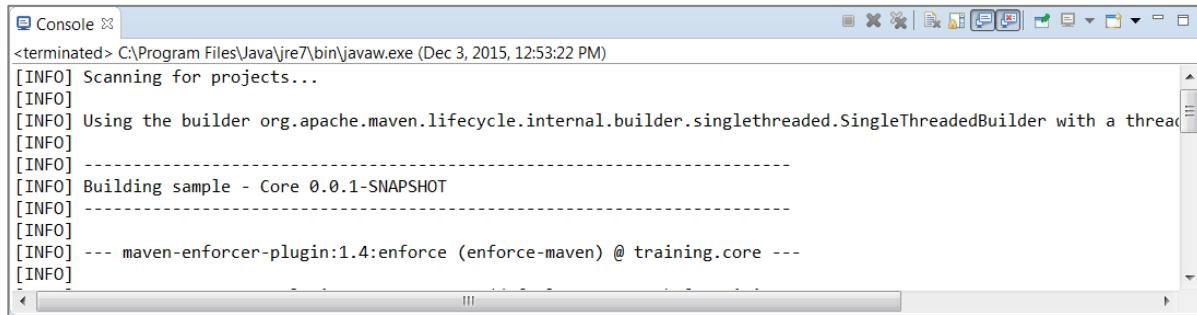
Name	Type	Value	Prote...	Mand...	Multiple	Auto Cr...
jcr:primaryType	Name	cq:Component	true	true	false	true
jcrt:title	String	Banner	false	false	false	false

+

## Building and Deploying a Project Using Maven

After you complete a part of your AEM project development, you can manually build and deploy your content package or OSGi bundle.

After you deploy your project, you can check the progress of the command in the **Console**, which is part of the Eclipse Workspace.



The screenshot shows the Eclipse IDE's Console view. The title bar says "Console". The main area 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 of size = 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]
```

An AEM project has the following modules:

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

After your project is set up, you can use the Export to Server and Import to Server options to synchronize content between Eclipse and the AEM server.

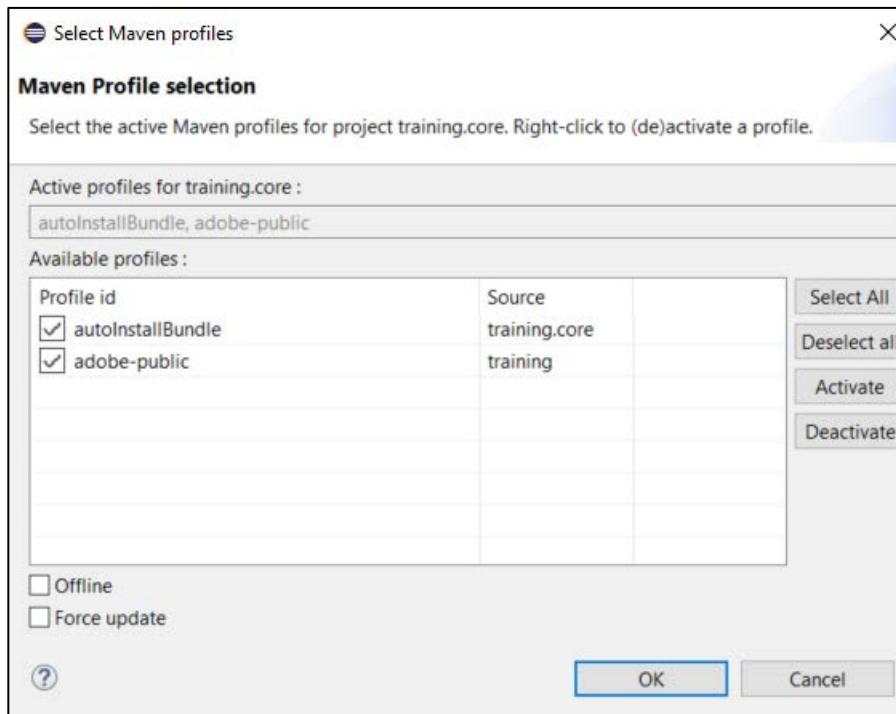
## Exercise 4: Build and deploy the project to AEM

In this exercise, you will build the project by setting up the Maven profile for each module and deploy the project to AEM. This exercise includes the following two tasks:

1. Build the AEM project
2. Deploy the project

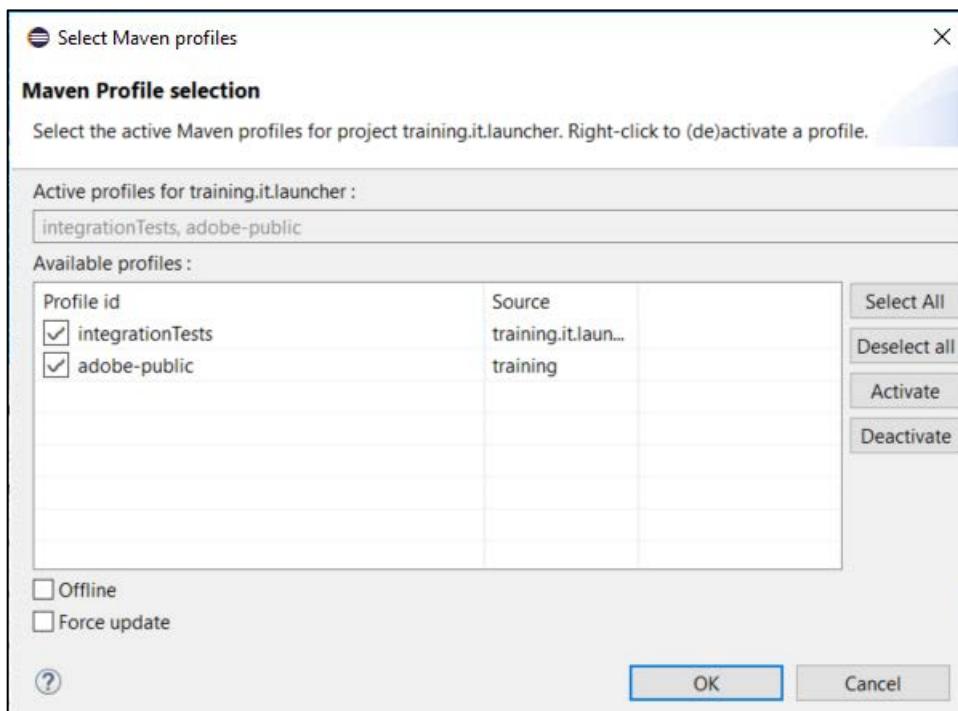
### Task 1: Build the AEM project

1. Right-click **training.core**, and select **Maven > Select Maven Profiles**. The Maven Profile selection window opens.
2. In the Select Maven profiles window, select **autoInstallBundle** and **adobe-public** (auto activated) from the Available profiles section, and click **OK**, as shown. The **autoInstallBundle** for **training.core** is activated.

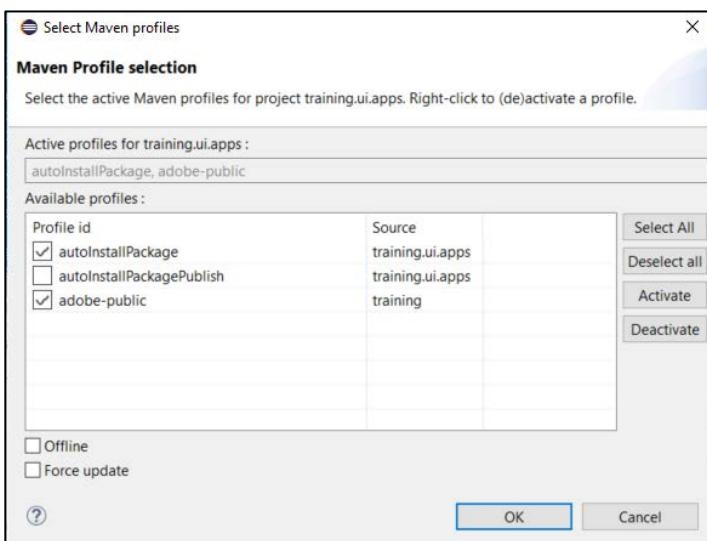


3. Right-click **training.it.launcher**, and select **Maven > Select Maven Profiles**. The Maven Profile Selection screen is displayed.

4. Select **integrationTests** and **adobe-public** (auto activated) from the **Available profiles** section, and click **OK**, as shown. The integration tests for **training.ui.launcher** is activated.



5. Right-click **training.ui.apps**, and click **Maven > Select Maven Profiles**. The Maven Profile Selection screen is displayed.  
 6. Select **autoInstallPackage** and **adobe-public** (auto activated) from the Available profiles section, and click **OK**, as shown. The **autoInstallPackage** for the **training.ui.apps** is activated.



+

7. Right-click training.ui.content, and select Maven > Select Maven Profiles. The Maven Profile Selection screen is displayed.
8. Select **autoInstallPackage** and **adobe-public** from the Available profiles section, and click OK. The **autoInstallPackage** for the training.ui.content is activated.

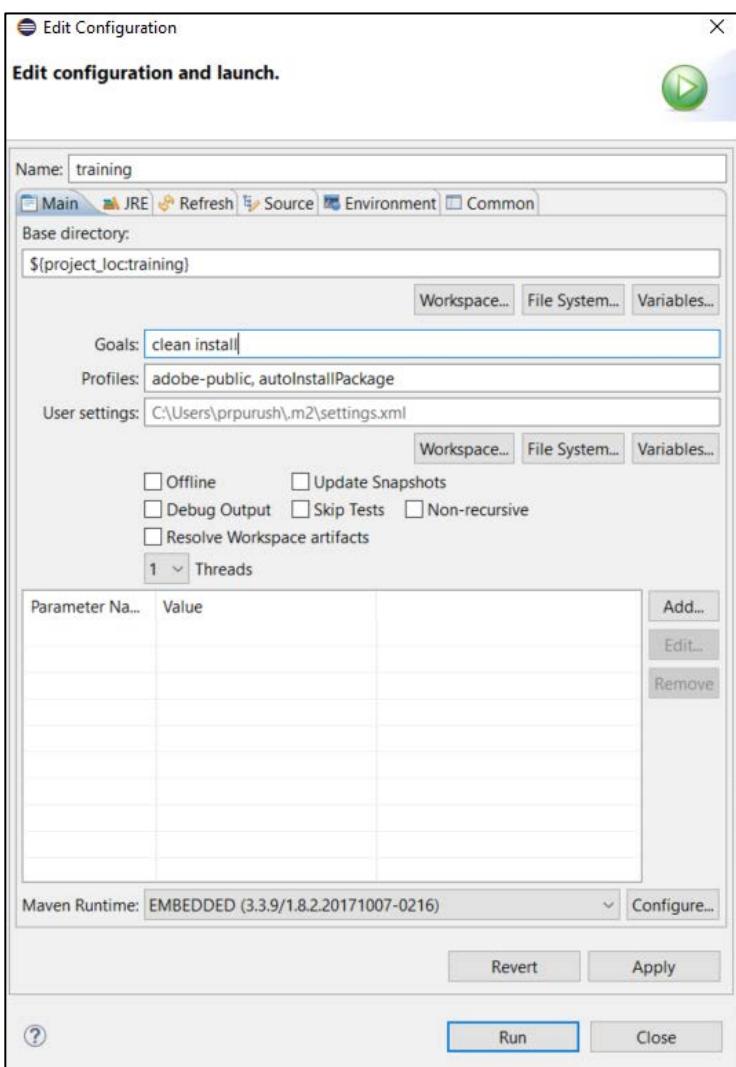
You have now set up your Maven profiles.



**Note:** If you see *No DS descriptor found at path target/classes/OSGI-INF/com.adobe.training.core...*, it is Eclipse looking for files that you do not need. You can configure Eclipse to ignore these messages in your own environment.

## Task 2: Deploy the project

1. Right-click **training** (parent directory), and select **Run As > 2 Maven build**. The **Edit configuration and launch** dialog box opens.
2. In the **Edit configuration and launch** dialog box, in the **Goals** field, enter **clean install** and in the **Profiles** field, enter: **adobe-public, autoInstallPackage**, as shown:



**Note:** You should run the **autoInstallPackage** profile to ensure you have the correct content pages built within the repository. If you run **autoInstallBundle** first, the build will fail to upload the bundle into Apache Felix.

- Click **Apply**, and then click **Run**. The Success message appears:

```
[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\adobe\it\training\0.0.1-SNAPSHOT\pom.xml
[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]
```

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

- Verify both **ui.apps** and **ui.content** content packages are installed in **CRXDE** lite and the core bundle installed in the Web Console.
- Log in to AEM with the **admin** credentials.
- Navigate to Adobe Experience Manager > Tools > CRXDE Lite.
- Click the **Package** icon. The installed packages are shown.



- Verify you see the installed packages, as shown:

	<b>training.ui.content-0.0.1-SNAPSHOT.zip</b> Version: 0.0.1-SNAPSHOT   Last installed 14:25   admin UI content package for TrainingProject	Share   763.8 KB
	<b>training.ui.apps-0.0.1-SNAPSHOT.zip</b> Version: 0.0.1-SNAPSHOT   Last installed 14:25   admin UI apps package for TrainingProject	Share   98.8 KB

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

Main	OSGi	Sling	Status	Web Console
Bundle information: 473 bundles in total - all 473 bundles active				
<input type="button" value="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)			

## Exercise 5: Configure the AEM Server in Eclipse

In this task, you will configure the AEM Server in Eclipse to enable content synchronization.

You can synchronize the code (Java, HTML, jsp, css, and js) in Eclipse workspace with the code in the JCR. For example, you have application logic in Eclipse that represents a HTL component, such as the Front-End Brackets tool. You can synchronize this code in Eclipse with the code in the AEM JCR. That is, you can import the code you write in Eclipse into the JCR.

If you make a change in the JCR by using CRXDE Lite, you can export these changes to the Eclipse workspace (filesystem):

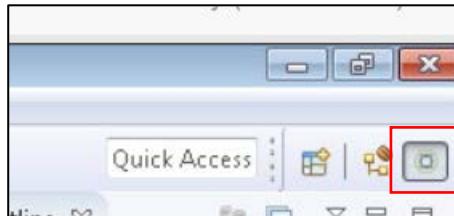
- Sync from Eclipse to the JCR is *Exporting to the server* (auto or manual).
- Sync from the JCR to Eclipse is *Importing from the Server* (manual only)

In this exercise, you will build the project by setting up the Maven profile for each module and deploy the project to AEM. This exercise includes two tasks:

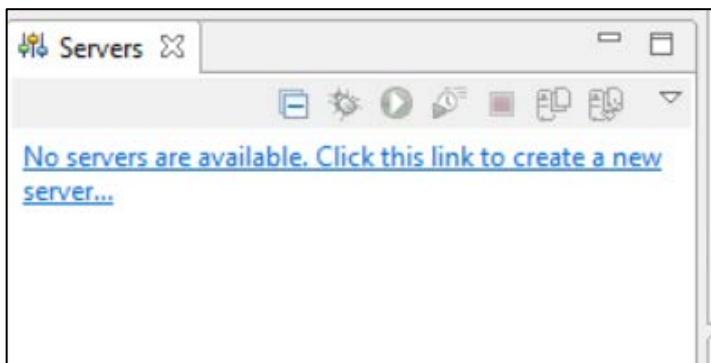
1. Configure the AEM Server
2. Export content from Eclipse

### Task 1: Configure the AEM Server

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

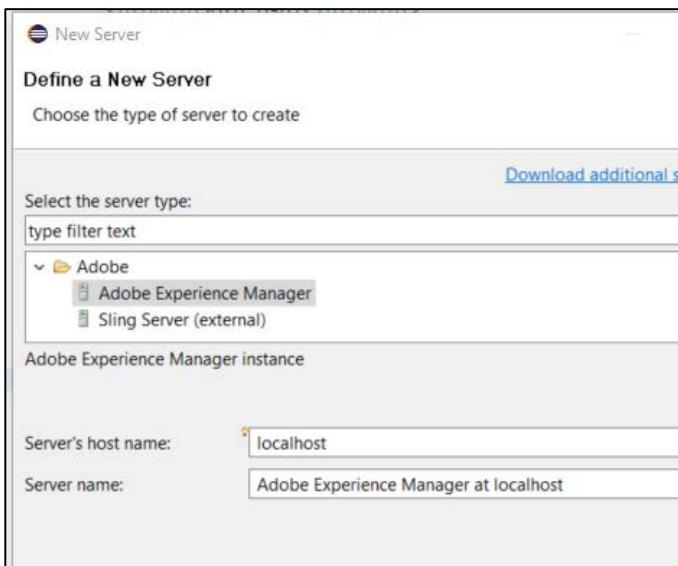


2. On the left-hand side of the Eclipse Workspace below Project Explorer, notice the **Servers** tab. Click **the No Servers are available.**



The **Define a New Server** dialog box opens.

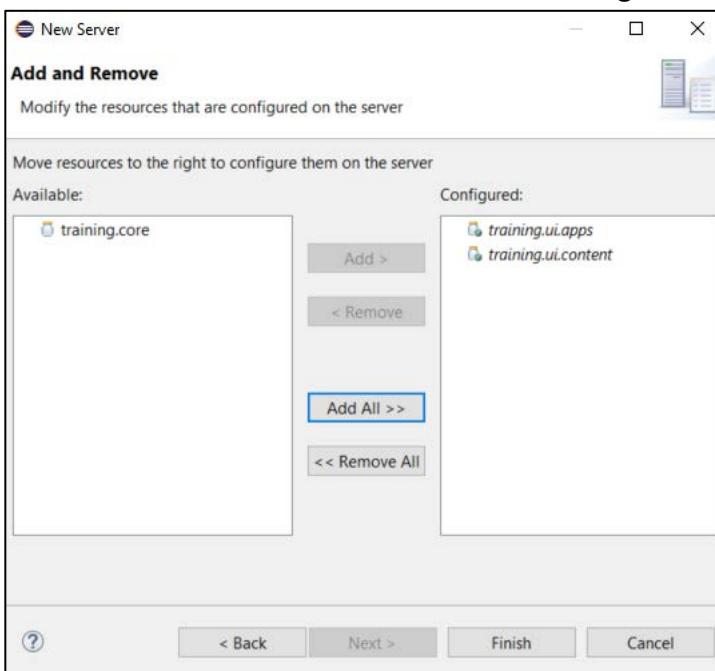
3. In the **Define a New Server** dialog box, select **Adobe > Adobe Experience Manager**, as shown:



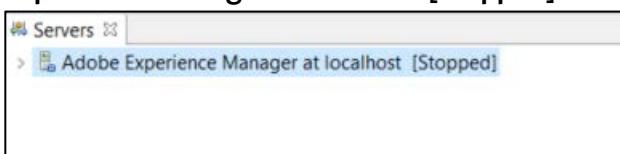
4. Accept the default settings for **Server's host name** and **Server name** fields.

5. Click **Next**. The **Add and Remove resources** dialog box opens.

6. Select **training.ui.apps** and **training.ui.content** one by one, and click the **Add** button to move the specified resources from the **Available** section to the **Configured** column, as shown:

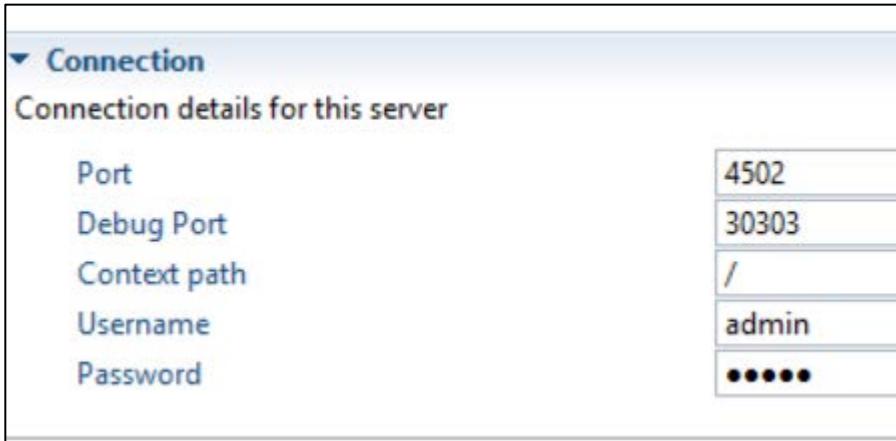


7. Click **Finish** on the **New Server** screen. The AEM Server is now defined.  
8. On the left-hand side of the workspace (below **Project Explorer**), select the **Servers** tab and note how **Adobe Experience Manager at localhost [Stopped]** is now available, as shown:



9. You can modify the configuration for the AEM Server. To do this, double-click the **Adobe Experience Manager at localhost [Stopped]** to open it in the editor.

10. In the **Connection** area, change **Port** to **4502**, as shown:



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

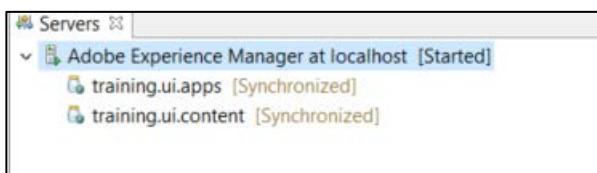
11. Save the changes (**File > Save**).



**Note:** You have now created a connection from Eclipse to the AEM server on your computer

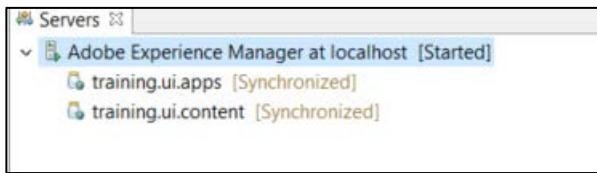
12. Right-click **Adobe Experience Manager at localhost** on the **Servers** tab, and click **Start**. The server is started.

13. Verify the Server has started, as shown:



The contents are now in sync with the AEM server.

14. Verify the Server has started, as shown:



The contents are now in sync with the AEM Server.

## Task 2: Export content from Eclipse

1. In Project Explorer, navigate to: **training.ui.apps > src / main /content/jcr\_root > apps> trainingproject > components > content > helloworld**
2. Double-click **helloworld.html**. The html page opens.
3. Add the following line to the comment part:

"This is a change in Eclipse"

```

4  LIMITATIONS UNDER THE LICENSE.
5  *!-->
6  <p data-sly-test="${properties.text}">Text property: ${properties.text}</p>
7
8@<pre data-sly-use.hello="com.adobe.training.core.models.HelloWorldModel">
9 HelloWorldModel says:
10 ${hello.message}
11 "This is a change in Eclipse"
12 </pre>

```

4. Save the changes.



**Note:** When you save **helloworld.html**, the AEM Server connection in Eclipse exports **helloworld.html** to the JCR.

5. In CRXDE lite, browse to **apps > trainingproject > components > content > helloworld > helloworld.html** and double-click **helloworld.html**. The html page opens.
6. Verify the change in **CRXDE Lite**:

The screenshot shows a browser window for 'helloworld.html'. The title bar says 'helloworld.html'. The content area displays the following code:

```

18 <pre data-sly-use.hello="com.adobe.training.core.models.HelloWorldModel">
19 HelloWorldModel says:
20 ${hello.message}
21 "This is a change in Eclipse"
22 </pre>
23

```

7. To import content from the JCR to Eclipse (manually), go to **CRXDE Lite** and navigate to **apps > trainingproject > components > content > helloworld**.
8. Right-click the page node, and select **Create > Create File**.
9. Enter the name as **Test.html**.
10. Click **OK**. The page gets created. Save the changes.



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

11. In Eclipse, under **ui.apps**, select **/apps/trainingproject/components/content/ helloworld**, right-click and select **Import > Import from Server**.
12. Accept the default settings, and click **Finish**. The project is imported from the server.
13. Under **training.ui.apps**, navigate to **/apps/trainingproject/components/content/ helloworld** and verify **Test.html** appears in your project, as shown:



**Note:** Step 11 is a very typical process in AEM Java development to sync new content from the JCR to Eclipse. Remember that Eclipse is our master repository 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.



**IMPORTANT!** If you create something in CRXDE Lite that you want to keep, you must sync it back to Eclipse using the process above.

## References

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



# OSGi Configurations and Run Modes

## Introduction

OSGi configurations are used by developers and administrators to manage Adobe Experience Manager (AEM) instance settings in conjunction with run modes. This course will teach you the use cases and best practices for creating custom OSGi configurations and run modes.

## Objectives

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

- Explain different methods of creating OSGi configurations
- Set run modes in the command line when starting or installing AEM
- Explain how OSGi configurations are organized to match run modes
- Create custom OSGi configurations using Java Content Repository (JCR) nodes
- Start AEM with a custom run mode
- Create a custom OSGi configuration node for a custom run mode

## OSGi Configurations

OSGi is a fundamental element in the technology stack of AEM and supports modular deployment of bundles. You can stop, install, and start these bundles individually. The interdependencies are handled automatically. Each OSGi component is contained in one of the deployed bundles, which helps manage applications easily.

OSGi configurations are system parameters and settings contained in bundles that you can manage and configure.

## Methods to Create OSGi Configurations

You can use three methods to create OSGi configurations:

1. Creating configuration nodes in the JCR (**Preferred Method**)
2. Using the Web Console
3. Using configuration files



**Note:** The preferred method as per AEM best practices is **creating configuration nodes in the JCR**.

---

### Creating Configuration Nodes in the JCR

You can define configurations for each run mode by creating specific nodes by following these guidelines:

- Create nodes under `/apps` of type `sling:OsgiConfig`
- Organize folders for your configuration nodes by run mode

Using this method, it is possible to share a configuration among several instances, which is a best practice in AEM development.

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 before creating configurations:

- 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, create a configuration (**config**) for all run modes, such as `config.author` for the author environment and `config.publish` for the publish environment.
- Requirements of configurations or factory configurations

- Parameters—Configure the individual parameters, 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.

## Using the Web Console

The Web Console (<http://localhost:4502/system/console/configMgr>) is a built-in AEM administrative console used to display and manage OSGi configurations. You can configure all bundles through the Configuration tab, which means you can use it to configure AEM system parameters.

While configurations can be directly and rapidly edited in the Web Console, the configurations made with this console are applied immediately, require no restart, and apply to the current instance, regardless of the current run mode, or any subsequent changes to the run mode.

It is recommended *not* to use this method as the changes made to an instance:

- Persist only on that instance
- Are not attributable to a user
- Are not associated with a run mode
- Are not tied to a change control system



**Note:** Use this method only for temporary testing in a development environment.

---

## Using Configuration Files

Configuration files are files ending with the extension **\*.config** in the /launchpad directory of your AEM crx-quickstart directory structure. Although these files can be used to manage OSGi configurations, it is recommended that you never edit these files to manage OSGi configurations. This is a risky process and could lead to system instability. If configuration files are accessed, they should be kept as view-only (or read-only) and only be used for troubleshooting purposes.

## 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`.
- The default settings in AEM are saved in `*.config` files under `/crx-quickstart/launchpad/config`.



**Warning:** You must never edit the folders or files under the `/crx-quickstart/launchpad/config` folder.

---

## Adding a New Configuration to the Repository

When creating configuration nodes, you must ensure their names are equal to the Persistent Identity (PID) of the configuration in the Web Console. For example, you can navigate to the Web Console 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 the `config` followed with a period. All the run modes that the config applies to are also separated with a period, such as `config.author`, `config.publish`, `config.author.dev` and `config.author.foo.dev`.

To create a configuration 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 following folders based on the run mode:
  - `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.

- Whether the configuration already exists 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 Order at Startup

The following order of precedence is used when resolving configuration nodes upon AEM startup:

1. Repository nodes under `/apps/*/config`, either with type `sling:OsgiConfig` or property files
2. Repository nodes with type `sling:OsgiConfig` under `/libs/*/config`
3. 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 AEM 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 and syntax:

```
/apps/<your_application>/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.

## Run Modes

### Introduction to Run Modes

You can run your AEM 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.

### Custom Run Modes

AEM 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 AEM 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 run mode 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, such as author, foo, and dev. These run modes can be set as Java Virtual Machine (JVM) options. For example, to set run modes from the command line:

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

Alternatively, in the start script (or batch file used to start AEM):

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

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

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

---

 **Note:** It is recommended that as a best practice, you define run modes using the command line, or by editing the start scripts located in /crx-quickstart/bin/start. It is not recommended to use the sling.properties file to define run modes.

---

## Configurations per Run Mode

To create separate configuration settings per run mode in the JCR, create folder names using the syntax of config.<runmode> for each run mode used. For example, configurations that apply to the publish run mode would be set in this folder: /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 are:

- 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
- 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)

## Exercise 1: Create custom OSGi configurations and start AEM with a custom run mode

**Scenario:** As a developer or system administrator, you need to:

- Follow the best practices by creating OSGi configurations in the JCR using /apps
- Be able to start AEM with a custom run mode using the command line
- Create multiple sets of custom OSGi configurations to match custom run modes in order to specify different groups of settings for each instance or environment you are deploying

In a typical development process, there are multiple servers designated for specific tasks in your infrastructure. Each server will have different configurations depending on location, type, task, and purpose. You need to use OSGi configurations and Run Modes to manage these configurations.

### Task 1.1: Create a custom OSGi configuration node

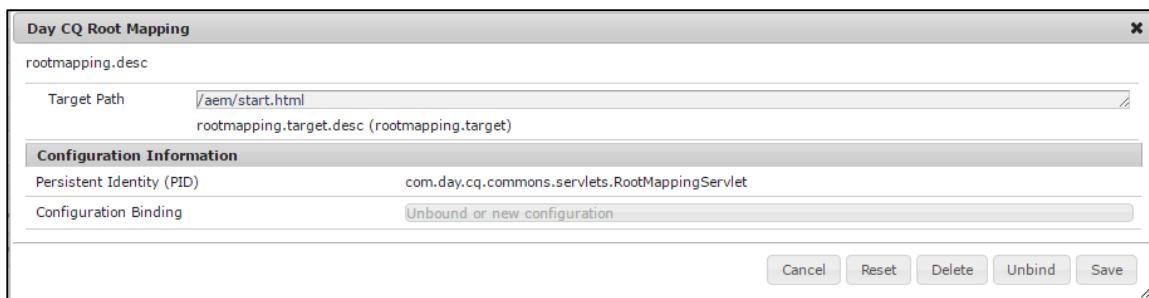
In this task, you will create a custom OSGi configuration node to change the default “root” start page (<http://localhost:4502>) that loads when you sign in to AEM from **start.html** to **sites.html**.

1. Ensure your AEM author instance is running and you are logged on.
2. Navigate to the **Web Console** at <http://localhost:4502/system/console> (or within AEM, use **Tools > Operations > Web Console**).
3. Select OSGi > Configuration.

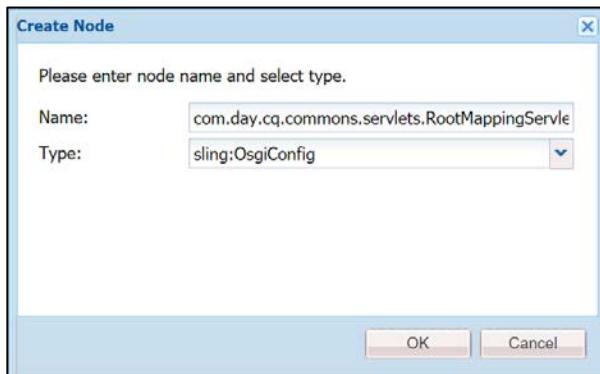
Main	OSGi	Sling	Status	Web Console
Bundle	Bundles	... in total - all 544 bundles		
	Components			
	Configuration	Apply Filter	Filt	
Id	Events			
0	Log Service	org.apache.felix.frame		
165	OSGi Installer	rg.apache.abdera.clien		
	Dashboards Dependencies			

4. To locate the **Root Mapping** OSGi configuration, search for the words: **Day CQ Root Mapping** in your browser.

- Click this configuration to open it in a dialog window:



- Copy the value for **Persistent Identity (PID)**. You will create a configuration node for this configuration in CRDXE Lite using this value under apps as per the best practice to change OSGi configurations. Keep this Web Console page open for reference.
- In a new browser tab, open CRXDE Lite by navigating to <http://localhost:4502/crx/de> (or within AEM, use **Tools > CRXDE Lite**).
- Navigate to **apps > weretail**.
- Right-click the **weretail** folder and create a new folder node by selecting **Create > Create Folder**.
- Enter **config.author** in the **Name** field.
- Click **OK**.
- Click **Save All** in the top-right corner to ensure your new folder is saved.
- Right-click the **config.author** folder node and create a new node by selecting **Create > Create Node**.
- Specify the following details:



- Paste the following **PID** you copied from the Web Console in the **Name** field:  
**com.day.cq.commons.servlets.RootMappingServlet**

- Select the Type as: **sling:OsgiConfig**

---

**Tip:** Ensure there are no whitespaces or bullet points preceding the **Name** value. These may be included in the value you copied previously from the Web Console, so if they do get pasted in, remove them.

---

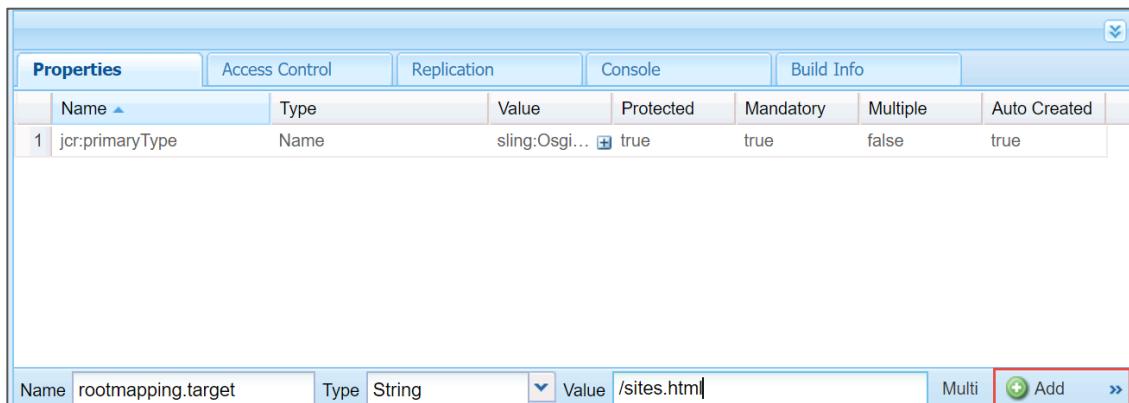
15. Click **OK**, and then click **Save All** in CRXDE Lite to ensure your new node is saved.
16. With the **com.day.cq.commons.servlets.RootMappingServlet** node selected, navigate to the JCR Properties tab of CRXDE Lite and add the following:
  - Name: **rootmapping.target**
  - Type: **String**
  - Value: **/sites.html**

---

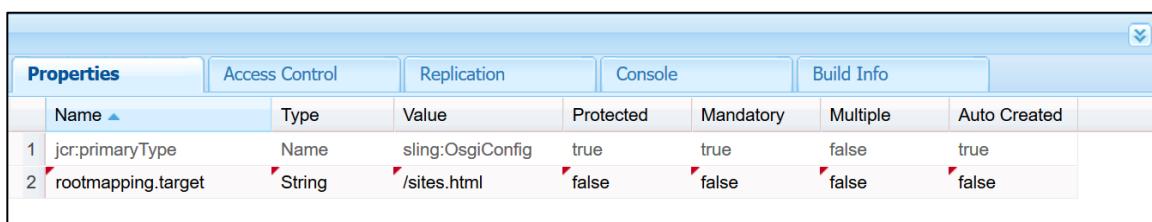
**Tip:** The property value of **rootmapping.target** can be obtained from the OSGI configuration in the Web Console. Property names are always in parentheses at the end of the description of each field. Refer to the screen shot in Step #5 to see how this matches.

---

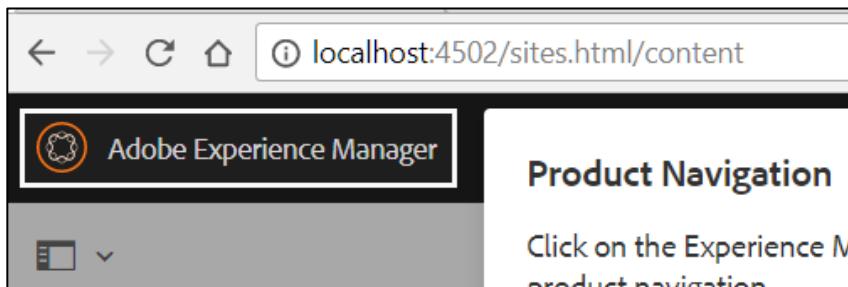
17. Click **Add** in the bottom-right corner in the JCR Properties tab to add your property.



The property is added to the **Properties** tab as shown:



18. Click **Save All** in CRXDE Lite to ensure your property is saved.
19. Open a new browser tab and navigate to the AEM author instance (<http://localhost:4502>).  
The **Sites** page should open, as this is what you configured with your custom configuration node.



---

 **Note:** AEM picks up this configuration because you placed it under `config.author` and the instance is using the author run mode currently. In Task 1.2, you will use a tool to check the current run modes of an instance.

---

## Task 1.2: Start AEM with a custom run mode

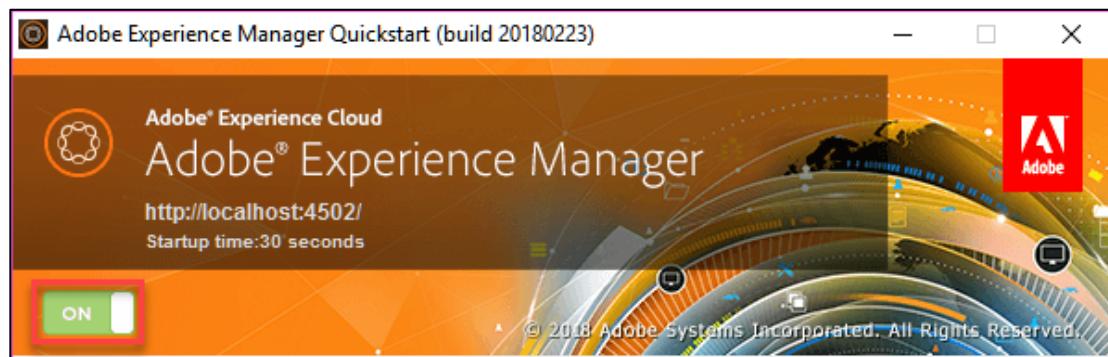
In this task, you will start your AEM author instance using a custom run mode called dev via command-line.

1. In the AEM Web Console (<http://localhost:4502/system/console>), navigate to **Status > Sling Settings**.
2. Observe the current run modes your author instance is using (**s7connect, crx3, author, samplecontent, crx3tar**):

The screenshot shows the 'Sling Settings' page of the AEM Web Console. At the top, there's a navigation bar with links for Main, OSGI, Sling, Status, and Web Console. Below the navigation is a timestamp: 'Date: March 14, 2018 2:19:24 PM PDT'. Under the 'Apache Sling Settings' section, there are several configuration parameters. One of these parameters is 'Run Modes', which is listed as '[s7connect, crx3, author, samplecontent, crx3tar]'. This entire line of text is highlighted with a red rectangular box.

3. You will now add another run mode and use this tool at the end of this task to verify your custom run mode.

4. To shut down your running AEM author instance, you can click the click the **ON / OFF** toggle button in the GUI window:



- 5.
6. If you are running AEM using the command line, use **CTRL+C** (in Windows) in your command window to shut down AEM.
7. Now, you will start AEM using the command line in order to use a custom run mode. This is because the custom run modes can be generated using command line parameters. Navigate to the directory on your machine where your author instance quickstart file resides.



**Tip:** If you are using a ReadyTech instance, this directory should be C:\adobe\AEM\author.

8. Start AEM again using the following command that specifies the author and dev run modes:
9. `java -jar aem-author-4502.jar -r author,dev -gui`
10. Your AEM instance should start up again, this time with a command window available to view details of the startup. In addition, the GUI window will also be available.



**Tip:** Be patient as it may take up to two minutes for your instance to start.

11. Sign in to AEM again.
12. In the AEM Web Console (<http://localhost:4502/system/console>), navigate to **Status > Sling Settings** (or, refresh that page in your browser tab, as you may already have it open). Your custom run mode should now appear:

localhost:4502/system/console/status-slingsettings

## Adobe Experience Manager Web Console

### Sling Settings

Main OSGI Sling Status Web Console

Date: March 14, 2018 2:56:35 PM PDT

Apache Sling Settings

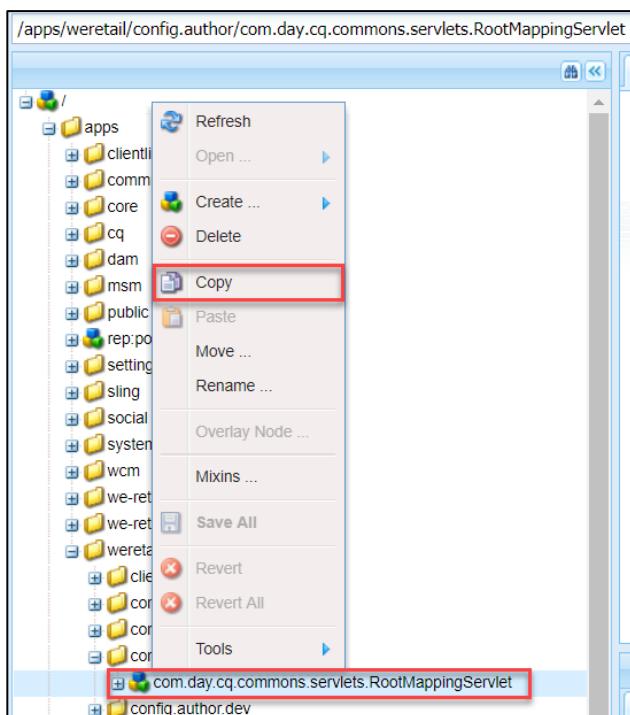
```
Sling ID = 103b79d7-555a-4d9a-98e3-7d7fd5b62454
Sling Name = Instance 103b79d7-555a-4d9a-98e3-7d7fd5b62454
Sling Description = Instance with id 103b79d7-555a-4d9a-98e3-7d7fd5b62454 and run modes [dev, s7connect, crx3, author, samplecontent, crx3tar]
Sling Home = C:\AEM\6.4\Load 21 -RCS\crx-quickstart
Sling Home URL = file:/C:/AEM/6.4/Load%2021%20-RCS/crx-quickstart/
Run Modes = [dev, s7connect, crx3, author, samplecontent, crx3tar]
```

## Task 1.3: Create a custom OSGi configuration node for a custom run mode

In this task, you will change the default “root” page to load CRXDE Lite for the custom ‘dev’ run mode.

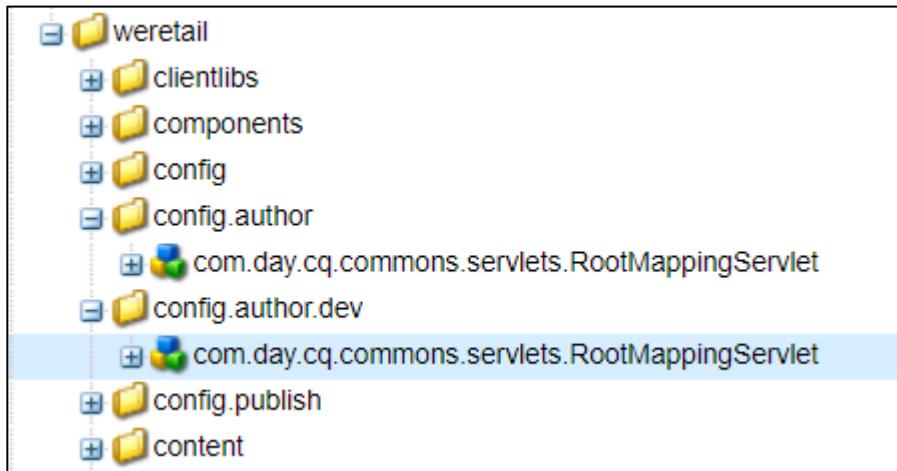
To create another root mapping OSGi configuration node:

1. Open **CRXDE Lite** using <http://localhost:4502/crx/de> or from AEM by navigating to **Tools > CRXDE Lite**.
2. Navigate to **apps > weretail**.
3. Right-click the **weretail** folder to create another folder node.
4. Select **Create > Create Folder**.
5. Enter **config.author.dev** in the **Name** field:
6. Click **OK**, and click **Save All** in CRXDE Lite to ensure your new folder is saved.
7. Now, instead of manually creating another root mapping node that is very similar to the one you created in Task 1.1, you can copy the node you created and paste it into your new folder and make the necessary changes.
8. Copy the **com.day.cq.commons.servlets.RootMappingServlet** node from the **config.author** folder by right-clicking the node and selecting **Copy**.



9. Right-click the **config.author.dev** folder and select **Paste**.

10. Click Save All.
11. Your node structure should now look like this:



12. Now, to change the properties of your copied node (config.author.dev), double-click the **Value** field for the rootmapping.target property in the **Properties** tab. This will make the field editable.
13. Enter /crx/de/index.jsp in the Value field.

Properties		Access Control	Replication	Console
	Name	Type	Value	
1	jcr:created	Date	2017-04-05T22:40:10.559+05:30	
2	jcr:createdBy	String	admin	
3	jcr:primaryType	Name	sling:OsgiConfig	
4	rootmapping.target	String	/crx/de/index.jsp	

14. Click **Save All** in **CRXDE Lite** to ensure your property is saved.
15. Now, shut down your AEM author instance again. Refer to Step 3 of Task 1.2 if you need instructions to shut down an AEM instance.
16. Start AEM again using this command that specifies the author and dev run modes:
17. `java -jar aem-author-4502.jar -r author,dev -gui`
18. To verify if the change was made to the dev run mode, when <http://localhost:4502> opens in your browser and you sign in, it should automatically navigate you to the CRXDE Lite page.
19. Question: Why does CRXDE Lite now load, but not the Sites page? What command could you use to start AEM with the Sites page?

## Answers to Questions

**Question:** Why does CRXDE Lite now load, but not the Sites page? What command could you use to start AEM with the Sites page?

**Answer:** CRXDE Lite loads because the root mapping property folder **config.author.dev** matches the dev custom run mode you supplied in the command when starting AEM. The Sites page matches the author run mode, but in this case, the dev run mode takes precedence when the same PID is used in two or more configuration folder nodes. This is because the configuration matching the highest number of run modes is used. The command you could use to start AEM with the Sites page is:

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

## References

For further information on OSGi Configurations and Run Modes, visit:

- Configuring Run Modes: <https://helpx.adobe.com/experience-manager/6-4/sites/deploying/using/configure-runmodes.html>
- Configuration Nodes: <https://helpx.adobe.com/experience-manager/6-4/sites/deploying/using/configuring-osgi.html>
- Lists of Common Configurations: <https://helpx.adobe.com/experience-manager/6-4/sites/deploying/using/osgi-configuration-settings.html>

# Configuring Custom Loggers in AEM



## Introduction

Adobe Experience Manager (AEM) log files provide detailed information about the current system state of AEM. It is important to maintain the AEM system, so that the system runs without any issues. Log files enable you to debug some common issues that you encounter in AEM. In addition to the default system log files, you can create and customize your own log files. Your customized log files help you better track messages produced by your own applications and separate them from the default log entries.

## Objectives

After completing this course, you will be able to:

- Explain the Logging system in AEM
- Explain Loggers and Writers in AEM
- Configure custom Loggers in AEM

# AEM Logging System

In AEM, you can configure the following:

- Global parameters for the central logging service
- Request data logging for request information
- Specific settings for the individual services, such as an individual log file and format for the log messages

The user login process in AEM is based on Apache Sling. The root logger defines the global settings for the logging system in AEM. The root logger is configured by using Apache Sling Logging Configuration. The org.apache.sling.commons.log bundle manages the logging system. This bundle helps:

- Implement the OSGi log service specification and register the LogService and LogReader services
- Export four commonly used APIs:
  - Apache Commons Logging
  - Simple Logging Façade for Java (SLF4J)
  - Log4j
  - Java.util.logging
- Configure the logging system by using Logback, which is integrated with the OSGi environment
- Configure the logging system by using both Logback xml or OSGi

In AEM, you can configure:

- Global logging. This defines the global settings for logging in AEM:
  - Logging level
  - Central log file location
  - Number of versions saved
  - Version rotation - either maximum size or a time interval
  - Format used when writing log messages



## Types of Log Files in AEM

The types of log files in AEM are:

- Access.log: Registers all access requests sent to AEM and the repository
- Audit.log: Registers all modern 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
- Upgrade.log: Provides a log of all upgrade operations

These files are available in the installation directory /crx-quickstart/logs.

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 AEM, 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 (Configured via OSGi Config)' and another for 'Appender'. The 'Logger' table lists various log entries with columns for Log Level, Additive, Log File, Logger, and Configuration. The 'Appender' table lists various appenders with columns for Appender, Configuration, and a link to edit.

Logger (Configured via OSGi Config)				
Log Level	Additive	Log File	Logger	Configuration
INFO	false	logs\upgrade.log	com.day.cq.compat.codeupgrade com.adobe.cq.upgrades com.adobe.cq.upgradesexecutor	<a href="#">Edit</a>
INFO	false	logs\audit.log	org.apache.jackrabbit.core.audit org.apache.jackrabbit.oak.audit	<a href="#">Edit</a>
INFO	false	logs\project-trainingproject.log	com.adobe.training	<a href="#">Edit</a>
DEBUG	false	logs\LoginTrace.log	org.apache.sling.auth.core.impl.SlingAuthenticator	<a href="#">Edit</a>
INFO	false	logs\project-we-retail.log	we.retail	<a href="#">Edit</a>
INFO	false	logs\request.log	log.request	<a href="#">Edit</a>
INFO	false	logs\auditlog.log	com.adobe.granite.audit	<a href="#">Edit</a>
INFO	false	logs\access.log	log.access	<a href="#">Edit</a>
INFO	false	logs\error.log	ROOT	<a href="#">Edit</a>
ERROR	false	logs\error.log	org.apache.sling.scripting.sightly.js.impl.jsapi.ProxyAsyncScriptableFactory	<a href="#">Edit</a>
INFO	false	logs\history.log	log.history	<a href="#">Edit</a>

Appender	
Appender	Configuration
File : [/logs/project-we-retail.log] C:\adobe\AEM\author\crx-quickstart\logs\project-we-retail.log	<a href="#">Edit</a>
File : [/logs/history.log] C:\adobe\AEM\author\crx-quickstart\logs\history.log	<a href="#">Edit</a>
File : [/logs/error.log] C:\adobe\AEM\author\crx-quickstart\logs\error.log	<a href="#">Edit</a>
File : [/logs/auditlog.log] C:\adobe\AEM\author\crx-quickstart\logs\auditlog.log	<a href="#">Edit</a>
File : [/logs/audit.log] C:\adobe\AEM\author\crx-quickstart\logs\audit.log	<a href="#">Edit</a>
File : [/logs/request.log] C:\adobe\AEM\author\crx-quickstart\logs\request.log	<a href="#">Edit</a>
File : [/logs/access.log] C:\adobe\AEM\author\crx-quickstart\logs\access.log	<a href="#">Edit</a>
File : [/logs/LoginTrace.log] C:\adobe\AEM\author\crx-quickstart\logs>LoginTrace.log	<a href="#">Edit</a>
File : [/logs/project-trainingproject.log] C:\adobe\AEM\author\crx-quickstart\logs\project-trainingproject.log	<a href="#">Edit</a>
File : [/logs/upgrade.log] C:\adobe\AEM\author\crx-quickstart\logs\upgrade.log	<a href="#">Edit</a>



## Loggers and Writers

The logging system in AEM consists of two elements, a Logging Logger and a Logging Writer. The Logging Logger collects data from different components inside AEM, filters them by requested severity level, and redirects the output to a configured Logging Writer. The Logging Writer persists the data provided by the logger to the physical file. For example, the error.log file is created by the Logging Writer on a rotational basis.

### Loggers and Writers for an individual service

In addition to the global settings, AEM allows you to configure specific settings for an individual service:

- 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 log messages
- Logger -the OSGi service supplies log messages

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

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

Logging Logger and Logging Writer 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 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

The following table lists the standard Writers and Loggers available in AEM:

Logger	Links to – Logger/Writer
Apache Sling Customizable Request Data Logger ( <code>org.apache.sling.engine.impl.log.RequestLoggerService</code> ) Writes messages about requests to the <code>request.log</code> file	Apache Sling Request Logger ( <code>org.apache.sling.engine.impl.log.RequestLogger</code> ) Write messages to either <code>request.log</code> or <code>access.log</code>
Apache Sling Logging Logger configuration ( <code>org.apache.sling.commons.log.LogManager.factory.config</code> ) Writes information messages to <code>logs/error.log</code>	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 <code>ogr.apache.pdfbox</code>	Does not link to any specific Writer. It creates and uses an implicit writer with default configuration.

In the above table, the first logger links to another logger, and the second logger links to the writer. The third logger does not link to a specific writer, so it creates and uses an implicit writer with default configuration (daily log rotation).



## Creating Your Own Loggers and Writers

To define your Logger/Writer pair:

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 other parameters as required.

## Exercise 1: Observe project logger config node

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

1. Open the AEM instance folder (**C:\adobe\AEM\author**), as shown:

	Name	Date modified	Type
...	crx-quickstart	3/13/2018 2:54 PM	File folder
...	aem-author-4502.jar	2/12/2018 4:01 PM	JAR File
...	license.properties	1/11/2018 9:32 AM	PROPERTIES File

2. Navigate to **crx-quickstart\logs\**. The list of log files is shown:

	Name	Date modified	Type
This PC	access.log	1/26/2017 10:50 AM	Text Document
Apple iPhone	request.log	1/26/2017 10:50 AM	Text Document
Desktop	error.log	1/26/2017 10:21 AM	Text Document
Documents	project-trainingproject.log	1/26/2017 10:05 AM	Text Document



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

3. Launch **Eclipse** by double-clicking the shortcut for Eclipse on your desktop, and open your workspace. The trainingProject opens in the Eclipse Development Environment.

4. In Project Explorer, navigate to training.ui.apps > src/main/content/jcr\_root [nt:folder] > apps [nt:folder] > trainingproject [nt:folder]/config [nt:folder]. You will see the file org.apache.sling.commons.log.LogManager.factory.config-trainingproject, as shown:



5. Double-click the file **org.apache.sling.commons.log.LogManager.factory.config-trainingproject**. The file opens in the Eclipse text editor, as shown:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0"
3   xmlns:jcr="http://www.jcp.org/jcr/1.0" jcr:primaryType="sling:OsgiConfig"
4     org.apache.sling.commons.log.file="logs/project-trainingproject.log"
5     org.apache.sling.commons.log.level="info"
6     org.apache.sling.commons.log.names="[com.adobe.training]"
7     org.apache.sling.commons.log.pattern="\{0,date,yyyy-MM-dd HH:mm:ss.SSS\} \{4\} [\{3\}] \{5\}" />
8

```

6. Click the **JCR Properties** tab below the file. The **Properties** window opens.

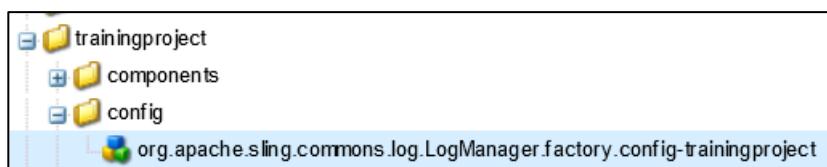
7. Observe the **JCR properties** for the log, as shown:

/apps/trainingproject/config/org.apache.sling.commons.log.LogManager.factory.config-trainingproject			
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	\{0,date,yyyy-MM-dd HH:mm:ss.SSS\} \{4\} [\{3\}] \{5\}	fa

8. Log on to AEM with the **admin** credentials. You are now logged on to the server.

9. Click **Adobe Experience Manager** at the top left and navigate to Tools > General > CRXDE Lite. The **CRXDE Lite** screen opens.

10. Browse to apps > trainingproject > config > org.apache.sling.commons.log.LogManager.factory.config-trainingproject. The org.apache.sling.commons.log.LogManager.factory.config-trainingproject is selected, as shown:



11. Double-click `org.apache.sling.commons.log.LogManager.factory.config-trainingproject`. The Properties window opens. Observe the values for the log properties:

Properties		Access Control	Replication	Console	Build Info
Name	Type	Value			
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}			

12. Open a browser and go to <http://localhost:4502/system/console/configMgr>. The **Adobe Experience Manager Web Console Configuration** page opens, as shown:

The screenshot shows the 'Adobe Experience Manager Web Console Configuration' interface. At the top, there's a navigation bar with links for Main, OSGI, Sling, Status, and Web Console, along with a Log out button. Below the navigation, a message states 'Configuration Admin Service is running.' and a warning about deprecated configurations. The main content area is a table titled 'Configurations' with columns for Name, Bundle, and Actions. The table lists several configurations, including 'A scheduled task', 'Ad-hoc Task Purge', 'Adobe AEM Analytics Adapter Factory', 'Adobe AEM Analytics HTTP Client', 'Adobe AEM Analytics Report Importer', and 'Adobe AEM Classifications Exporter'. Each row in the table has edit, delete, and other action icons.

13. Use your browser to search for and find **Apache Sling Logging Logger Configuration**. The Apache Sling Logging Logger Configuration is listed, as shown:

The screenshot shows a search results page in the AEM console with the query 'Sling Logging Logger Configuration'. The results list various logging configurations, including 'Queue: Granite Workflow External Process Polling Queue', 'Queue: Granite Workflow Queue', 'Queue: Granite Workflow Timeout Queue', 'Queue: IDS Processing Queue', 'Queue: Maintenance Queue', 'Queue: org/apache/sling/distribution/queue/{0}', 'Queue: Pre Upgrade Tasks Queue', 'Queue: Scene7 Synchronization', 'Apache Sling Job Thread Pool', 'Apache Sling Jobs Health Check', 'Apache Sling JSP Script Handler', 'Apache Sling Jsp Script Handler Health Check', 'Apache Sling Log Tracer', 'Apache Sling Logging Configuration', and 'Apache Sling Logging Logger Configuration'. The last two items in the list, 'Apache Sling Logging Configuration' and 'Apache Sling Logging Logger Configuration', are highlighted with a red rectangular box. On the right side of the results, there's a sidebar for 'Apache Sling Commons Log' with icons for edit, delete, and other actions.

14. Under Apache Sling Logging Logger Configuration, find the configuration called, logs/project-trainingproject.log: info.
15. Click **Apache Sling Logging Logger Configuration** to open it. The logs for Apache Sling Logging Logger Configuration opens, as shown:

Apache Sling Logging Logger Configuration	
✓	» logs/access.log: INFO
✓	» logs/audit.log: INFO
✓	» logs/auditlog.log: INFO
✓	» logs/error.log: ERROR
✓	» logs/history.log: INFO
✓	» logs/project-trainingproject.log: info
✓	» logs/project-we-retail.log: info
»	logs/request.log: INFO

16. Expand **logs/project-trainingproject.log: info** and verify the properties are same as the configuration node.

**Apache Sling Logging Logger Configuration**

Configure Loggers with levels, pattern and destination. See <http://sling.apache.org/site/logging.html> for more detailed documentation and description.

Log Level	<input checked="" type="text"/> Information
Log File	<input type="text"/> logs/project-trainingproject.log
Message Pattern	<input type="text"/> {0,date,yyyy-MM-dd HH:mm:ss,SSSS} {4} {3} {5}
Logger	<input type="text"/> com.adobe.training
Additivity	<input type="checkbox"/>

⚠ 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.additiv)

**Configuration Information**

Persistent Identity (PID)	<input type="text"/> org.apache.sling.commons.log.LogManager.factory.config.7c70446c-a918-44a7-ad57-e4e84ca3c91d
Factory Persistent Identifier (Factory PID)	<input type="text"/> org.apache.sling.commons.log.LogManager.factory.config
Configuration Binding	<input type="text"/> slinginstall:C:\adobe\AEM\author\crx-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 custom loggers

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

1. Open the AEM instance folder (**C:\adobe\AEM\author**), as shown:

OSDisk (C:) > adobe > AEM > author			
	Name	Date modified	Type
⚡	crx-quickstart	3/13/2018 2:54 PM	File folder
⚡	aem-author-4502.jar	2/12/2018 4:01 PM	JAR File
⚡	license.properties	1/11/2018 9:32 AM	PROPERTIES File

2. Navigate to **crx-quickstart\logs\**. The list of log files is shown:

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

3. Open the **Log Support** console at <http://localhost:4502/system/console/slinglog>. The **Loggers** opens.

4. Click **Add new Logger**, as shown.

**Adobe Experience Manager Web Console**  
**Log Support**

Main OSGi Sling Status Web Console

Log Service Stats: 3965 categories, 7 appender, 0 Dynamic appenders

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**



Notice that a new row got added, as shown:

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	

5. On the Log Level drop-down menu (where **INFO** is the default), select **DEBUG**, as shown:



6. Enter the Log File (name) as, **logs\LoginTrace.log**.

7. Enter the Logger as, **org.apache.sling.auth.core.impl.SlingAuthenticator**, as shown:



8. Click **Save**. You will see the logger is created successfully, as shown:

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	
DEBUG	logs\LoginTrace.log	org.apache.sling.auth.core.impl.SlingAuthenticator	
INFO	logs\error.log	ROOT	
INFO	logs\request.log	log.request	

9. Navigate to **crx-quickstart\logs** directory again and see if the **LoginTrace.log** was created successfully, as shown:

Name	Date modified	Type	Size
access.log	3/13/2018 3:20 PM	Text Document	28 KB
access.log.2018-03-08	3/8/2018 3:28 PM	2018-03-08 File	100 KB
audit.log	2/13/2018 2:10 PM	Text Document	0 KB
auditlog.log	2/13/2018 2:10 PM	Text Document	0 KB
error.log	3/13/2018 3:18 PM	Text Document	2,708 KB
error.log.2018-03-07	3/8/2018 9:28 AM	2018-03-07 File	2,727 KB
error.log.2018-03-08	3/8/2018 3:17 PM	2018-03-08 File	14 KB
history.log	2/24/2018 10:16 PM	Text Document	2 KB
LoginTrace.log	3/13/2018 3:20 PM	Text Document	1 KB
project-trainingproject.log	3/13/2018 2:55 PM	Text Document	2 KB
project-trainingproject.log.2018-03-07	3/7/2018 4:38 PM	2018-03-07 File	2 KB
project-we-retail.log	2/13/2018 2:14 PM	Text Document	0 KB
request.log	3/13/2018 3:20 PM	Text Document	16 KB

+

10. Log out from **AEM**, and then log in again.
11. Open the **LoginTrace.log** file and observe how the log file contains the entries as you configured them in the logger settings.

```
DEBUG [tcp25820528-64] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
*DEBUG* [tcp25820528-1456] org.apache.sling.auth.core.impl.SlingAuthenticator doHandleSecurity: Trying to get a session for null
*DEBUG* [tcp25820528-1464] org.apache.sling.auth.core.impl.SlingAuthenticator doHandleSecurity: Trying to get a session for null
*DEBUG* [tcp25820528-1472] org.apache.sling.auth.core.impl.SlingAuthenticator doHandleSecurity: Trying to get a session for null
*DEBUG* [tcp25820528-1465] org.apache.sling.auth.core.impl.SlingAuthenticator doHandleSecurity: Trying to get a session for null
*DEBUG* [tcp25820528-60] org.apache.sling.auth.core.impl.SlingAuthenticator doHandleSecurity: Trying to get a session for null
*DEBUG* [tcp25820528-1456] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
*DEBUG* [tcp25820528-1464] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
*DEBUG* [tcp25820528-1472] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
*DEBUG* [tcp25820528-1465] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
*DEBUG* [tcp25820528-60] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
*DEBUG* [tcp25820528-1500] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
*DEBUG* [tcp25820528-1500] org.apache.sling.auth.core.impl.SlingAuthenticator setHandleSecurity: Trying to get a session for null
*DEBUG* [tcp25820528-61] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
*DEBUG* [tcp25820528-61] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
*DEBUG* [tcp25820528-1501] org.apache.sling.auth.core.impl.SlingAuthenticator doHandleSecurity: Trying to get a session for null
*DEBUG* [tcp25820528-1501] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
*DEBUG* [tcp25820528-1501] org.apache.sling.auth.core.impl.SlingAuthenticator doHandleSecurity: Trying to get a session for null
*DEBUG* [tcp25820528-1472] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
*DEBUG* [tcp25820528-1472] org.apache.sling.auth.core.impl.SlingAuthenticator doHandleSecurity: Trying to get a session for null
*DEBUG* [tcp25820528-60] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
*DEBUG* [tcp25820528-60] org.apache.sling.auth.core.impl.SlingAuthenticator doHandleSecurity: Trying to get a session for null
*DEBUG* [tcp25820528-1501] org.apache.sling.auth.core.impl.SlingAuthenticator doHandleSecurity: Trying to get a session for null
*DEBUG* [tcp25820528-1501] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
*DEBUG* [tcp25820528-1500] org.apache.sling.auth.core.impl.SlingAuthenticator doHandleSecurity: Trying to get a session for null
*DEBUG* [tcp25820528-1465] org.apache.sling.auth.core.impl.SlingAuthenticator doHandleSecurity: Trying to get a session for null
*DEBUG* [tcp25820528-1465] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
*DEBUG* [tcp25820528-1501] org.apache.sling.auth.core.impl.SlingAuthenticator doHandleSecurity: Trying to get a session for null
*DEBUG* [tcp25820528-1501] org.apache.sling.auth.core.impl.SlingAuthenticator setAttributes: ResourceResolver stored as request attribute: user=admin
```



**Note:** The log entries ensure the events are captured in the log file.

# Deep Dive into OSGi Architecture



## Introduction

Adobe Experience Manager (AEM) architecture consists of frameworks such as Open Services Gateway Initiative (OSGi) and Apache Sling. OSGi defines a dynamic component written in Java. The OSGi specifications enable a development model where dynamic application comprises reusable components.

## Objectives

After completing this course, you will be able to:

- Explain OSGi architecture
- Implement a bundle activator
- Explain OSGi annotations
- Implement OSGi configurations

# OSGi – An Overview

The OSGi service platform is a Java-based application server for networked devices.

In AEM, OSGi is used to control the composite bundles. OSGi enables you to create applications from smaller, reusable, and collaborative components.

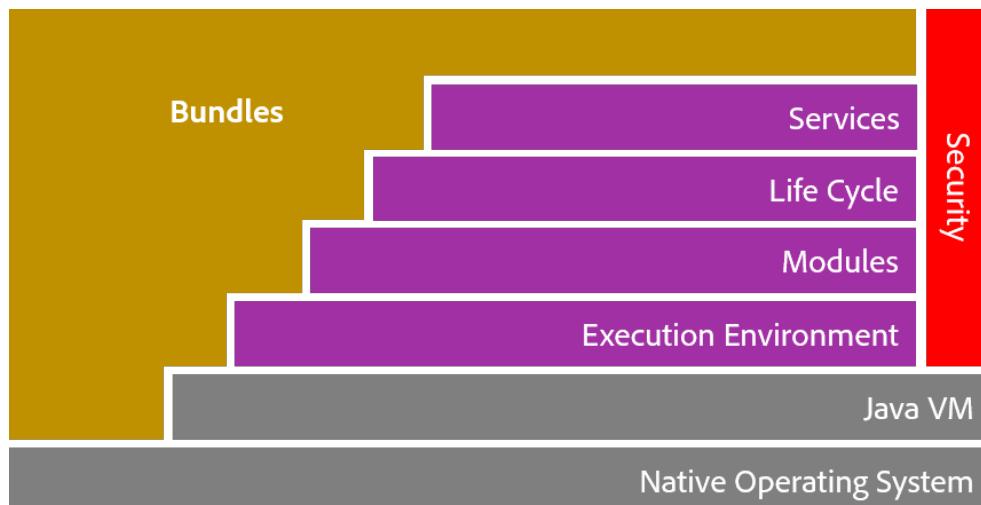
AEM composite bundles can be installed, started, or stopped individually. The interdependencies are automatically handled. Each OSGi component is contained in one of the bundles. Bundles are the OSGi components made by the developers.

The OSGi specifications enable components to hide their implementations from other components when communicating through services, which are objects specifically shared between components. This surprisingly simple model has far reaching effects for almost any aspect of the software development process.

An OSGi application is a collection of bundles that interact using service interfaces. Developers can develop and deploy bundles independently. Bundles and their associated services may appear or disappear at any time.

## OSGi Architecture

The OSGi has a layered model, as shown in the following figure:



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

## Bundles

Bundles are built on Java's existing standard way of packaging classes and resources together—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:

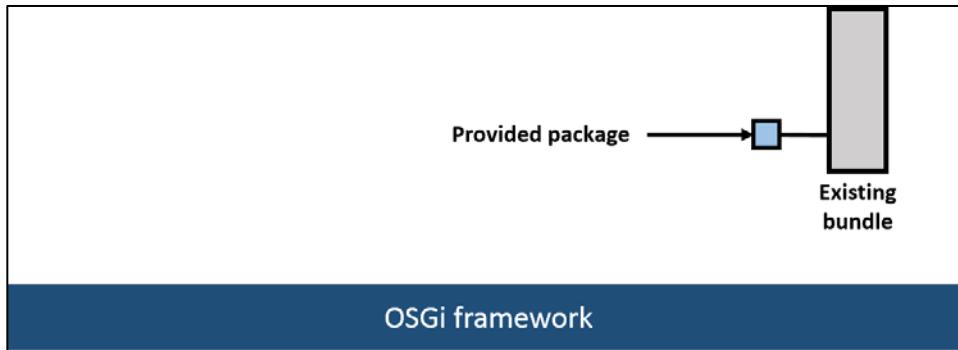
- Bundle name(s):
  - A *symbolic* name used by OSGi to determine the bundle's unique identity
  - An optional, human-readable, descriptive name
- Bundle version
- The list of services imported and exported by this bundle
- Optional information, such as the:
  - Minimum Java version the bundle requires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

### Life Cycle:

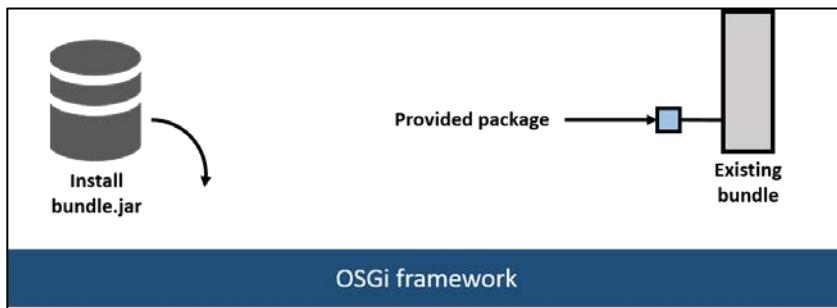
A life cycle layer adds bundles that can be dynamically installed, started, stopped, updated and uninstalled. Bundles rely on the module layer for class loading, but add an API to manage the modules in run time. The life cycle layer introduces dynamics that are generally not part of an application. Extensive dependency mechanisms are used to assure the correct operation of the environment. Life cycle operations are fully protected with the security architecture.

## Dependency Management Resolution in OSGi

A bundle is present in the container.

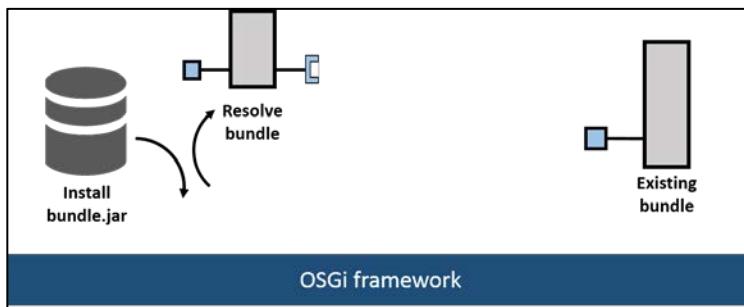


When a new bundle is provided, it is installed into the OSGi container.

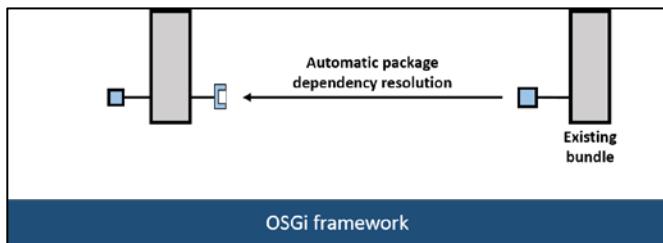


The OSGi container resolves the new bundle.

**Note:** The Resolved state of a bundle is the state it can reach after being installed and when all its required dependencies are satisfied.



The OSGi container provides automatic dependency resolution.



+

## Modules

Modularity is at the core of the OSGi specifications and is embodied in the bundle concept. Modularity is about keeping things local and not sharing. You should be familiar with the term *bundle* in Java, which means a JAR file. In Java bundle, the contents of the JAR are completely visible to all the other JARs. In OSGi bundle, the contents of JAR are hidden unless they are explicitly exported. If a OSGi bundle wants to use another JAR, it must explicitly import the parts it needs. Therefore, there is no sharing of the JAR.

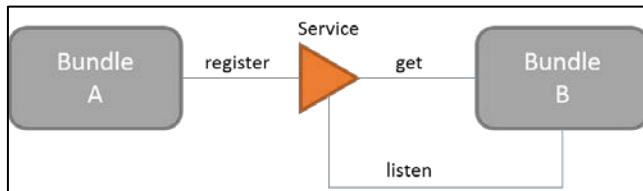
Although the code hiding and explicit sharing provides many benefits, such as allowing multiple versions of the same library being used in a single VM, the code sharing is there only to support OSGi services model. The OSGi services model is about bundles that collaborate.

## 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 the 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:



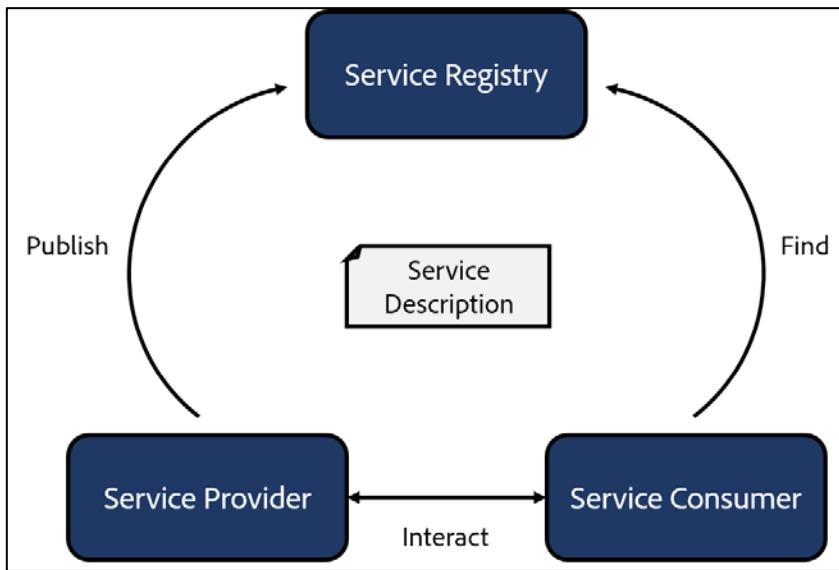
Each service registration has a set of standard and custom properties. An expressive filter language is available to select only the services in which you are interested. Properties can be used to find the proper service or can play other roles at the application level.

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 by using a publish/find/bind mechanism. For example, 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, as shown:

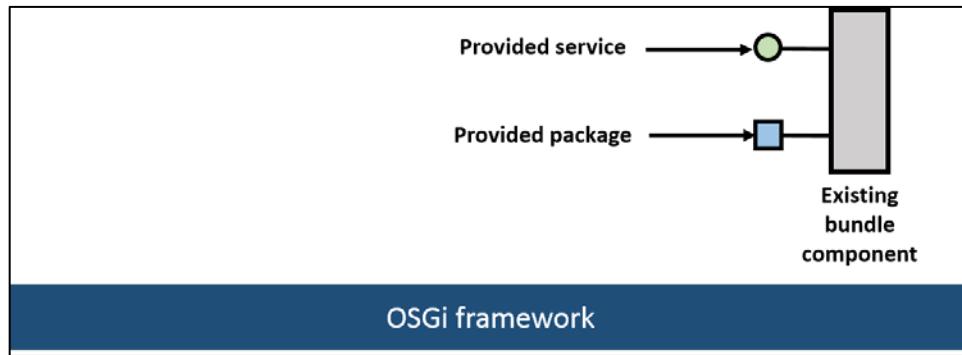


Instead of registering a listener/consumer object with the service provider, the consumer creates an object that implements the OSGi listener interface, providing a method that should be called when the event of interest occurs. When the desired event occurs, the service provider requests a list of all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

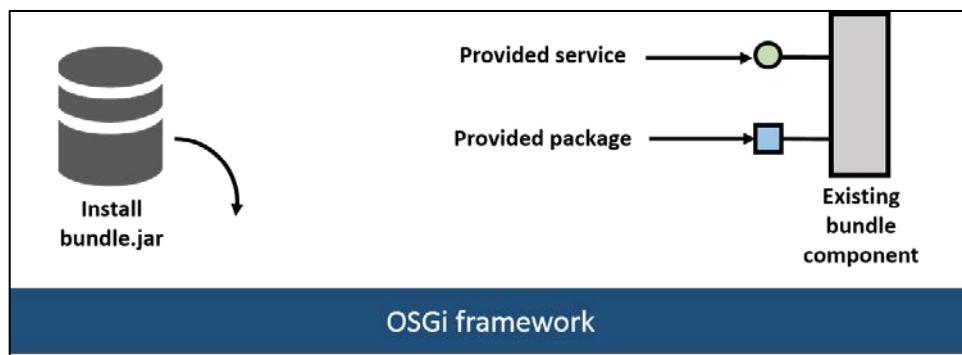
## Dynamic Service Lookup

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

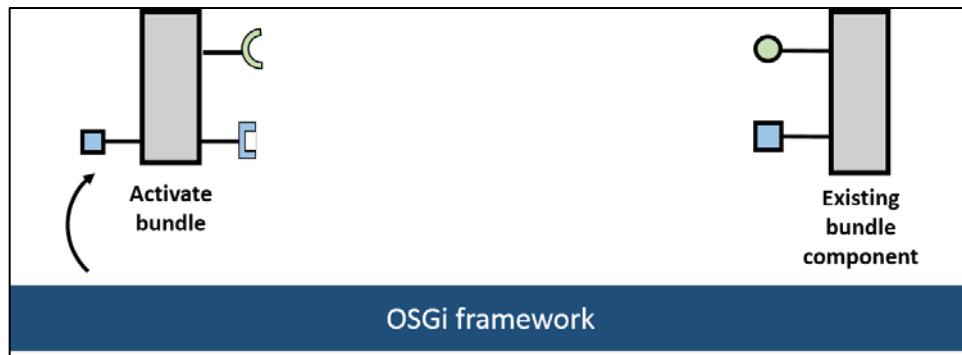
1. Examine the existing service.



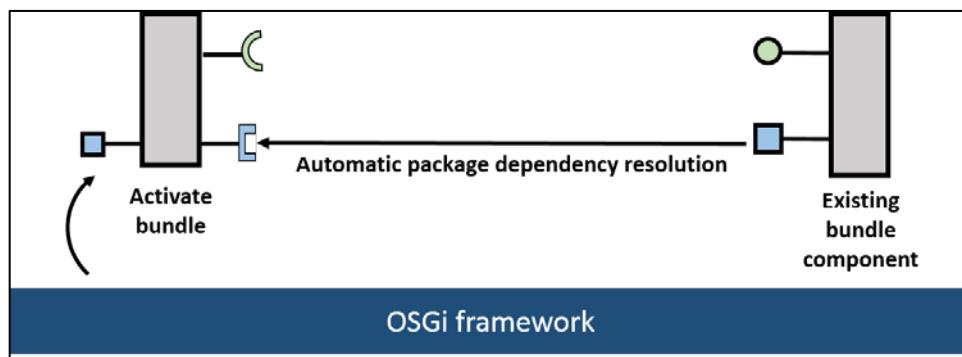
2. Install the new bundle.



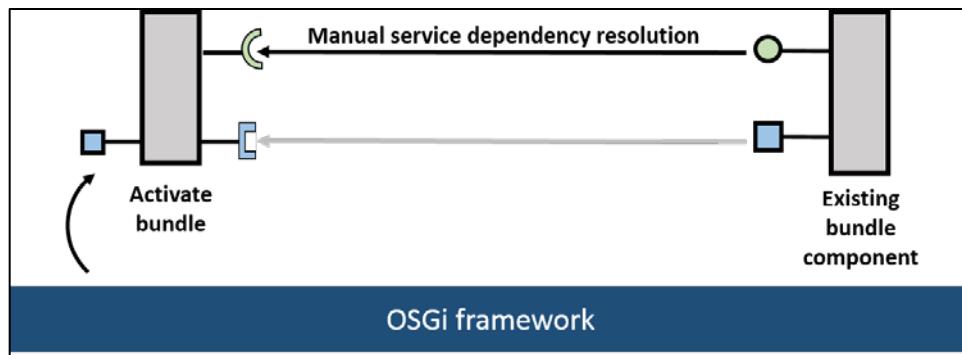
3. Activate the new bundle.



4. Automatic resolution of package dependency.



5. Manual service dependency resolution.



## OSGi Service Advantages

In OSGi-based systems, functionality is mainly provided through services. Services implement one or more interfaces, which define the type of service provided. It is the life cycle 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:

- 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 of Declarative Services:

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

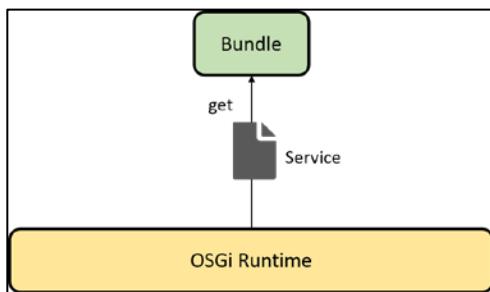
Components are declared by using XML configuration files contained in the respective bundle, and listed in the Service-Component bundle manifest header. Declarative services resolve the bundle through XML configuration files. You may handwrite and register these configuration files.

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



## Deployment of Bundles

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 need 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:** There are no interdependencies between these bundles, so developers have more freedom to change the bundles at a later stage. This simplifies application development tasks process and reduces bugs .
- **Reuse:** The OSGi component model makes it very easy to use third-party components in an application.
- **Real World:** The OSGi framework is dynamic and is a perfect match for many real-world scenarios. Applications can reuse the powerful primitives of the service registry in their own domain. This reduces the time for writing code and provides global visibility, debugging tools, and more functionality.
- **Easy Deployment:** The OSGi technology specifies how components are installed and managed. Therefore, it is easy to deploy and integrate OSGi technology in the 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 enables 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. Only the bundles that can collaborate are wired together in the same class space. This enables various bundles to function with their own library.
- **Simple:** Easy-to-use annotations inform the runtime how a class wants to use the dynamics, configuration, and dependencies on other services. The default configurations completely hide the dynamics and OSGi. This simple model enables the gradual use of more advanced features.
- **Small:** The OSGi Release 4 Framework can be implemented in about a 300KB JAR file. Therefore, OSGi can be used on a large range of devices—from very small to small to mainframes.
- **Fast:** One of the primary responsibilities of the OSGi framework is loading the classes from bundles. OSGi prewires bundles and exactly knows which bundle provides the class for each bundle. This lack of searching is a significant speed-up factor at startup.
- **Secure:** Java has a very powerful fine-grained security model at the bottom, but it has turned out very hard to configure in practice. The result is that most secure Java applications are running with a binary choice, no security or very limited capabilities. The OSGi security model also improves the usability. In this model, the bundle developer can specify the requested security details in an easily audited form, and the operator of the environment remains fully in charge.
- **Non-Intrusive:** The applications (bundles) in an OSGi environment can use any facility of VM. The best practice in OSGi is to write Plain Old Java Objects. There is no special interface required for OSGi services—even a Java String object can act as an OSGi service. With this strategy, you can port the application code to another environment easily.



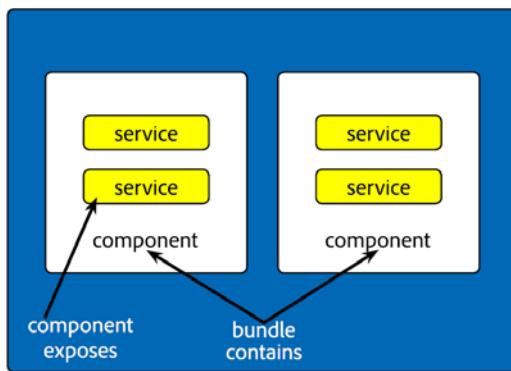
- **Runs Everywhere:** The OSGi APIs do not use classes that are not available on all environments. A bundle does not start if it contains the code that is not available in the execution environment.

## Components

Components are the main building blocks for OSGi applications.

A component:

- Is provided by a bundle
- Is a piece of software managed by an OSGi container
- Is a Java object created and managed by a OSGi container.
- Can provide a service
- Can implement one or more Java interfaces as services



A component can publish itself as a service and/or can have dependencies on other components and services. The OSGi container will activate a component only when all the required dependencies are met or available. Each component has an implementation class, and can optionally implement a public interface providing this service.

A service can be consumed or used by components and other services. Basically, a bundle needs the following to become a component:

- An XML file, where you describe the service the bundle provides and the dependencies of other services of the OSGi framework
- A manifest file header entry to declare that the bundle behaves as a component
- The 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

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

## Annotations in OSGi

You can annotate OSGi components by 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 need to implement Activators, which extend the `DependencyActivatorBase` class. The `@Component` annotation enables the OSGi Declarative Services to register your component.

If the `@Component` 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

Some of the attributes you can use with the @Component annotation are:

- **metatype**: Indicates whether the Metatype Service data is generated. If this parameter is set to true, 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 enables 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 service interfaces are provided by the component, and which service interfaces are provided. 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 Service Component Runtime. 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 the <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 by using this tag to identify the component if it is registered as a service. For example, you can set the service.description and service.vendor properties.

## Java Compiler Annotations

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

- **@Override**
- **@SuppressWarnings**

### @Override

The **@Override** annotation informs the compiler that 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 enables components to get or retrieve configuration, and provides the entry point for management agents to retrieve and update configuration data. The combination of the configuration properties and the meta type description for a given PID is used to build the UI to configure the service and/or component.

# Exercise 1: Implement a bundle activator

---

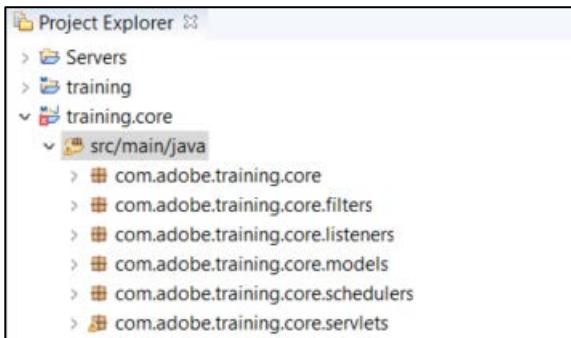
In this exercise, you will implement a bundle activator class that exposes a start method when the bundle is started and run a stop method when the bundle is stopped:

This exercise includes two tasks:

1. Create a new java class
2. Deploy the project

## Task 1: Create a new Java class

1. Launch **Eclipse** by double-clicking the **Eclipse** shortcut on the desktop. The Eclipse Launcher opens.
2. Specify the Workspace as **C:\Workspace**, and click **Launch**. The Workspace opens.
3. In Project Explorer, navigate to **training.core > src/main/java**, as shown:



**Note:** To activate the code inside the bundle, you need to implement the `org.osgi.framework.BundleActivator` interface.



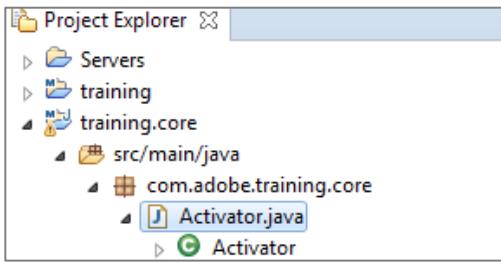
**Global Note:** Whenever you encounter any error related to missing package or duplicate import in Eclipse, please clean your project. Click **Project > Clean**.

4. Copy the file **Activator.java** from **Exercise\_Files/08\_Deep\_Dive\_OSGi**.



**Note:** The code is provided as part of the **Exercise\_Files** under **/Exercise\_Files/08\_Deep\_Dive\_OSGi**. Copy and paste the file from the exercise files referenced to `Activator.java` to Eclipse. Please do not copy the code from this exercise book. The code given here is only for illustrative purposes.

5. Right-click **com.adobe.training.core** and paste the file. The **Activator.java** class is created, as shown:

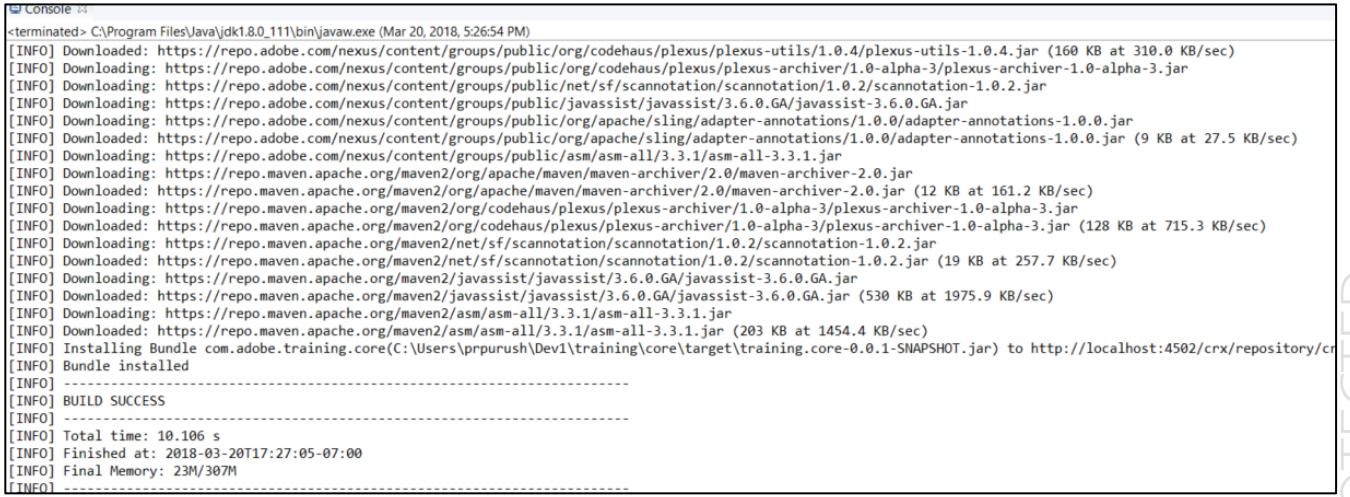


**Note:** Activator.java uses the start() and stop() methods to create log entries based on the bundle status.

```
1. package com.adobe.training.core;
2.
3. import org.osgi.service.component.annotations.Activate;
4. import org.osgi.service.component.annotations.Component;
5. import org.osgi.service.component.annotations.Deactivate;
6.
7. import org.slf4j.Logger;
8. import org.slf4j.LoggerFactory;
9.
10. @Component
11.
12. public class Activator {
13.     private final Logger logger = LoggerFactory.getLogger(getClass());
14.
15.     @Activate
16.     public void startBundle() {
17.         logger.info("#####Bundle Started#####");
18.     }
19.
20.     @Deactivate
21.     public void stopBundle() {
22.         logger.info("#####Bundle Stopped#####");
23.     }
24. }
```

## Task 2: Deploy the project

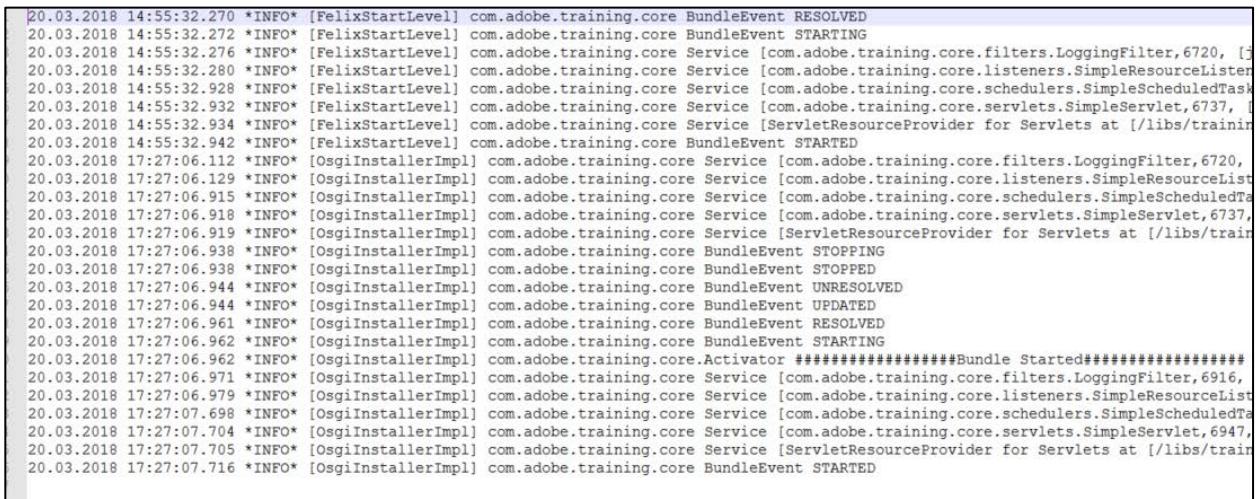
1. In Project Explorer, right-click **training.core**, and select Run As > Maven install. The build starts.
2. Verify that the bundle is installed successfully, as shown:



```

[terminated] > C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Mar 20, 2018, 5:26:54 PM)
[INFO] Downloaded: https://repo.adobe.com/nexus/content/groups/public/org/codehaus/plexus/plexus-utils/1.0.4/plexus-utils-1.0.4.jar (160 KB at 310.0 KB/sec)
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar
[INFO] Downloaded: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar (9 KB at 27.5 KB/sec)
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/asm/asm-all/3.3.1/asm-all-3.3.1.jar
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.0/maven-archiver-2.0.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.0/maven-archiver-2.0.jar (12 KB at 161.2 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar (128 KB at 715.3 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar (19 KB at 257.7 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/javassist/javassist-3.6.0.GA.jar (530 KB at 1975.9 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/asm/asm-all/3.3.1/asm-all-3.3.1.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/asm/asm-all/3.3.1/asm-all-3.3.1.jar (203 KB at 1454.4 KB/sec)
[INFO] Installing Bundle com.adobe.training.core(C:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar) to http://localhost:4502/crx/repository/crx
[INFO] Bundle installed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 10.106 s
[INFO] Finished at: 2018-03-20T17:27:05-07:00
[INFO] Final Memory: 23M/307M
[TNEO]
```

3. On your desktop, navigate to **crx-quickstart\logs**.
4. Open the log file **project-trainingproject.log** by using a text editor.
5. Scroll down to the bottom of the file, and observe the log messages that indicate that the bundle is started, as shown:



```

20.03.2018 14:55:32.270 *INFO* [FelixStartLevel] com.adobe.training.core BundleEvent RESOLVED
20.03.2018 14:55:32.272 *INFO* [FelixStartLevel] com.adobe.training.core BundleEvent STARTING
20.03.2018 14:55:32.276 *INFO* [FelixStartLevel] com.adobe.training.core Service [com.adobe.training.core.filters.LoggingFilter, 6720, ...]
20.03.2018 14:55:32.280 *INFO* [FelixStartLevel] com.adobe.training.core Service [com.adobe.training.core.listeners.SimpleResourceListener]
20.03.2018 14:55:32.282 *INFO* [FelixStartLevel] com.adobe.training.core Service [com.adobe.training.core.schedulers.SimpleScheduledTask]
20.03.2018 14:55:32.932 *INFO* [FelixStartLevel] com.adobe.training.core Service [com.adobe.training.core.servlets.SimpleServlet, 6737, ...]
20.03.2018 14:55:32.934 *INFO* [FelixStartLevel] com.adobe.training.core Service [ServletResourceProvider for Servlets at [/libs/training]]
20.03.2018 14:55:32.942 *INFO* [FelixStartLevel] com.adobe.training.core BundleEvent STARTED
20.03.2018 17:27:06.112 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service [com.adobe.training.core.filters.LoggingFilter, 6720, ...]
20.03.2018 17:27:06.129 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service [com.adobe.training.core.listeners.SimpleResourceList]
20.03.2018 17:27:06.915 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service [com.adobe.training.core.schedulers.SimpleScheduledTa...
20.03.2018 17:27:06.918 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service [com.adobe.training.core.servlets.SimpleServlet, 6737, ...]
20.03.2018 17:27:06.919 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service [ServletResourceProvider for Servlets at [/libs/training]]
20.03.2018 17:27:06.938 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent STOPPING
20.03.2018 17:27:06.938 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent STOPPED
20.03.2018 17:27:06.944 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent UNRESOLVED
20.03.2018 17:27:06.944 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent UPDATED
20.03.2018 17:27:06.961 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent RESOLVED
20.03.2018 17:27:06.962 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent STARTING
20.03.2018 17:27:06.962 *INFO* [OsgiInstallerImpl] com.adobe.training.core.Activator #####Bundle Started#####
20.03.2018 17:27:06.971 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service [com.adobe.training.core.filters.LoggingFilter, 6916, ...]
20.03.2018 17:27:06.979 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service [com.adobe.training.core.listeners.SimpleResourceList]
20.03.2018 17:27:07.698 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service [com.adobe.training.core.schedulers.SimpleScheduledTa...
20.03.2018 17:27:07.704 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service [com.adobe.training.core.servlets.SimpleServlet, 6947, ...]
20.03.2018 17:27:07.705 *INFO* [OsgiInstallerImpl] com.adobe.training.core Service [ServletResourceProvider for Servlets at [/libs/training]]
20.03.2018 17:27:07.716 *INFO* [OsgiInstallerImpl] com.adobe.training.core BundleEvent STARTED

```

6. Close the log file.

## Exercise 2: Create and use a custom service

---

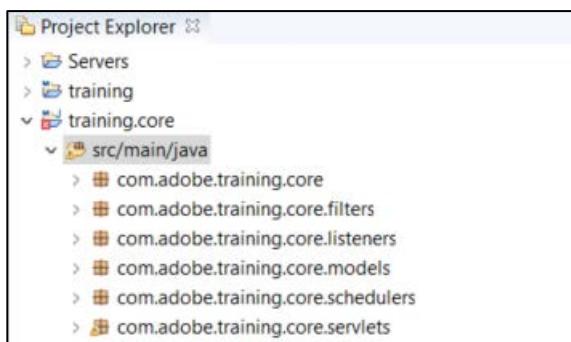
In this exercise, you will create and use a custom service.

This exercise includes three tasks:

1. Create a service and an implementation
2. Deploy the bundle and expose the service in HTL
3. Test the service

### Task 1: Create a service and an implementation

1. In Project Explorer, navigate to **training.core > src/main/java**, as shown:

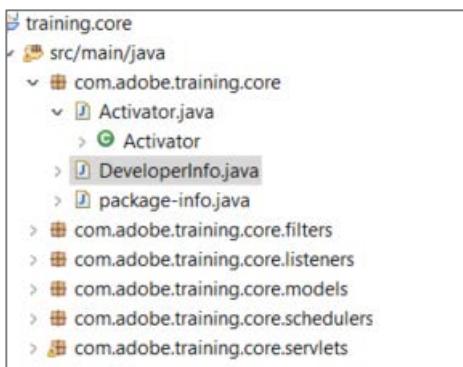


2. Copy the file **DeveloperInfo.java** from Exercise\_Files/08\_Deep\_Dive\_OSGi.



**Note:** The code is provided as part of the Exercise\_Files under Exercise\_Files/08\_Deep\_Dive\_OSGi. Copy and paste the file from the exercise files referenced to DeveloperInfo.java to Eclipse. Please do not copy the code from this exercise book. The code given here is only for illustrative purposes.

3. Right-click **com.adobe.training.core**, and paste the file. The **DeveloperInfo.java** is created, as shown:



---

4. Implement the service interface to get the information about the bundle developer in the **DeveloperInfo.java** class, as shown:



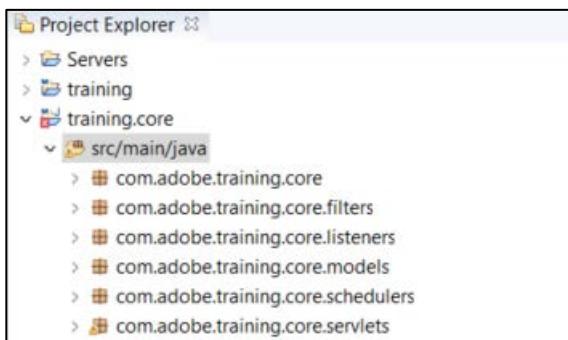
**Note:** The code is provided as part of the Exercise\_Files under Exercise\_Files/08\_Deep\_Dive\_OSGi.

Copy and paste the file from the exercise files folder under com.adobe.training.core in Eclipse. Please do not copy the code from this exercise book. The following code is only for illustrative purposes.

```
1. package com.adobe.training.core;
2. /**
3. * Service interface to get the information about the bundle Developer
4. *
5. * Example HTML:
6. * <h3 data-sly-
7. * use.devInfo="com.adobe.training.core.DeveloperInfo">Developer Info: ${devInfo.DeveloperInfo}</h3>
8. *
9. * Example code can be inserted into a HTML component:
10. * /apps/trainingproject/components/structure/page/partials/main.html
11. *
12. * Example JSP:
13. * com.adobe.training.core.DeveloperInfo devInfo = sling.getService(com.adobe.training.core.DeveloperInfo.cl
14. ass)
15. * <h3>Developer Info: <%= devInfo.getDeveloperInfo() %></h3>
16. *
17. * @author Kevin Nennig (nennig@adobe.com)
18. */
19. public interface DeveloperInfo {
20.     public String getDeveloperInfo();
21. }
```

5. Save the changes.

6. In Project Explorer, navigate to training.core > src/main/java, as shown:

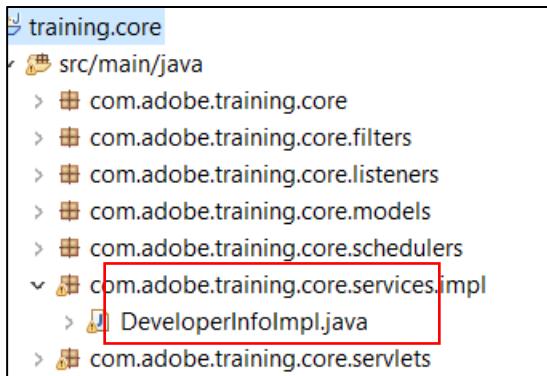


7. Right-click **src/main/java**, and select **New > Package**. The **New Java Package** window opens.
8. Enter the following value for the **Package** field: **com.adobe.training.core.services.impl** and click **Finish**.
9. Copy the file **DeveloperInfoImpl.java** from Exercises\_Folder/08\_Deep\_Dive\_OSGi.



**Note:** The code is provided as part of the Exercise\_Files under Exercise\_Files/08\_Deep\_Dive\_OSGi. Copy and paste the file from the exercise files referenced to **DeveloperInfoImpl.java** to Eclipse. Please do not copy the code from this exercise book. The code given here is only for illustrative purposes.

10. Right-click **com.adobe.training.core.services.impl**, and paste the file. The **DeveloperInfoImpl.java** class is created, as shown:



11. Examine the implementation of the *Simple* component of the **DeveloperInfoImpl.java** class, as shown:

```

1. package com.adobe.training.core.services.impl;
2.
3. import org.osgi.service.component.ComponentContext;
4. import org.osgi.service.component.annotations.Activate;
5. import org.osgi.service.component.annotations.Component;
6. import org.osgi.service.component.annotations.Deactivate;
7. import org.osgi.service.component.annotations.Modified;
8. import org.slf4j.Logger;
9. import org.slf4j.LoggerFactory;
10.
11. import com.adobe.training.core.DeveloperInfo;
12.
13. /**
14. * Component implementation of the DeveloperInfo Service.
15. */

```

+

```
16.  
17. @Component(service = DeveloperInfo.class, name = "TrainingDeveloperInfo")  
18.  
19. public class DeveloperInfoImpl implements DeveloperInfo {  
20.     private final Logger logger = LoggerFactory.getLogger(getClass());  
21.  
22.     //local variables to hold OSGi config values  
23.     private boolean showDeveloper;  
24.     private String developerName;  
25.     private String[] developerHobbiesList;  
26.     private String langPreference;  
27.  
28.     @Activate  
29.     @Modified  
30.     //http://blogs.adobe.com/experiencedelivers/experience-  
        management/osgi_activate_deactivatesignatures/  
31.     protected void activate(ComponentContext config) {  
32.         logger.info("#####Component config saved");  
33.     }  
34.  
35.     @Deactivate  
36.     protected void deactivate() {  
37.         logger.info("#####Component (Deactivated) Good-bye " + developerName);  
38.     }  
39.  
40.     /* Method used to show a simple OSGi service/component relationship */  
41.     public String getDeveloperInfo(){  
42.         return "Hello! I do not know who my developer is. I am a product of random development!!!";  
43.     }  
44.  
45.  
46. }
```

12. Examine the method `getDeveloperInfo()` in the `DeveloperInfoImpl.java` class.

13. Save the changes.

## Task 2: Deploy the bundle and expose the service in HTL

1. In Project Explorer, right-click **training.core** and select **Run As > Maven install**. The build starts.
2. Verify that the bundle is installed successfully, as shown:

```
[INFO] --- maven-bundle-plugin:3.3.0:bundle (default-bundle) @ training.core ---
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ training.core ---
[INFO] Installing C:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar to C:\Users\prpurush\.m2\repository\com\adobe\training.core\0.0.1-SNAPSHOT\training.core-0.0.1-SNAPSHOT.jar
[INFO] Installing C:\Users\prpurush\Dev1\training\core\pom.xml to C:\Users\prpurush\.m2\repository\com\adobe\training.core\0.0.1-SNAPSHOT\training.core-0.0.1-SNAPSHOT.pom
[INFO]
[INFO] --- maven-bundle-plugin:3.3.0: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:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar) to http://localhost:4502/crx/repository/crx/bundles
[INFO] Bundle installed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.858 s
[INFO] Finished at: 2018-03-21T11:59:14-07:00
[INFO] Final Memory: 31M/380M
[INFO] -----
```

3. Open a browser, and go to the AEM Web Console, <http://localhost:4502/system/console>. The **Adobe Experience Manager Web Console Bundles** page opens.
4. Expand the TrainingProject – Core (com.adobe.trianing.core), as shown:

Main	OSGi	Sling	Status	Web Console
Bundle information: 547 bundles in total - all 547 bundles active				
<input type="button" value="Apply Filter"/> <input type="button" value="Filter All"/>				
<b>ID</b>	<b>Name</b>			<b>Version</b>
547	TrainingProject - Core (com.adobe.training.core)			0.0.1.SNAPSHOT
	Symbolic Name	com.adobe.training.core		
	Version	0.0.1.SNAPSHOT		
	Bundle Location	jcr:install:/apps/trainingproject/install/training.core-0.0.1-SNAPSHOT.jar		
	Last Modification	Wed Mar 21 11:59:15 PDT 2018		
	Description	Core bundle for TrainingProject		
	Start Level	20		
	Exported Packages	com.adobe.training.core,version=1.0.0 com.adobe.training.core.filters,version=0.0.1 com.adobe.training.core.listeners,version=0.0.1 com.adobe.training.core.models,version=0.0.1 com.adobe.training.core.schedulers,version=0.0.1 com.adobe.training.core.servlets,version=0.0.1		
	Imported Packages	javax.annotation,version=0.0.0.1_008_JavaSE from org.apache.felix.framework (0) javax.inject,version=1.0.0 from org.apache.geronimo.specs.geronimo- atinject_1.0_spec (482) javax.servlet,version=2.6.0 from org.apache.felix.http.servlet-api (43)		

5. Scroll down, and look for the Service, **com.adobe.training.core.DeveloperInfo**.

- Verify that the service exists in the bundles, as shown:

Service ID 8565	Component ID: 3209 Types: com.adobe.training.core.DeveloperInfo Component Name: TrainingDeveloperInfo Component ID: 3210
-----------------	---

This indicates that the class was successfully installed in the OSGI.

- In Project Explorer, right-click **training.core**, and select **Run As > Maven install**. The build starts.
- In Eclipse, under Project Explorer, navigate **training.ui.apps > src/main/content/jcr\_root [nt:folder] > apps [nt:folder] > trainingproject [nt:folder] > components [nt:folder] > content [nt:folder] > helloworld [cq:Component]**.
- Expand **helloworld [cq:Component]** and double-click **helloworld.html**. The file opens in XML Editor.
- Add the following line to the **helloworld.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/08\_Deep\_Dive\_OSGi/html.txt**.

```
'<p data-sly-test="${properties.text}">Text property: ${properties.text}</p>  
<pre data-sly-use.devInfo="com.adobe.training.core.DeveloperInfo">Developer Info: ${devInfo.DeveloperInfo} </pre>  
<pre data-sly-use.hello="com.adobe.training.core.models.HelloWorldModel">
```

- Save changes to the file.



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



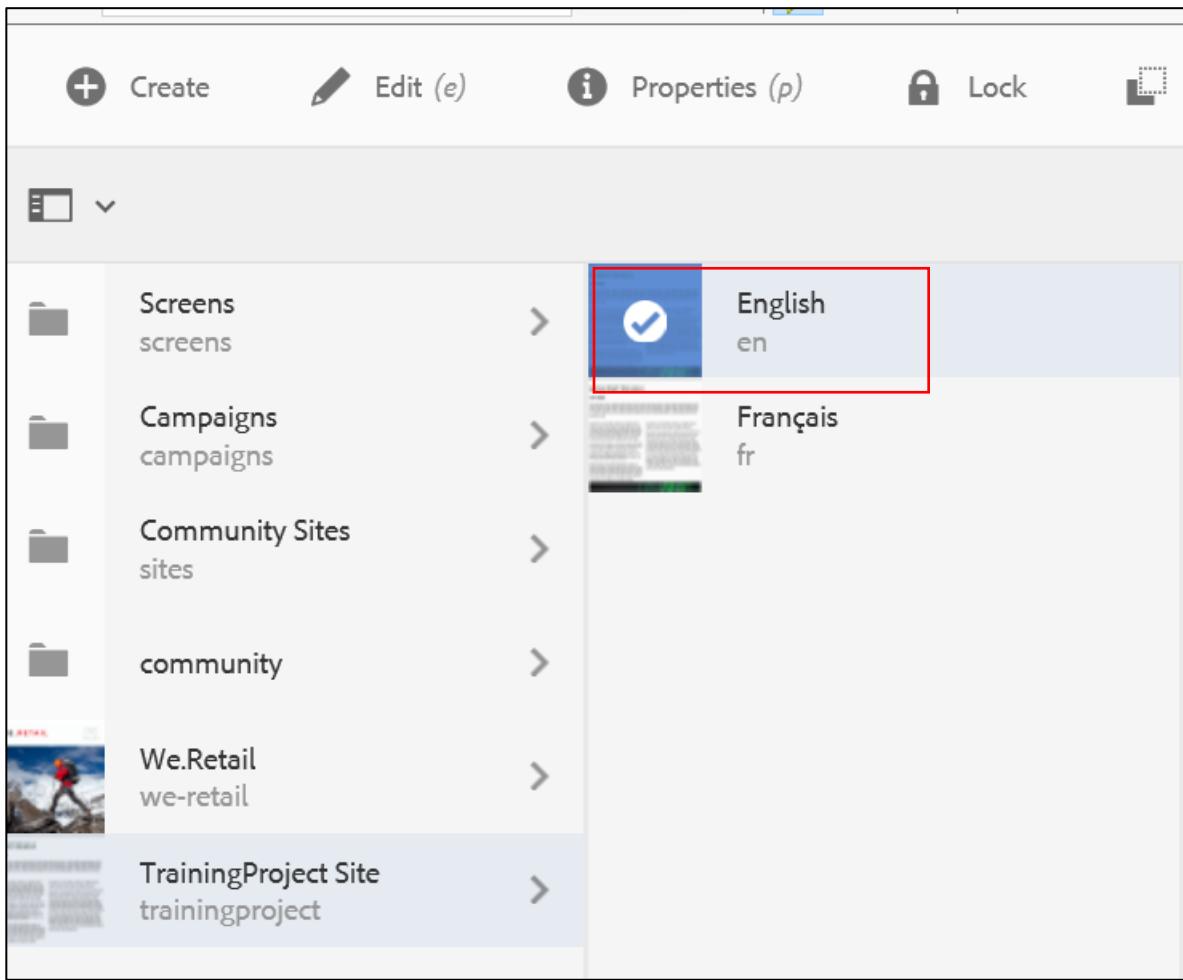
**Note:** If you have already set up your AEM Server connection on the Servers tab in Eclipse and started it, the main.html file needs to be synced with the JCR immediately after saving the changes.

- To verify whether your file successfully saved to the JCR, open **CRXDE Lite to /apps/trainingproject/components/content/helloworld/ helloworld.html** and see if that file contains your change.
- Check if the file contains the changes you made..
- Alternatively, you can force an update with full redeploy to the JCR by right-clicking **training.ui.apps** and selecting **Run AS > Maven Install**.
- Verify the bundle is installed successfully.



## Task 3: Test the service

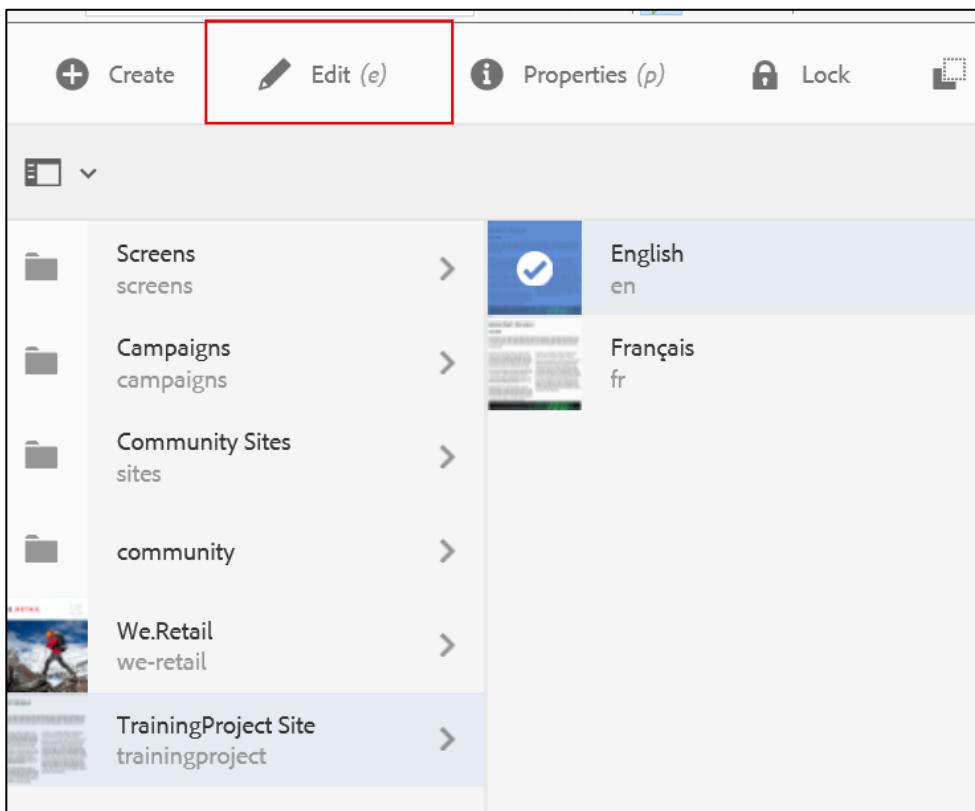
1. Log in to AEM, and click **Adobe Experience Manager** in the upper-left corner.
2. Click **Navigation > Sites**. The **Sites** console opens.
3. Select TrainingProject Site > English, as shown:



The screenshot shows the AEM Sites console interface. At the top, there are buttons for 'Create', 'Edit (e)', 'Properties (p)', 'Lock', and a copy icon. Below this is a toolbar with a folder icon and a dropdown menu. The main area displays a list of sites under the 'Screens' category. The 'TrainingProject Site' is highlighted with a blue background. To its right, there are two language entries: 'English en' (which is selected, indicated by a red border and a checkmark icon) and 'Français fr'. The other sites listed are 'Screens screens', 'Campaigns campaigns', 'Community Sites sites', and 'community'.

The page is updated with options.

4. Click **Edit (e)**, as shown:



5. Verify you see the message, as shown:



## Exercise 3: Code OSGi configurations

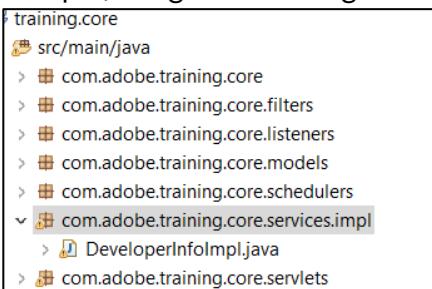
In this exercise, you will code the custom OSGi configurations for an OSGi component. This will allow an administrator to configure the component from the Web Console. These configurations can then be implementation specific by using run modes.

This exercise includes three tasks:

1. Update the OSGi component
2. Deploy the bundle
3. Test the service

### Task 1: Update the OSGi component

1. In Eclipse, navigate to training.core > src/main/java > com.adobe.training.core.services.impl, as shown:



2. Double-click **DeveloperInfoImpl.java** to open it in XML Editor.
3. Copy the contents from Exercise\_Files/08\_Deep\_Dive\_OSGi/DeveloperInfoImpl-v2.java to **DeveloperInfoImpl.java** in Eclispe, as shown:

```
1. package com.adobe.training.core.services.impl;  
2.  
3. import org.osgi.service.component.annotations.Activate;  
4. import org.osgi.service.component.annotations.Component;  
5. import org.osgi.service.component.annotations.Deactivate;  
6. import org.osgi.service.component.annotations.Modified;  
7. import org.osgi.service.metatype.annotations.Designate;  
8.  
9. import org.slf4j.Logger;  
10. import org.slf4j.LoggerFactory;  
11.  
12. import java.util.Arrays;
```

```
13.  
14. import com.adobe.training.core.DeveloperInfo;  
15. import com.adobe.training.core.services.DeveloperInfoConfiguration;  
16.  
17. /**  
18. * Component implementation of the DeveloperInfo Service. This gets the developer info from the OSGi Configuration  
19. * There are 4 OSGi Configuration Examples:  
20. * -Boolean  
21. * -String  
22. * -String Array  
23. * -Dropdown  
24. */  
25.  
26. @Component(service = DeveloperInfo.class, name = "TrainingDeveloperInfo")  
27.  
28. @Designate(ocd = DeveloperInfoConfiguration.class)  
29. public class DeveloperInfoImpl implements DeveloperInfo {  
30.     private final Logger logger = LoggerFactory.getLogger(getClass());  
31.  
32.     //local variables to hold OSGi config values  
33.     private boolean showDeveloper;  
34.     private String developerName;  
35.     private String[] developerHobbiesList;  
36.     private String langPreference;  
37.  
38.     @Activate  
39.     @Modified  
40.     //http://blogs.adobe.com/experiencedelivers/experience-management/osgi_activate_deactivatesignatures/  
41.     protected void activate(DeveloperInfoConfiguration config) {  
42.  
43.         showDeveloper = config.developerinfo_showinfo();  
44.         developerName = config.developerinfo_name();  
45.         developerHobbiesList = config.developerinfo_hobbies();
```

```

46.     langPreference = config.developerinfo_language();
47.     logger.info("#####Component config saved");
48. }
49.
50. @Deactivate
51. protected void deactivate() {
52.     logger.info("#####Component (Deactivated) Good-bye " + developerName);
53. }
54.
55. /**
56. * Method used to show how OSGi configurations can be brought into a OSGi component
57. */
58. public String getDeveloperInfo(){
59.
60.     String developerHobbies = Arrays.toString(developerHobbiesList);
61.
62.     if(showDeveloper)
63.
64.         return "Created by " + developerName
65.             + ". <br>Hobbies include: " + developerHobbies
66.             + ". <br>Preferred programming language in AEM is " + langPreference;
67.     return "";
68. }
69.
70. /**
71. * Method used to show a simple OSGi service/component relationship
72. public String getDeveloperInfo(){
73.     return "Hello! I do not know who my developer is. I am a product of random development!!!";
74. }
75. */
    }

```

4. Examine the modified code. Notice the name of the component at the beginning of the class **TrainingDeveloperInfo**, as shown:

```
@Component(service = DeveloperInfo.class, name = "TrainingDeveloperInfo")
```

5. Examine the Component implementation of the **DeveloperInfo** Service. This gets the developer info from the OSGi configuration.
6. Examine the various methods used in the class such as **activate()**, **deactivate()** and **getDeveloperInfo()** to understand the logic behind them.

Notice the activate method takes a special class called **DeveloperInfoConfiguration**. This is a custom configuration class where you set what properties should be exposed to OSGi through the Web Console.

7. Enter the following value for the package and name:

- Package: **com.adobe.training.core.services**
- Name: **DeveloperInfoConfiguration**

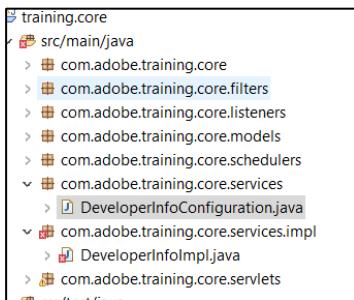
8. Click **Finish**.

9. Copy the contents of the file **DeveloperInfoConfiguration.java** from /Exercises\_Folder/08\_Deep Dive into OSGi to **DeveloperInfoConfiguration** in Eclipse.



**Note:** The code is provided as part of the Exercise\_Files under /Exercise\_Files/08\_Deep\_Dive\_OSGI/. Copy and paste the file from the exercise files referenced to **DeveloperInfoConfiguration.java** to Eclipse. Do not copy the code from this exercise book. The code given here is only for illustrative purposes.

10. Right-click **com.adobe.training.core.services**, and paste the file. The **DeveloperInfoConfiguration.java** class is created, as shown:



11. Open DeveloperInfoConfiguration.java in editor, as shown:

```
1. package com.adobe.training.core.services;
2.
3. import org.osgi.service.metatype.annotations.AttributeDefinition;
4. import org.osgi.service.metatype.annotations.ObjectClassDefinition;
5. import org.osgi.service.metatype.annotations.AttributeType;
6. import org.osgi.service.metatype.annotations.Option;
7.
8. @ObjectClassDefinition(name = "Developer Info Service")
9. public @interface DeveloperInfoConfiguration {
10.
11.     @AttributeDefinition(
12.         name = "Show Info",
13.         description = "Should the Developer information be shown?",
14.         type = AttributeType.BOOLEAN
15.     )
16.     boolean developerinfo_showinfo() default false;
17.
18.     @AttributeDefinition(
19.         name = "Name",
20.         description = "Name of the Developer",
21.         type = AttributeType.STRING
22.     )
23.     String developerinfo_name() default "";
24.
25.     @AttributeDefinition(
26.         name = "Hobbies",
27.         description = "List your favorite Hobbies",
28.         type = AttributeType.STRING
29.     )
30.     String[] developerinfo_hobbies() default {"swimming", "climbing"};
31.
```

```
32. @AttributeDefinition(  
33.     name = "Language",  
34.     description = "Favorite Language Preference",  
35.     options = {  
36.         @Option(label = "HTL", value = "HTL"),  
37.         @Option(label = "Java", value = "Java"),  
38.         @Option(label = "JSP", value = "JSP"),  
39.         @Option(label = "HTML", value = "HTML"),  
40.         @Option(label = "JavaScript", value = "JavaScript")  
41.     }  
42. )  
43. String developerinfo_language() default "";  
44. }
```

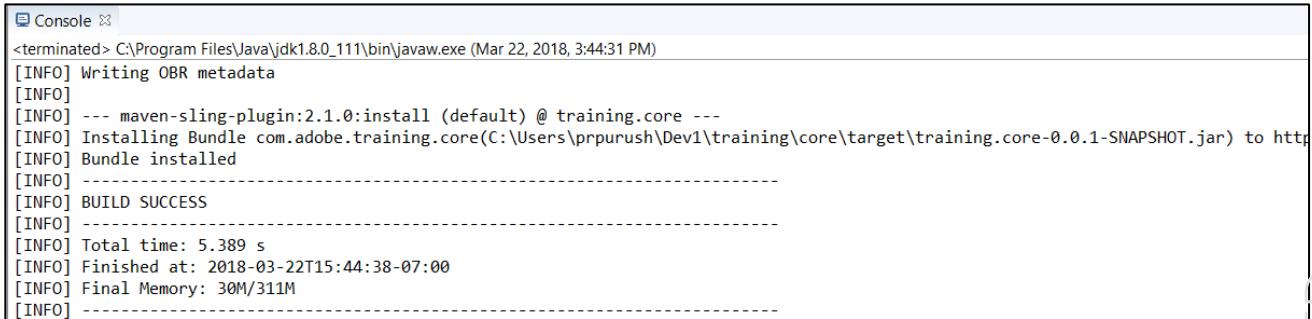
12. Save the changes.



**Note:** You get the configurations in the activate() method by using the methods defined in the DeveloperInfoConfiguration class.

## Task 2: Deploy the bundle

1. In Project Explorer, right-click **training.core**, and select **Run As > Maven install**. The build starts.
2. Verify the bundle installed successfully, as shown:



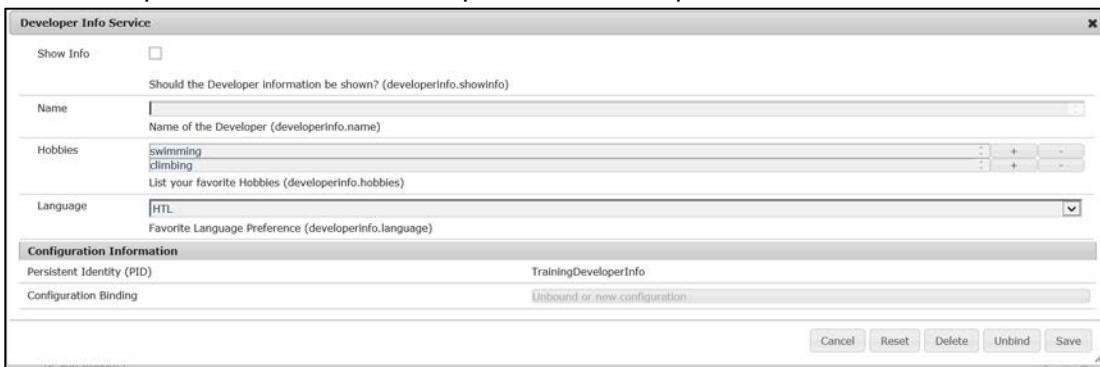
```

Console <terminated> C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Mar 22, 2018, 3:44:31 PM)
[INFO] Writing OBR metadata
[INFO]
[INFO] --- maven-sling-plugin:2.1.0:install (default) @ training.core ---
[INFO] Installing Bundle com.adobe.training.core(C:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar) to http://
[INFO] Bundle installed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.389 s
[INFO] Finished at: 2018-03-22T15:44:38-07:00
[INFO] Final Memory: 30M/311M
[INFO] -----

```

## Task 3: Test the service

1. Navigate to the Web Console in AEM (<http://localhost:4502/system/console/configMgr>). The **Adobe Experience Manager Web Console** page opens.
2. Under **OSGi > Configuration**, search for Developer Info.
3. Click Develop Info Service. The Developer Info Service opens, as shown:



4. Carry out the following steps:
  - a. Select the **Show Info** checkbox.
  - b. Enter **Scott Reynolds** in the Name field.
5. Save the changes.
6. Navigate to **Sites**. The **Sites** console opens.
7. Select **TrainingProject Site > English > Edit (e)** to open the page in a new tab in your browser.

+

8. Verify the following message, as shown:



**Note:** In this task, you updated the OSGi configurations from the Web Console. In a realistic situation, you should create an OSGi config node in the JCR.

## Reference



For more information on OSGi, refer:

<https://www.osgi.org/developer/architecture/>

<http://felix.apache.org/documentation/subprojects/apache-felix-service-component-runtime.html>

# Configuring the Sling Web Framework



## Introduction

Adobe Experience Manager (AEM) is developed by using frameworks, such as OSGi and Apache sling. OSGi defines a dynamic component that is written in Java. These specifications enable a development model where dynamic applications consists of reusable components.

## Objectives

After completing this course, you will be able to:

- Describe Apache Sling
- Explain the REST architecture
- Describe sling resolution process
- Create Sling servlets
- Configure Sling POST servlet
- Create system users

# Apache Sling

Apache sling is an open source web application framework that makes it easy to develop content-oriented applications. Apache sling:

- Is a Representational State Transfer (REST)-based web framework
- Is content-driven and using a JCR content repository
- Is powered by OSGi
- Uses scripts or Java Servlets to process HTTP requests
- Is resource-oriented and maps into JCR nodes

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

In Apache Sling, a request URL is first resolved to a resource. Based on this resource, as well as the request method and more properties of the request URL, a script or servlet is then selected to handle the request.

# RESTful Architecture

REST is an architectural style that defines a set of constraints and properties based on [HTTP](#). Web Services that conform to the REST architectural style, or **RESTful web services**, provide interoperability between computer systems on the [Internet](#). REST-compliant web services allow the requesting systems to access and customize textual representations of [web resources](#) by using a uniform and predefined set of [stateless operations](#).

Addressable resources present a uniform interface that allows transfers of state. For example, the interface allows the 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.

Four basic HTTP verbs are used in requests to interact with resources in a REST system:

- GET — retrieve a specific resource (by ID) or a collection of resources
- POST — create a new resource
- PUT — update a specific resource (by ID)
- DELETE — remove a specific resource (by ID)

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 is all you need to build a distributed application. HTTP is a rich application protocol that gives you capabilities, such as content negotiation and distributed caching. RESTful web applications try to leverage HTTP in its entirety by using specific architectural principles.

## Advantages of REST

- 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 URIs by firewall configuration, unlike other XML over HTTP messaging formats

## Understanding Sling Resolution Process

Apache Sling is resource oriented, and maintains all the 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. Following are 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*. 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 the order of the priority as follows:

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:

1. Decompose the URL
2. Mapping Request to resources
3. Resolve the Resource
4. Resolve the rendering script/servlet

### 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

After 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>

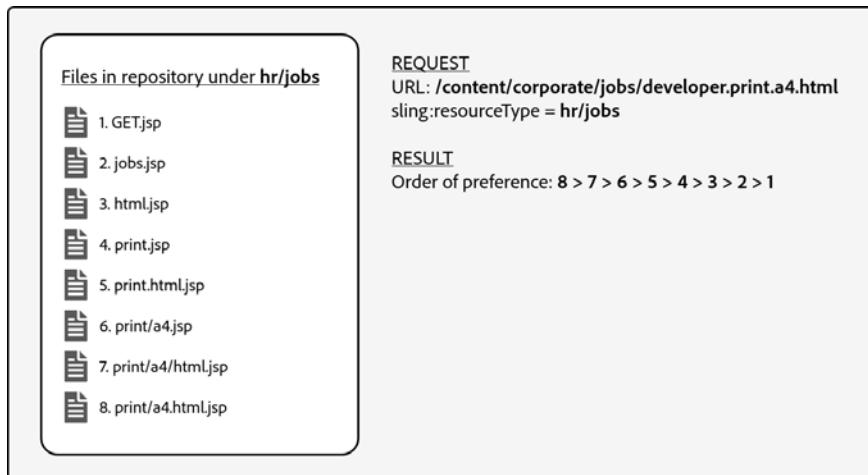
- Sling checks whether a node exists at the location specified in the request  
In the above example, it searches for the node **spy.html**.
- If no node is found at that location, the extension is dropped and the search is repeated. For example, it searches for the node **spy**.
- If no node is found, then Sling returns 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

## Resolve the 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 is 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 the selector matches, the better.

Example of rendering the content:



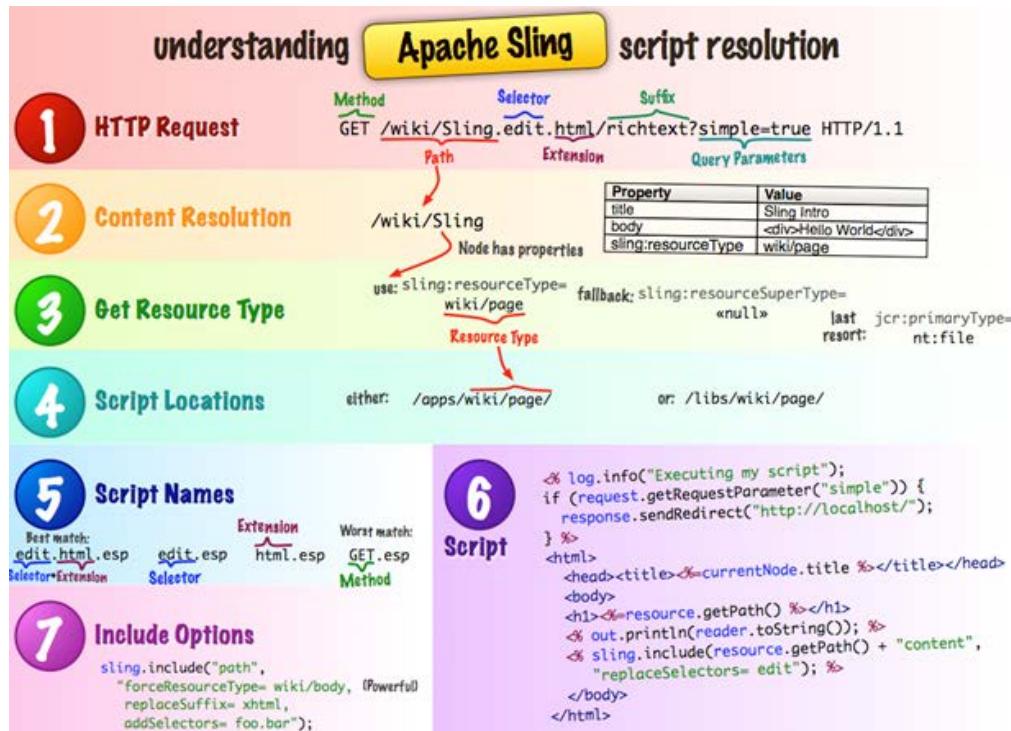
Super types are also taken into consideration when trying to locate a script. The advantage of resource super types is that they may form a hierarchy of resources where the default resource type `sling/servlet/default` (used by the default servlets) is effectively the root.



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 CRXDE Lite 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 <code>sling:alias</code> property, and may be attached to any node, which does not otherwise allow the setting of a property named <code>sling:alias</code> .
sling:MappingSpec	Mixin node <b>type defines the <code>sling:match</code>, <code>sling:redirect</code>, <code>sling:status</code>, and <code>sling:internalRedirect</code> properties</b>
sling:Mapping	The primary node type used to construct entries in <code>/etc/map</code>

The following properties are important when dealing with new resources.

Properties	Description
<code>sling:match</code>	Defines a partial regular expression used on a node's name to match the incoming request.
<code>sling:redirect</code>	Causes a redirect response to be sent to the client
<code>sling:status</code>	Defines the HTTP status code sent to the client
<code>sling:internalredirect</code>	Causes the current path to be modified internally to continue with resource resolution
<code>sling:alias</code>	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, the following are the mappings:

Regular Expression	Redirect	Internal	Description
http/example.com.80	http://www.example.com	no	Redirects all requests to the Second Level Domain to www
http/www.example.com.80	/example	yes	Prefixes the URI paths of the requests sent to this domain with the string /example
http/.+example.com.80	http://www.example.com	no	Redirects 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	Prefixes 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	Replaces 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	Replaces 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	Prepends the URI paths starting with /stories with /anecdotes for requests to localhost, regardless of actual port the request was received on.

## Working with Sling Servlets

Servlets can be registered as OSGi services. Apache 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, Apache Sling is oriented around resources, which usually map to JCR nodes. With Apache 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, Apache 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 JCR) directly from HTML forms or from the command line by using cURL.

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

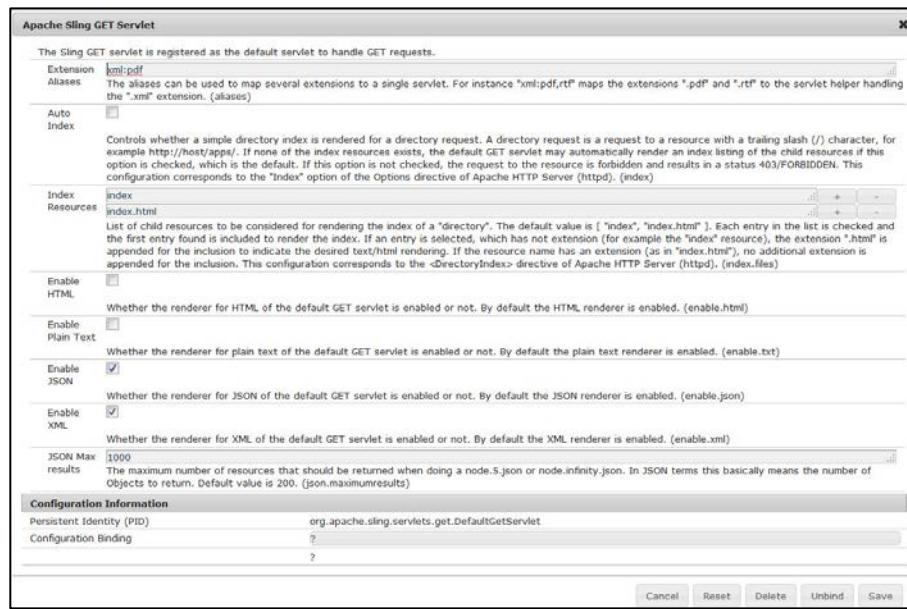
<http://localhost:4502/system/console/configMgr>

## Configuring the Default Sling GET Servlet

Apache Sling provides the 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, as shown:



## Configuring the Sling POST Servlet

The Sling POST Servlet provides a RESTful interface that enables you to customize the data content stored in the AEM JCR. This servlet maps nodes in the JCR repository to URLs and enables applications to modify the JCR content. A Sling POST Servlet is performed on the client by 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="title" value="" />
4. </form>
```



**Note:** This code will work only if the **com.adobe.granite.csrf.impl.CSRFFilter** filter is defined in OSGI configurations.

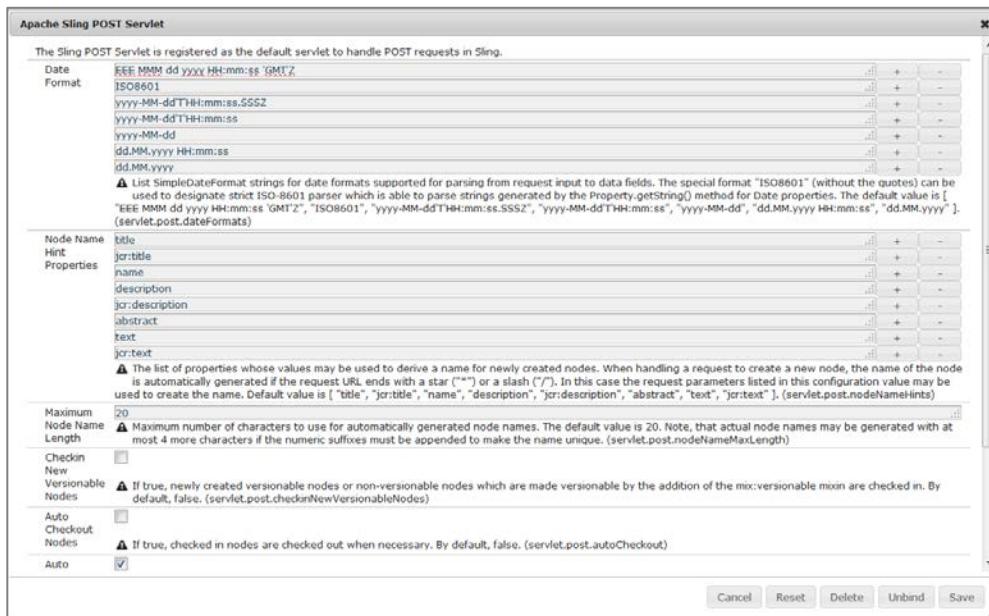


The above code will set the title and text properties on a node in the location /mycontent. If this node does not exist, it will be created. Otherwise, the existing content at that URL would be modified.

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, as shown:



## Exercise 1: Create a Sling servlet

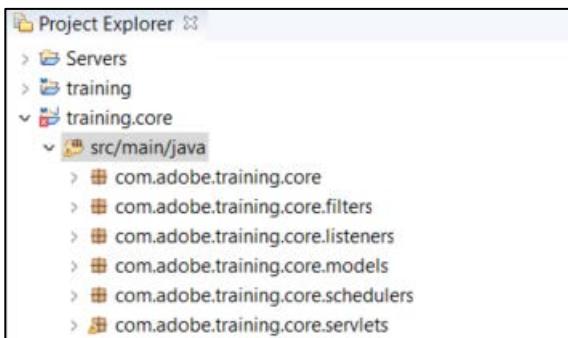
In this exercise, you will write a servlet that serves only DOGET requests from a specific URI or resource type. The response shall contain the JCR repository properties as JSON.

This exercise includes two tasks:

1. Create a servlet with a path
2. Create a servlet with a resource type

### Task 1: Create a servlet with a path

1. Launch **Eclipse** by double-clicking the **Eclipse** shortcut on the desktop. The Eclipse Launcher opens.
2. Specify the Workspace as **C:\Workspace** and click **Launch**. The Workspace opens.
3. In Project Explorer, navigate to **training.core > src/main/java**, as shown:

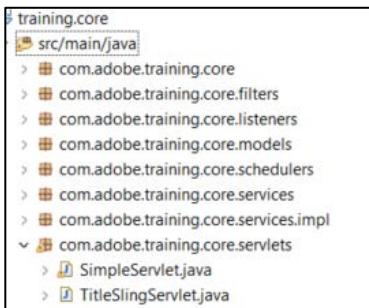


4. Copy the **TitleSlingServlet.java** file from **/Exercises\_Folder/09\_Sling\_Web\_Framework**.



**Note:** The code is provided as part of the Exercise\_Files under **/Exercise\_Files/09\_Sling\_Web\_Framework/**. Copy and paste the file from the exercise files referenced to **TitleSlingServlet.java** to Eclipse. Please do not copy the code from this exercise book. The code in the student guide is only for illustrative purposes.

5. Right-click **com.adobe.training.core.servlets** and paste the file. The **TitleSlingServlet.java** class is created, as shown:



6. Observe that the servlet **Path** mentioned in the code is **/bin/trainingproject/titleServlet**. So the request can be served on this path, as shown:

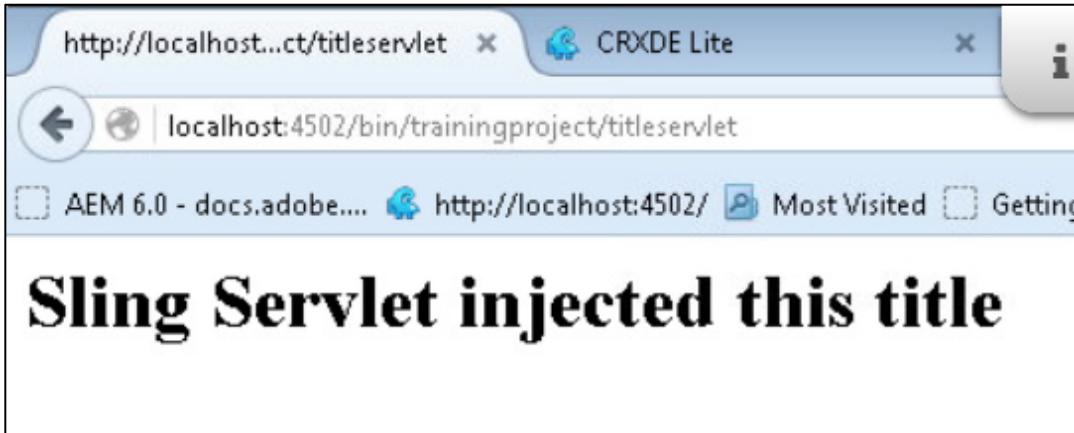
```
1. package com.adobe.training.core.servlets;
2.
3. import java.io.IOException;
4.
5. import javax.servlet.Servlet;
6. import javax.servlet.ServletException;
7. import org.apache.sling.api.SlingHttpServletRequest;
8. import org.apache.sling.api.SlingHttpServletResponse;
9. import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
10.
11. import org.osgi.service.component.annotations.Component;
12.
13. @Component( service = Servlet.class,
14.             name="TrainingTitleServlet",
15.             property = {
16.                 "sling.servlet.paths=/bin/trainingproject/titleServlet",
17.                 "sling.servlet.extensions=html"
18.             })
19.
20. //TitleSlingServlet uses resourceTypes and extensions to bind to URLs
21.
22. /*@Component( service = Servlet.class,
23.             name="TrainingTitleServlet",
24.             property = {
25.                 "sling.servlet.resourceTypes=/apps/trainingproject/components/content/title",
26.
27.                     "sling.servlet.extensions=html"
28.             })*/
29.
30. public class TitleSlingServlet extends SlingSafeMethodsServlet {
31.
32.     private static final long serialVersionUID = 1L;
33.
34.     @Override
35.     protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response)
36.             throws ServletException, IOException {
37.         response.setHeader("Content-Type", "text/html");
38.         response.getWriter().print("<h1>Sling Servlet injected this title</h1>");
39.         response.getWriter().close();
40.     }
41. }
```

7. In **Project Explorer**, right-click **training.core** and select **Run As > Maven install**. The build starts.

8. Verify the bundle installed successfully, as shown:

```
[INFO] Installing C:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar to C:\Users\prpurush\.m2\repository\com\adobe\t
[INFO] Installing C:\Users\prpurush\Dev1\training\core\pom.xml to C:\Users\prpurush\.m2\repository\com\adobe\training.core\0.0.1-SNAPSHOT\trai
[INFO]
[INFO] --- maven-bundle-plugin:3.3.0: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:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar) to http://local
[INFO] Bundle installed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.366 s
[INFO] Finished at: 2018-03-29T16:10:08-07:00
[INFO] Final Memory: 31M/360M
[INFO] -----
```

9. Verify you can call the title servlet through <http://localhost:4502/bin/trainingproject/titleServlet>, as shown:



## Task 2: Create a servlet with a resource type

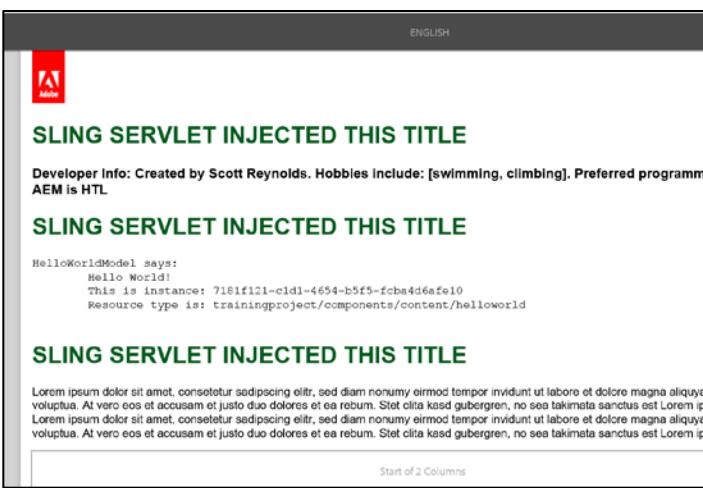
1. In **Project Explorer**, edit **TitleSlingServlet** to change the servlet implementation by commenting the following as shown:

```
1. /* @Component( service = Servlet.class,
2.     name="TrainingTitleServlet",
3.     property = {
4.         "sling.servlet.paths=/bin/trainingproject/titleServlet",
5.         "sling.servlet.extensions=html"
6.     })*/
```

2. Uncomment the following lines, as shown:

```
3. @Component( service = Servlet.class,
4.             name="TrainingTitleServlet",
5.             property = {
6.                 "sling.servlet.resourceTypes=/apps/trainingproject/components/content/title",
7.                 "sling.servlet.extensions=html"
8.             })
```

3. Right-click **training.core**, and select **Run As > Maven install** to build the project. The project runs successfully.
4. Open <http://localhost:4502/content/trainingproject/en.html> and see the output, as shown:



5. Notice how the servlet finds all the resources on the page by using **/apps/trainingproject/components/content/title** and injects the response from the servlet.

# 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.

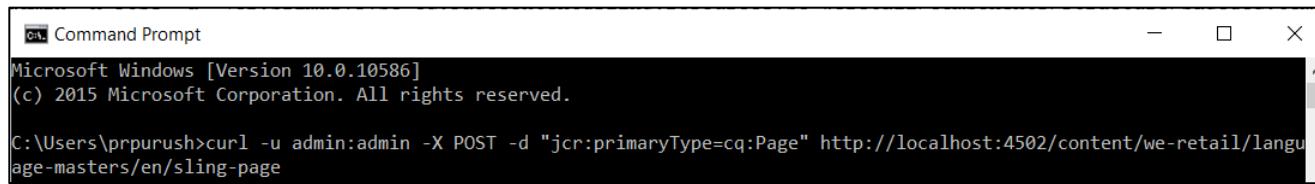
## Exercise 2: Add content with Sling post servlet

In this exercise, you will use the Sling Post Servlet to create and customize a page. The commands will help achieve the following:

1. Create a Page
2. Lock a Page
3. Add content to a page
4. Unlock the page
5. Publish the page

1. On the OS command line, execute the following command to create a page, as shown:

```
curl -u admin:admin -X POST -d "jcr:primaryType=cq:Page" http://localhost:4502/content/we-retail/language-masters/en/sling-page
```

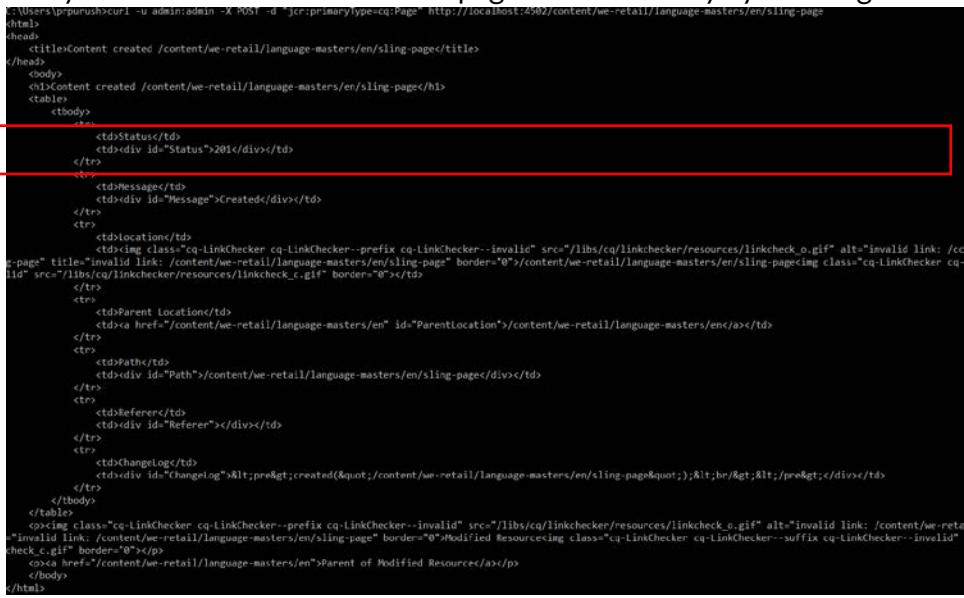


The screenshot shows a Windows Command Prompt window. The title bar says "Command Prompt". The window content shows the following text:  
Microsoft Windows [Version 10.0.10586]  
(c) 2015 Microsoft Corporation. All rights reserved.  
  
C:\Users\ppurush>curl -u admin:admin -X POST -d "jcr:primaryType=cq:Page" http://localhost:4502/content/we-retail/language-masters/en/sling-page



**Note:** The commands are available as part of /Exercises\_Folder/09\_Sling\_Web\_Framework/sling-curl-commands.txt

2. Verify that the command to create a page ran successfully by checking for the status Created, as shown:



The screenshot shows a browser console or developer tools interface. It displays the HTML response from the curl command. A red box highlights the "Status" column in a table where the value is "Created".  
  
The HTML code includes a table with several rows and columns, including "Status", "Message", "Location", "Path", "Referer", and "ChangeLog".  
  
Some parts of the code are heavily redacted with "REDACTED".

3. Run the next command to add content to the sling page, as shown:

```
curl -u admin:admin -X POST -d "jcr:primaryType=cq:PageContent&sling:resourceType=we-retail/components/structure/page&cq:template=/conf/we-retail/settings/wcm/templates/content-page&jcr:title=Sling cURL Page" http://localhost:4502/content/we-retail/language-masters/en/sling-page/jcr:content
```

```
<User><prp:push><url u_admin:admin -X POST -J <jcr-primary:type=cq>/pageContent& sling:resourceType=we-retail/components/structure/page/cq:template>-con/we-retail/settings/wcm/templates/content-page&crfittle-5
ling cURL Page <http://localhost:4502/content/we-retail/language-masters/en/sling-page/jcr:content
<html>
<head>
    <title>Content created /content/we-retail/language-masters/en/sling-page/jcr:content</title>
</head>
<body>
    <childContent created="/content/we-retail/language-masters/en/sling-page/jcr:content">
        <table>
            <tbody>
                <tr>
                    <td>Status</td>
                    <td><div id="Status">201</div></td>
                </tr>
                <tr>
                    <td>Message</td>
                    <td><div id="Message">Created</div></td>
                </tr>
                <tr>
                    <td>Locations</td>
                    <td><a href="/content/we-retail/language-masters/en/sling-page/_jcr_content" id="Location">/content/we-retail/language-masters/en/sling-page/_jcr_content</a></td>
                </tr>
                <tr>
                    <td>Parent Location</td>
                    <td><a href="/content/we-retail/language-masters/en/sling-page" id="Parentlocation">/content/we-retail/language-masters/en/sling-page</a></td>
                </tr>
                <tr>
                    <td>Path</td>
                    <td><div id="Path">/content/we-retail/language-masters/en/sling-page/jcr:content</div></td>
                </tr>
                <tr>
                    <td>Referer</td>
                    <td><div id="Referer"></div></td>
                </tr>
                <tr>
                    <td>Change Log</td>
                    <td><div id="ChangeLog">&lt;pre&gt;created("/content/we-retail/language-masters/en/sling-page/jcr:content");&lt;br/&gt;modified("/content/we-retail/language-masters/en/sling-page/jcr:content");&lt;br/&gt;resourceType="cq:template");&lt;br/&gt;modified("/content/we-retail/language-masters/en/sling-page/jcr:content/cq:template");&lt;br/&gt;modified("/content/we-retail/language-masters/en/sling-page/jcr:content/cq:lastModified");&lt;br/&gt;modified("/content/we-retail/language-masters/en/sling-page/jcr:content/cq:lastModifiedBy");&lt;br/&gt;&lt;pre&gt;</div></td>
                </tr>
            </tbody>
        </table>
        <p><a href="/content/we-retail/language-masters/en/sling-page/_jcr_content">Modified Resource</a></p>
        <p><a href="/content/we-retail/language-masters/en/sling-page/_parent">Parent of Modified Resource</a></p>
    </childContent>
</body>
</html>
```

4. Navigate to **Sites > We.Retail > Language Masters > English**, and notice the content is added to the **Sling Page**, as shown:

Screenshot of the AEM Experience Manager interface showing the navigation tree and a modal dialog.

The navigation tree on the left lists various language and regional configurations:

- Language Masters (language-masters)
- United States (us)
- Canada (ca)
- Switzerland (ch)
- Germany (de)
- France (fr)
- Spain (es)
- Italy (it)

Each configuration has a dropdown arrow icon next to it. The "English" configuration is currently selected, indicated by a blue background and a checkmark icon at the bottom right of its row.

The main content area shows the "English" configuration details:

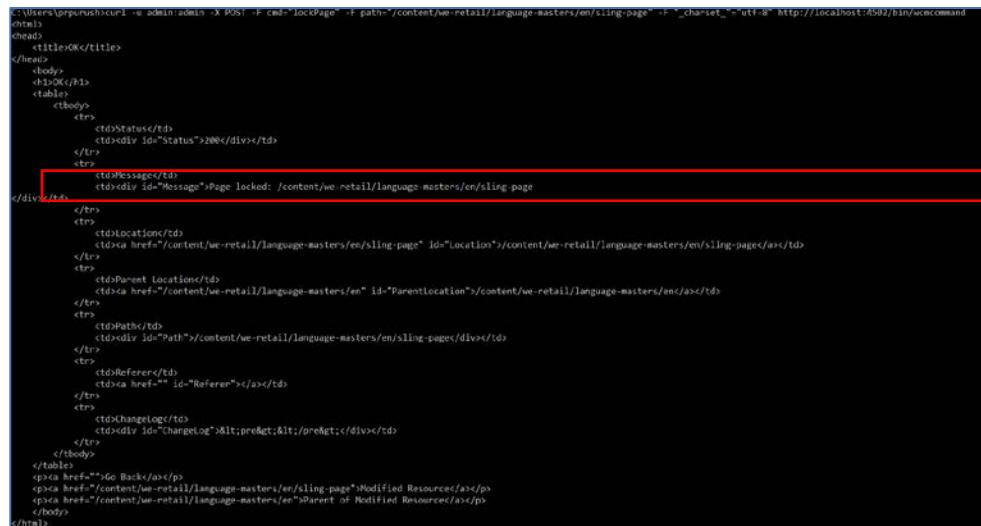
Page Title	Sling cURL Page
Name	sling-page
Template	Content Page
Modified	a few seconds ago
Modified By	Administrator
Language	English
Published	Not published

A red box highlights the "Sling cURL Page" field, and another red box highlights the checkmark icon at the bottom right of the English configuration row.

5. Run the command to lock the page, as shown:

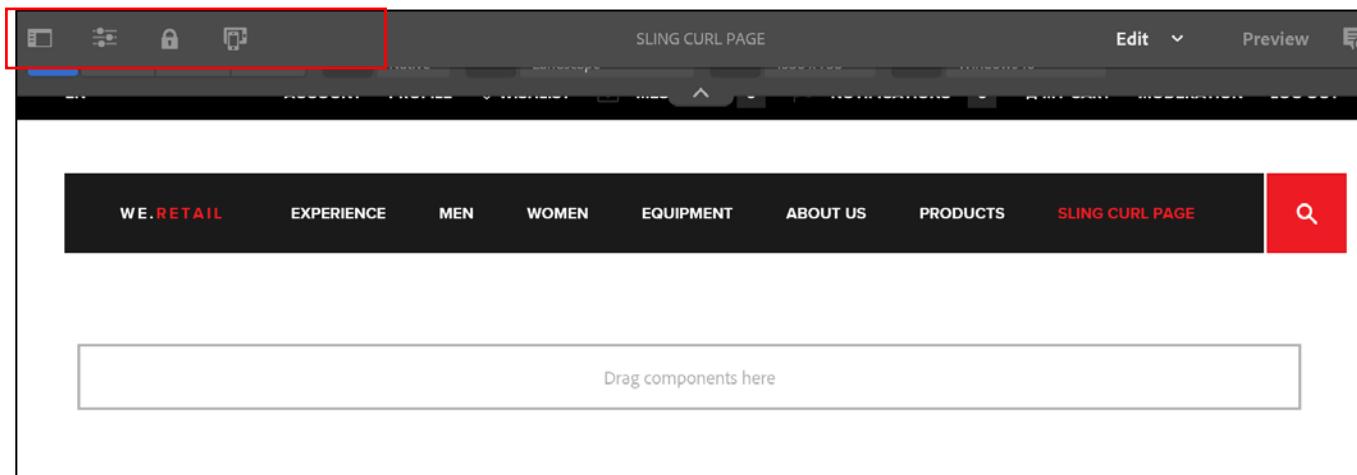
```
curl -u admin:admin -X POST -F cmd="lockPage" -F path="/content/we-retail/language-masters/en/sling-page" -F "_charset_"]="utf-8" http://localhost:4502/bin/wcmcommand
```

6. Verify that the command to lock the page ran successfully by checking for the status **Page locked**, as shown:



```
C:\Users\prpuru\curl -u admin:admin -X POST -F cmd="lockPage" -F path="/content/we-retail/language-masters/en/sling-page" -F "_charset_"]="utf-8" http://localhost:4502/bin/wcmcommand
html>
<head>
<title>OK</title>
</head>
<body>
<h1>OK</h1>
<table>
<tbody>
<tr>
<td>Status</td>
<td><div id="Status">200</div></td>
</tr>
<tr>
<td>Message</td>
<td><div id="Message">Page locked: /content/we-retail/language-masters/en/sling-page</div></td>
</tr>
<tr>
<td>Location</td>
<td><a href="/content/we-retail/language-masters/en/sling-page" id="Location">/content/we-retail/language-masters/en/sling-page</a></td>
</tr>
<tr>
<td>Parent Location</td>
<td><a href="/content/we-retail/language-masters/en" id="Parentlocation">/content/we-retail/language-masters/en</a></td>
</tr>
<tr>
<td>Path</td>
<td><div id="Path">/content/we-retail/language-masters/en/sling-pages</div></td>
</tr>
<tr>
<td>Referer</td>
<td><a href="#" id="Referer"></a></td>
</tr>
<tr>
<td>ChangeLog</td>
<td><div id="Changelog">&lt;pre&gt;&lt;/pre&gt;</div></td>
</tr>
</tbody>
</table>
</body>
</html>
```

7. Navigate to Sites > We.Retail > Language Masters > English, and click the thumbnail on the Sling cURL page.
8. Click Edit (e). The Sling cURL page opens in a new tab.
9. Notice the page is locked, as shown:



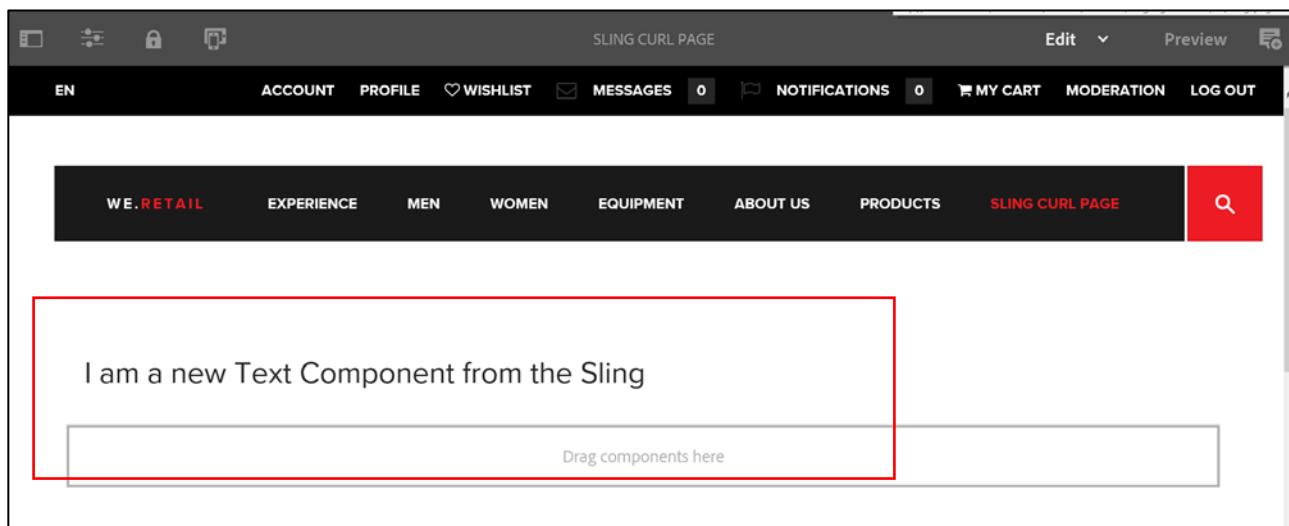
10. Run the command to add content to the page, as shown:

```
curl -u admin:admin -X POST -d "sling:resourceType=core/wcm/components/text/v2/text&text=<h3>I am a new Text Component from the Sling</h3>&textIsRich=true" http://localhost:4502/content/we-retail/language-masters/en/sling-page/jcr:content/root/responsivegrid/*
```

11. Verify that the command to add content to the page ran successfully by checking for the status **i\_am\_a\_new\_text**, as shown:

```
:[User]\rpurush>curl -u admin:admin -X POST -d "sling:resourceType=core/wcm/components/text/v2/text&text=<h3>I am a new Text Component from the Sling</h3>&textIsRich=true" http://localhost:4502/content/we-retail/language-masters/en/sling-page/jcr:content/root/responsivegrid/*
<title>Content created /content/we-retail/language-masters/en/sling-page/jcr:content/root/responsivegrid/_h3_i_am_a_new_text</title>
</head>
<body>
<h1>Content created /content/we-retail/language-masters/en/sling-page/jcr:content/root/responsivegrid/_h3_i_am_a_new_text</h1>
<table>
<tbody>
<tr>
<td>Status</td>
<td><div id="Status">201</div></td>
</tr>
<tr>
<td>Message</td>
<td><div id="Message">Created</div></td>
</tr>
<tr>
<td>Location</td>
<td><a href="/content/we-retail/language-masters/en/sling-page/_jcr_content/root/responsivegrid/_h3_i_am_a_new_text_" id="location">/content/we-retail/language-masters/en/sling-page/_jcr_content/root/responsivegrid/_h3_i_am_a_new_text</a></td>
</tr>
<tr>
<td>Parent Location</td>
<td><a href="/content/we-retail/language-masters/en/sling-page/_jcr_content/root/responsivegrid" id="parentlocation">/content/we-retail/language-masters/en/sling-page/_jcr_content/root/responsivegrid</a></td>
</tr>
<tr>
<td>Path</td>
<td><div id="Path">/content/we-retail/language-masters/en/sling-page/jcr:content/root/responsivegrid/_h3_i_am_a_new_text</div></td>
</tr>
<tr>
<td>Referrer</td>
<td><div id="referrer"></div></td>
</tr>
<tr>
<td>Changelog</td>
<td><div id="changelog">&lt;pre&gt;created("/content/we-retail/language-masters/en/sling-page/jcr:content/root");&lt;br&gt;created("/content/we-retail/language-masters/en/sling-page/jcr:content/root/responsivegrid/_h3_i_am_a_new_text");&lt;br&gt;modified("/content/we-retail/language-masters/en/sling-page/jcr:content/root/responsivegrid/_h3_i_am_a_new_text/_sling:resourceType");&lt;br&gt;modified("/content/we-retail/language-masters/en/sling-page/jcr:content/root/responsivegrid/_h3_i_am_a_new_text/_text");&lt;br&gt;modified("/content/we-retail/language-masters/en/sling-page/jcr:content/cq:lastModified");&lt;br&gt;modified("/content/we-retail/language-masters/en/sling-page/jcr:content/cq:lastModifiedBy");&lt;br&gt;&lt;/pre&gt;</div></td>
</tr>
</tbody>
</table>
```

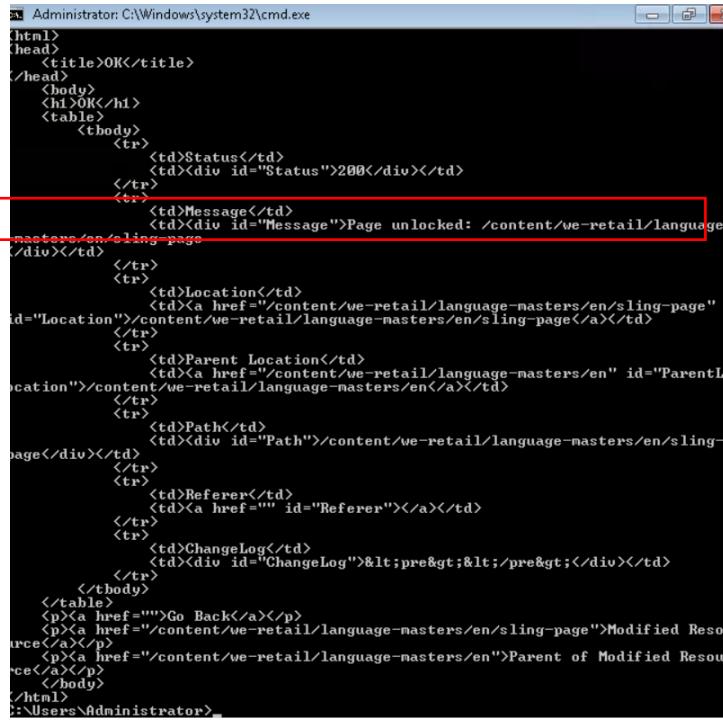
12. Go to <http://localhost:4502/editor.html/content/we-retail/language-masters/en/sling-page.html>, and verify the content is added to the page, as shown:



13. Run the command to unlock the page, as shown:

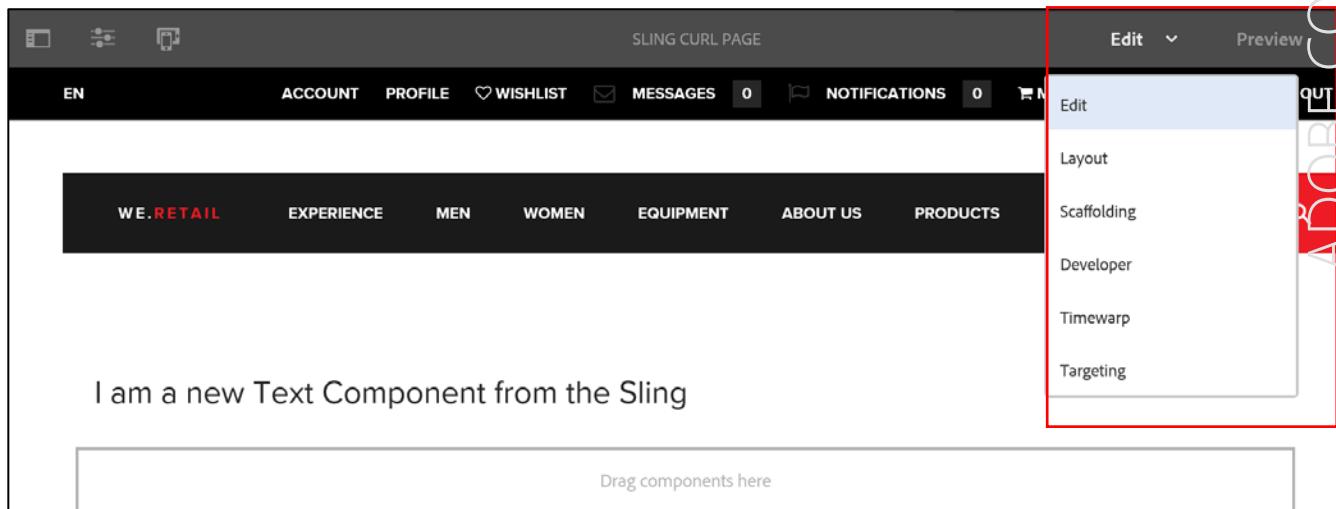
```
curl -u admin:admin -X POST -F cmd="unlockPage" -F path="/content/we-retail/language-masters/en/sling-page" -F _charset_="utf-8" http://localhost:4502/bin/wcmcommand
```

14. Verify that the command to unlock the page ran successfully by checking for the status, **Page unlocked**, as shown:



```
<html>
<head>
<title>OK</title>
</head>
<body>
<h1>OK</h1>
<table>
<tbody>
<tr>
<td>Status</td>
<td><div id="Status">200</div></td>
</tr>
<tr>
<td>Message</td>
<td><div id="Message">Page unlocked: /content/we-retail/language-masters/en/sling-page</div></td>
</tr>
<tr>
<td>Location</td>
<td><a href="/content/we-retail/language-masters/en/sling-page" id="Location">/content/we-retail/language-masters/en/sling-page</a></td>
</tr>
<tr>
<td>Parent Location</td>
<td><a href="/content/we-retail/language-masters/en" id="ParentLocation">/content/we-retail/language-masters/en</a></td>
</tr>
<tr>
<td>Path</td>
<td><div id="Path">/content/we-retail/language-masters/en/sling-page</div></td>
</tr>
<tr>
<td>Referer</td>
<td><a href="" id="Referer"></a></td>
</tr>
<tr>
<td>ChangeLog</td>
<td><div id="ChangeLog">&lt;pre&gt;&lt;/pre&gt;</div></td>
</tr>
</tbody>
</table>
<p><a href="">Go Back</a></p>
<p><a href="/content/we-retail/language-masters/en/sling-page">Modified Resource</a></p>
<p><a href="/content/we-retail/language-masters/en">Parent of Modified Resource</a></p>
</body>
</html>
:C:\Users\Administrator_
```

15. Go to <http://localhost:4502/editor.html/content/we-retail/language-masters/en/sling-page.html>, and verify the page is unlocked for editing, as shown:



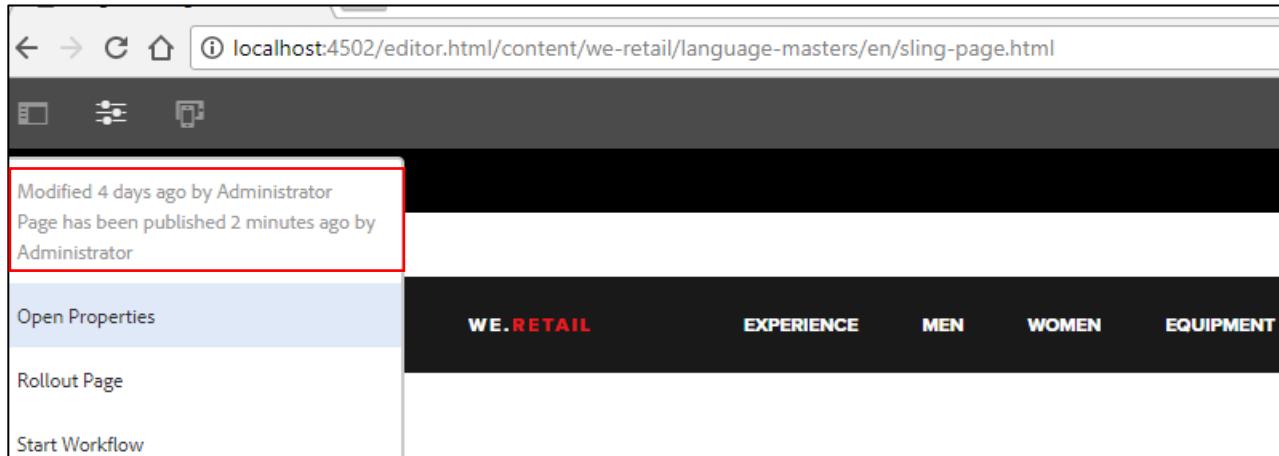
16. Run the command to activate the page, as shown:

```
curl -u admin:admin -X POST -F path="/content/we-retail/language-masters/en/sling-page" -F cmd="activate" http://localhost:4502/bin/replicate.json
```

17. Verify that the command to activate the page ran successfully by checking for the status, **Replication Started**, as shown:

```
C:\Users\prpurush>curl -u admin:admin -X POST -F path="/content/we-retail/language-masters/en/sling-page" -F cmd="activate" http://localhost:4502/bin/replicate.json
<html>
<head>
</head>
<body>
<h1></h1>
<table>
<tbody>
<tr>
<td>Status</td>
<td><div id="Status">200</div></td>
</tr>
<tr>
<td>Message</td>
<td><div id="Message">Replication started for /content/we-retail/language-masters/en/sling-page</div></td>
</tr>
<tr>
<td>Location</td>
<td><a href="" id="Location"></a></td>
</tr>
<tr>
<td>Parent Location</td>
<td><a href="" id="ParentLocation"></a></td>
</tr>
<tr>
<td>Path</td>
<td><div id="Path">/content/we-retail/language-masters/en/sling-page</div></td>
</tr>
<tr>
<td>Referer</td>
<td><a href="" id="Referer"></a></td>
</tr>
<tr>
<td>ChangeLog</td>
<td><div id="ChangeLog">&lt;pre&gt;&lt;/pre&gt;</div></td>
</tr>
</tbody>
</table>
<p><a href="">Go Back</a></p>
<p><a href="">Modified Resource</a></p>
<p><a href="">Parent of Modified Resource</a></p>
</body>
</html>
```

18. Go to <http://localhost:4502/editor.html/content/we-retail/language-masters/en/sling-page.html>, and verify the page is activated.



# Creating System Users

Adobe suggests developers to create service users and map them to Service User Mapper. A service user is a JCR user with no password set and a minimal set of privileges that are necessary to perform a specific task. Having no password set means that it will not be possible to log in with a service user.

## Problem

To access the data storage in the Resource Tree and/or JCR, 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 a good practice.

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 that are specially configured for service level access (usually done on unix systems).
- Allow administrators to configure the assignment of service users to services.

## Concept of Service

A service is a piece or collection of functionality. The examples of services are the Sling queuing system, Tenant Administration, or a Message Transfer System. Each service is identified by a unique service name. Because a service is 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. The examples of subservice name are the names for subsystems in a Message Transfer System, such as accepting messages, queueing messages, and delivering messages.

The combination of the *Service Name* and *Subservice Name* defines the *Service ID*. The *Service ID* is finally mapped to a Resource Resolver and/or JCR user ID for authentication.

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

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

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

### Implementation of Service Authentication:

The implementation in Apache sling of the service authentication concept described above consists of three parts: ServiceUserMapper, ResourceResolverFactory, and SlingRepository.

### ServiceUserMapper

The ServiceUserMapper service allows for the 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:

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

## ResourceResolverFactory

The second part is support for service access to the Resource Tree. The `ResourceResolverFactory` service is enhanced with a new factory method, as shown below:

```
ResourceResolver getServiceResourceResolver(Map<String, Object> authenticationInfo) throws 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.

## SlingRepository

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

```
Session loginService(String subServiceName, String workspace) throws LoginException, 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 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.

## Exercise 3: Create a system user

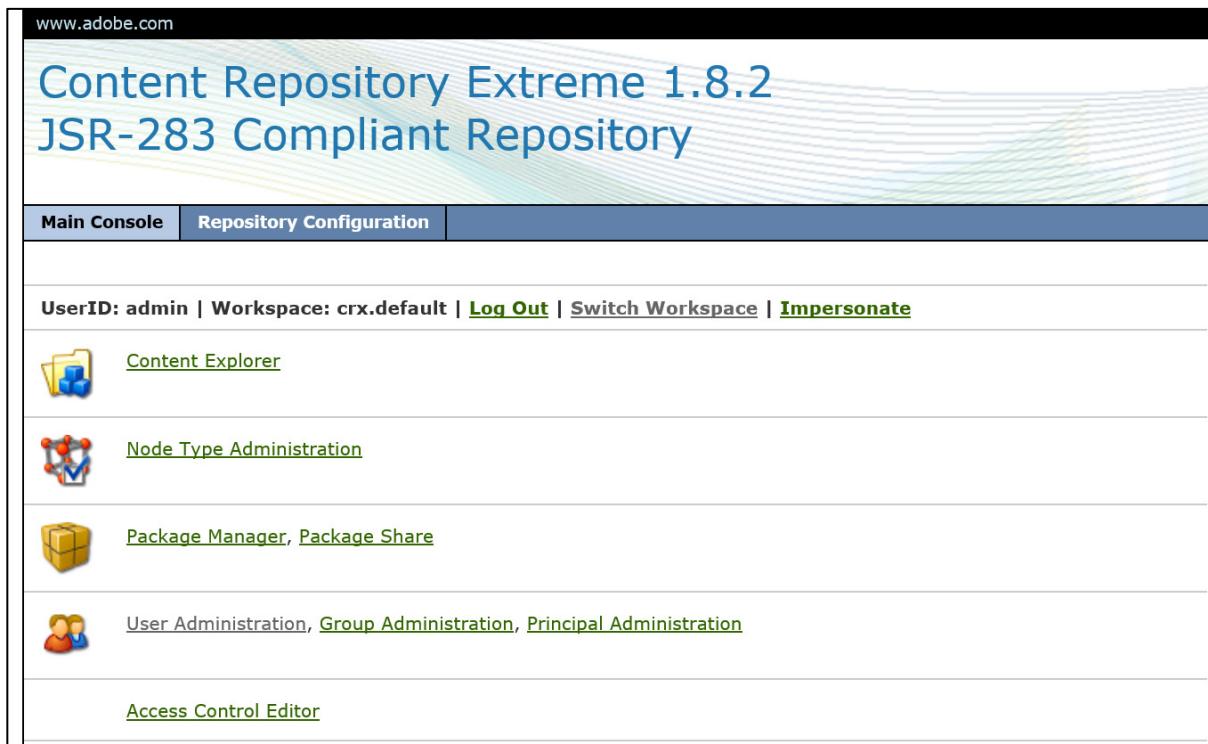
In this exercise, you will create a system user, provide access rights to the user, and test the executing tasks as the user.

This exercise includes two tasks:

1. Create a service user
2. Create the mapping to the user

### Task 1: Create a service user

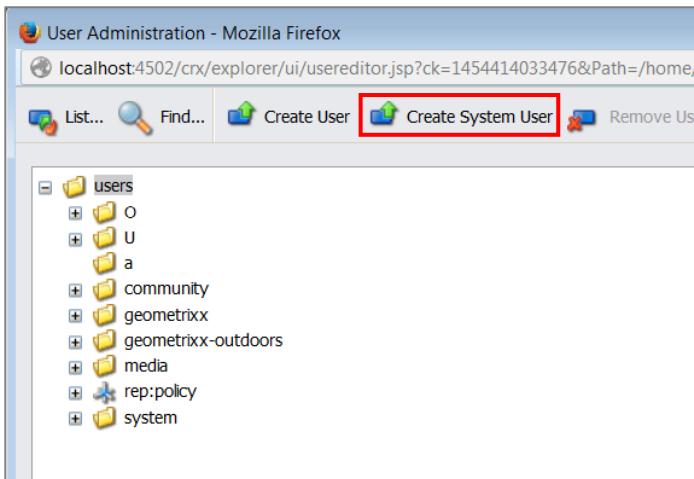
1. Navigate to: <http://localhost:4502/crx/explorer>.
2. Log in as admin/admin. The Content Repository console opens, as shown:



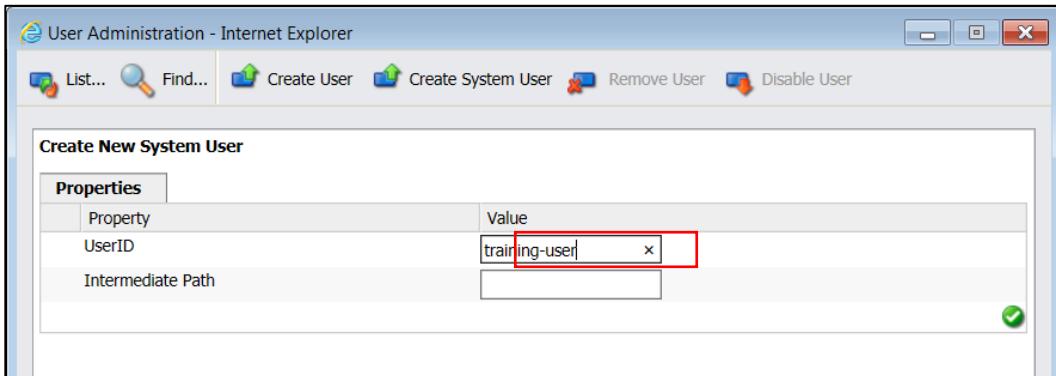
3. Click **User Administration**. The User Administration window opens.

+

4. In the window, click **Create System User**, as shown. The **Create New System User** wizard opens.

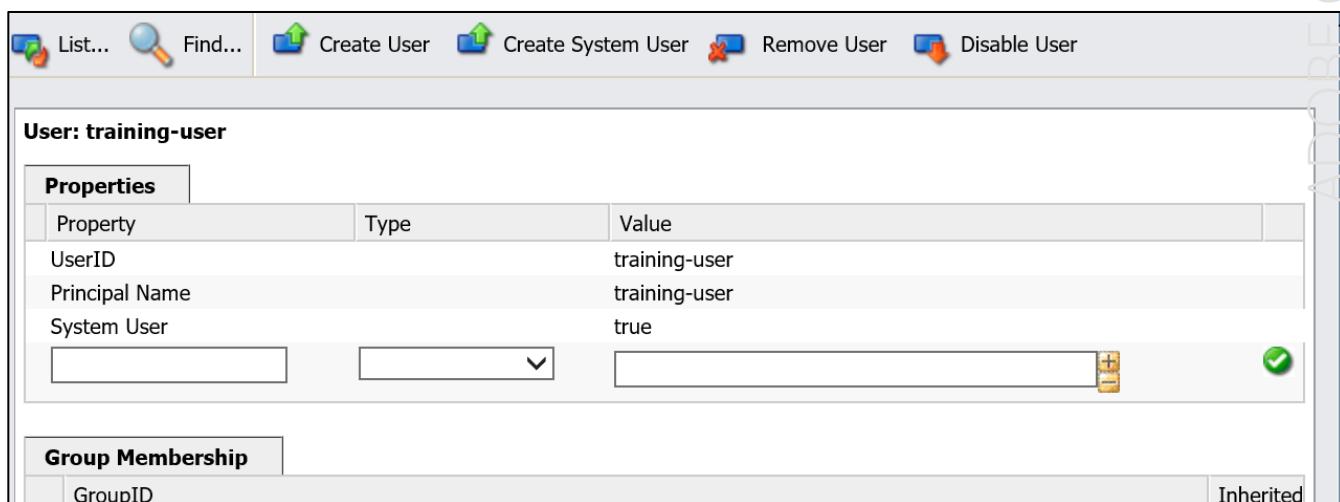


5. In the UserID field, enter **training-user**, as shown:



6. Click the green checkmark in the lower-right corner of the dialog box.

7. Click **Close**. The **training-user** is created.



8. Go to <http://localhost:4502/useradmin>. The **AEM Security** console opens, as shown:

The screenshot shows the AEM Security interface. At the top, there's a toolbar with icons for search, hide users/groups, edit, and other functions. Below the toolbar is a table listing various users and groups. The columns are labeled T..., ID, Name, Pub., and Mod. The table contains entries such as 'account-manager', 'activity-service', 'activitypurgesrv', 'admin', 'administrators', 'af-template-script-writers', 'analytics-administrators', 'analytics-content-updater...', 'analyticsservice', 'anonymous', 'assetdownloadservice', 'assetlinkshareservice', 'assetusagetrackeruser', 'async-jobs-service', 'audit-service', and 'authentication-service'. To the right of the table, there are tabs for Properties, Groups, Members, Permissions, Impersonators, and Preferences. The 'Properties' tab is currently selected.

9. Find the **administrators** group and double-click it to open, as shown:

The screenshot shows the 'Properties' tab for the 'administrators' group. The tab bar also includes Groups, Members, Permissions, Impersonators, and Preferences. The form fields include: ID\* (administrators), Name (empty), Mail (empty), About (empty), and Path (/home/groups/a/administrators). There is a 'Save' button at the top left and a 'Help' link at the top right.

10. Click the **Members** tab. The **Members** tab opens.

+

11. On the left, search for **training-user** by using the search option, as shown:

AEM - Security

training-user

T...	ID	Name	Pub.	Mod.
		training-user		

administrators

Properties Groups **Members** Permissions Impersonators Preferences

Name  
replication-receiver

12. Drag the **training-user** into the **Members** tab. The **training-user** is added to the **Members** list, as shown:

administrators

Properties Groups **Members** Permissions Impersonators Preferences

Save | Remove

Name

replication-receiver

training-user

13. Click **Save**, as shown:

Edit

administrators

Properties Groups **Members** Permissions Impersonators

Save | Remove

Name

replication-receiver

training-user

+

## Task 2: Create mapping from the bundle to the user

In this task, you will create the mapping from our bundle to the user with an OSGi configuration node. The configuration we are going to setup is the ServiceUserMapperImpl.

1. Navigate to CRXDE Lite: <http://localhost:4502/crx/de>
2. Under /apps/trainingproject/config, right-click config and choose Create >Create node.
  - Name: **org.apache.sling.serviceusermapping.impl.ServiceUserMapperImpl.amended-training**
  - Type: **sling:OsgiConfig**
3. Click OK and save the changes. The node **org.apache.sling.serviceusermapping.impl.ServiceUserMapperImpl.amended-training** is created.
4. 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:

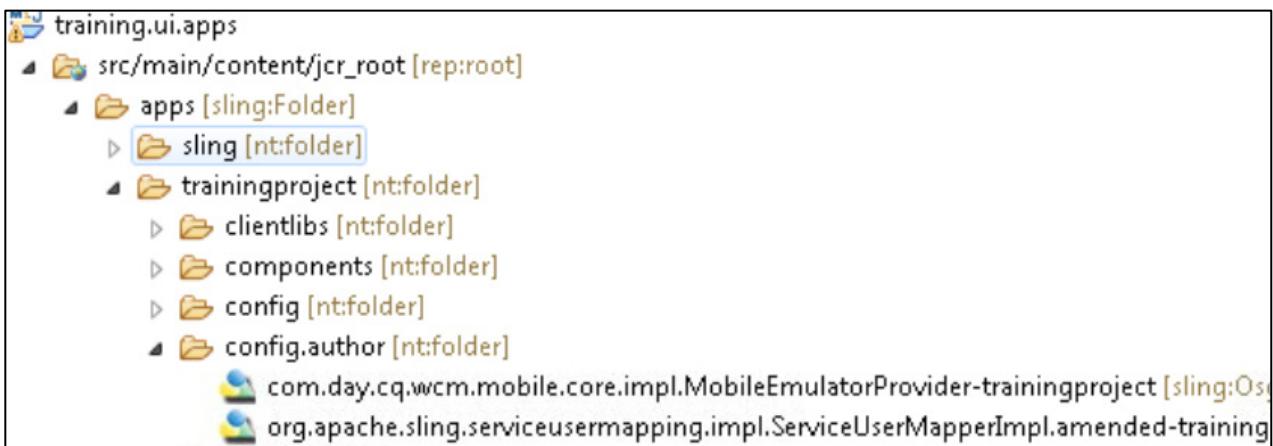
Bundle: **com.adobe.training.core**, Subservice name: **training** and Service User: **training-user**

The screenshot shows the CRXDE Lite interface. On the left, the file tree displays a folder structure under /apps/trainingproject, including clientlibs, components, config, config.author, i18n, install, templates, tests, and wcm. The 'config.author' folder is expanded, showing two sub-nodes: 'com.day.cq.wcm.mobile.core.impl.MobileEmulatorProvider-trainingproject' and 'org.apache.sling.serviceusermapping.impl.ServiceUserMapperImpl.amend'. The 'ServiceUserMapperImpl.amend' node is selected. On the right, the properties panel is open with tabs for Properties, Access Control, Replication, Console, and Build Info. The 'Properties' tab is active, showing five properties in a table:

Name	Type	Value
1 jcr:created	Date	2018-04-20T20:45:08.281+08:00
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-user

5. Save the changes. The properties are saved.
6. In Eclipse, go to **training.ui.apps**.
7. Right-click **training.ui.apps** and select Import > Import from Server.

8. Accept the default values and click **Finish**. The changes are imported from server, as shown:



9. Navigate to the **Web Console**, and choose **Status > Sling Service User Mappings**. The user mappings are displayed.

10. Do a browser find: **training** and observe the new user mapping, as shown:

 **Note:** You can find the Persistent Identity (PID) in the Web Console configuration console by searching Service User. The **-training** is a unique identifier because this configuration is a configuration factory and, therefore, it needs a unique identifier.

```
x Find: training Previous Next Options ▾ 4 matches
com.adobe.aemds.guide.aemds-guide-core / fd-service
com.adobe.aemds.guide.aemds-guide-core-impl / fd-service
com.adobe.aemds.formsmanager.adobe-aemds-formsanddocuments-core / fd-service
com.adobe.livecycle.forms.forms-manager-package-bundle / fd-service
com.adobe.livecycle.formsportal-bundle / fd-service
com.adobe.forms.common.adobe-xfaforms-common / fd-service
oauthservice
com.adobe.cq.social.cq-social-commons / user-reader / oauthservice
resourcecollectionservice
com.adobe.cq.social.cq-social-content-fragments-impl / dam-collection-service / resourcecollectionservice
com.adobe.cq.social.cq-social-moderation / dam-collection-service / resourcecollectionservice
com.adobe.cq.social.cq-social-console / dam-collection-service / resourcecollectionservice
service-user-admin
com.adobe.cq.social.cq-social-console / service-user-admin / service-user-admin
social-enablement-tmp-manager
com.adobe.cq.social.cq-social-enablement-impl / tmpmanager / social-enablement-tmp-manager
training-user
com.adobe.training.core / training / training-user
translation-job-service
com.day.cq.wcm.translation / translation-job / translation-job-service
com.day.cq.dam.cq-dam-core / translation-job / translation-job-service
*** Mappings by principals (122 principals): (format: service name / sub service name / principal names)
account-manager
com.adobe.cq.cq-account / account-management-service / [account-manager, content-reader-service, group]
activity-service
com.adobe.granite.activitystreams / activity-service / [activity-service]
activitypurgesrv
com.day.cq.dam.cq-dam-core / activitypurgesrv / [activitypurgesrv]
analytics-content-updater-service
com.adobe.cq.cq-analytics-integration / content-updater / [analytics-content-updater-service]
analyticssservice
```

 **Note:** You can create a node with the serialized XML document instead. Working with XMLs (that represent nodes) outside of AEM is a typical developer practice in an IDE. The XML file named **UserMapping-training.xml** is provided as part of **/Exercise\_Files/09\_Handling\_Events**.

# Access DataLayer in Adobe Experience Manager using Sling



## Introduction

Sling Models provide excellent support for web content authors to build pages in Adobe Experience Manager (AEM) and to easily customize the pages in AEM to their requirements. Sling Models let AEM developers access Sling content without creating their own adapters. Thus, avoiding a large amount of boilerplate code.

## Objectives

After completing this course, you will be able to:

- Describe ResourceResolver
- Describe Sling Models
- Create a Sling Model
- Discuss Sling Model Exporter in AEM
- Extend a core component
- Discuss Query Index
- Describe Index configuration
- Configure queries to search resources

# Understanding ResourceResolver

The ResourceResolver defines the service API which may be used to resolve resource objects. The resource resolver is available to the request processing servlet through the `SlingHttpServletRequest.getResourceResolver()` method. A resource resolver can also be created through the `ResourceResolverFactory`.

The ResourceResolver is also adaptable to get adapters to other types. A JCR based resource resolver might support adapting to the JCR Session used by the resolver to access the JCR Repository.

A ResourceResolver is generally not thread safe. As a consequence, an application that uses the resolver, its returned resources and objects resulting from adapting either the resolver or a resource, must provide proper synchronization to ensure not more than one thread concurrently operates against a single resolver, resource, or resulting objects.

## Accessing Resources

The ResourceResolver interface defines two kinds of methods to access resources—the `resolve` method and the `getResource` method. The difference lies in the algorithm applied to find the requested resource, and in the behavior in case a resource cannot be found.

## Lifecycle

A Resource Resolver has a life cycle which begins with the creation of the Resource Resolver using any of the factory methods and ends with calling the `close()` method. It is very important to call the `close()` method after the resource resolver is not used anymore to ensure the system resources are properly cleaned up.

To check whether a Resource Resolver can still be used, the `isLive()` method can be used.

## 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, and mapping to certain objects
- By an AdapterFactory that can map arbitrary objects
- A combination of both

## Working with Sling Models

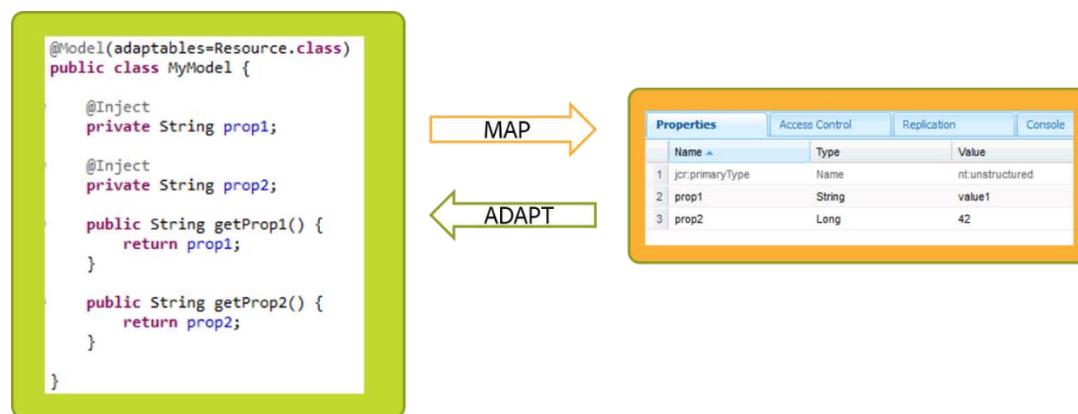
Sling Models let you map Java objects to Sling resources. When developing an AEM project, you can define a model object (a Java object) and map that object to Sling resources. Sling Models have the following objectives:

- Entirely annotation driven
- Use standard annotations where possible
- Pluggable
- Support out-of-the-box (OOTB) resource properties, SlingBindings, OSGi services, request and 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 the Sling Model with AEM without creating your own adapters
- You need to map your Plain Old Java Object (POJO) into a Sling Resource, as shown:



## Benefits of Using Sling Models

- Saves time from creating own adapters (avoiding boilerplate code)
- Supports both classes and interfaces
- Allows you to adapt multiple objects—minimal required resource and SlingHttpServletRequest
- Works with existing Sling infrastructure (for example, changes to other bundles are not required)
- Provides the ability to mock dependencies with tools like Mockito @InjectMocks

## Sling Model Implementation

Sling Models allow developers to inject method return values (in an interface) and fields (in a class) based on Sling resources and OSGi services. Most injections are available when adapting either a Resource or SlingHttpServletRequest object.

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, as shown:

```

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. }
```

In most cases, using a Sling Model Interface requires 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 customization 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 Java class is annotated with @Model and the adaptable class. Fields that need to be injected are annotated with @Inject, as shown:

```
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 will be 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 does not 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:

The following table shows the different types of Sling Model annotations and its 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 <i>propertyName</i> will be looked up from the Resource (after first adapting it to a ValueMap) and it is injected. If property is not found, it will return null.
@Default	@Inject @Default(values="AEM") private String technology;	A default value (for Strings, Arrays & primitives)
@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(Value Map.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).  //Code snippet will ensure that resource is retrieved from the bindings, not a request attribute.
@PostConstruct	@PostConstruct protected void sayHello() { logger.info("hello"); }	Allows for the definition of methods to be executed after the instance has been instantiated, and all the injects have been performed.

+

## 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 is 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 ensure the right injector is used)
- Better IDE support

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>

The screenshot shows the Adobe Experience Manager Web Console interface. At the top, it says "Adobe Experience Manager Web Console" and "Sling Models". Below the header is a navigation bar with links: Main, OSGi, Sling, Status (which is selected), and Web Console. To the right of the navigation bar is the Adobe logo. Underneath the navigation bar, there is a timestamp: "Date: January 7, 2016 9:49:03 PM CET". Below the timestamp are four download buttons: "Download As Text", "Download As Zip", "Download Full Text", and "Download Full Zip". The main content area displays a list of Sling Model injectors. The list is organized into sections: "Sling Models Injectors:", "Sling Models Inject Annotation Processor Factories:", and "Sling Models Implementation Pickers:". Each section contains a list of injector names, such as "resource-resolver", "script-bindings", "valuemap", etc.

```
Sling Models Injectors:  
resource-resolver - org.apache.sling.models.impl.injectors.ResourceResolverInjector  
script-bindings - org.apache.sling.models.impl.injectors.BindingsInjector  
valuemap - org.apache.sling.models.impl.injectors.ValueMapInjector  
resource-path - org.apache.sling.models.impl.injectors.ResourcePathInjector  
child-resource - org.apache.sling.models.impl.injectors.ChildResourceInjector  
request-attributes - org.apache.sling.models.impl.injectors.RequestAttributeInjector  
osgi-services - 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 the messages in to appropriate log file.

## Exercise 1: Create a Sling Model

In this exercise, you will create a Sling Model as a Java class and map it into a resource that consists of a set of properties on 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 display them in the browser.

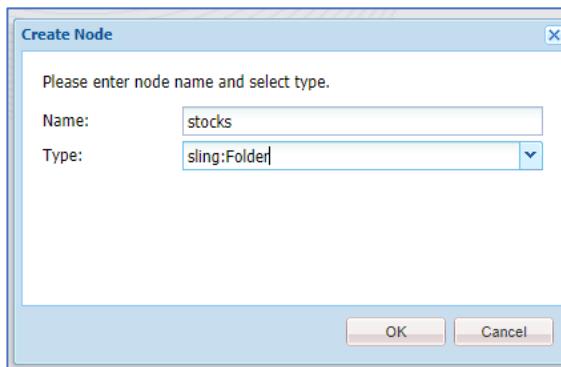
This exercise includes three tasks:

1. Create data for the Sling Model
2. Adapt the resource to a Sling Model
3. Adapt the request to a Sling Model

### Task 1: Create data for the Sling Model

This first task will create an adhoc data structure that our Sling Model will represent.

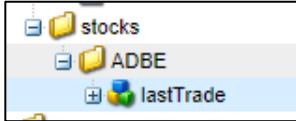
1. In CRXDE Lite, right-click **content** and select **Create > Create Node**. The **Create Node** dialog box opens.
2. In the **Create Node** dialog box, carry out the following functions, as shown:
  - a. In the **Name** field, enter **stocks**.
  - b. Select **sling:Folder** from the **Type** drop-down menu.



3. Click **OK**. The node is created.
4. Click **Save All**.
5. Create a child node under **stocks**, carry out the following functions:
  - a. In the **Name** field, enter **ADBE**.
  - b. Select **sling:OrderedFolder** from the **Type** drop-down menu.
6. Click **OK** and select **Save All** to save the changes. The node **ADBE** is created.

7. Create a child node under **ADBE**, carry out the following functions:

- In the **Name** field, enter **lastTrade**.
  - Select **nt:unstructured** from the **Type** drop-down menu.
8. Click **OK**. The node is created, as shown:



9. Click Save All.

10. In the newly created node, create the following four properties as follows:

- Name: **lastTrade** of type **String** and Value: **300**
- Name: **dayOfLastUpdate** of type **String** and Value: **11/13/2016**
- Name: **companyName** of type **String** and Value: **Adobe Systems Incorporated**
- Name: **week52High** of type **String** and Value: **310**

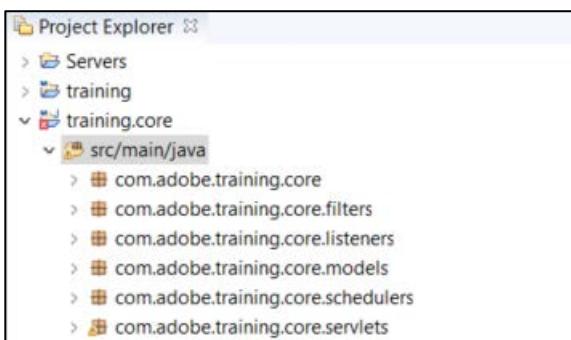
Properties			Access Control	Replication	Console	Build Info
	Name	Type	Value			
1	companyName	String	Adobe Systems Incorporated			
2	dayOfLastUpdate	String	11/13/2016			
3	jcr:primaryType	Name	nt:unstructured			
4	lastTrade	String	300			
5	week52High	String	310			

11. Click Save All.

## Task 2: Adapt the resource to a Sling Model

In this task you will adapt a sling resource to the Sling Model.

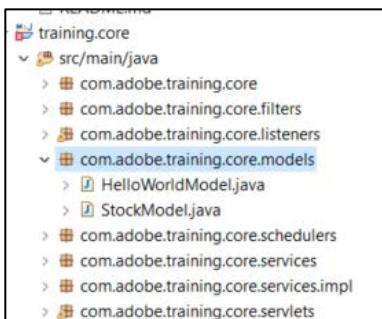
1. Launch **Eclipse** by double-clicking the **Eclipse** shortcut on the desktop. The Eclipse Launcher opens.
2. Specify the Workspace as **C:\Workspace** and click **Launch**. The Eclipse Workspace opens.
3. In Project Explorer, navigate to **training.core > src/main/java**, as shown:



4. Copy the **StockModel.java** file from **/Exercises\_Folder/10\_Sling\_Models**.

 **Note:** The code is provided as part of the Exercise\_Files under **/Exercise\_Files/10\_Sling\_Models/**. Copy and paste the file from the exercise files referenced to StockModel.java to Eclipse. Please do not copy the code from this exercise book. The code in the student guide is only for illustrative purposes.

5. Right-click **com.adobe.training.core.models** and paste the file. The **StockModel.java** class is created, as shown:



6. Double-click **StockModel.java** to open it in editor, as shown:

```
41. package com.adobe.training.core.models;
42.
43. import javax.inject.Named;
44.
45. import org.apache.sling.api.resource.Resource;
46. import org.apache.sling.api.resource.ValueMap;
47. import org.apache.sling.models.annotations.DefaultInjectionStrategy;
```

```

48. import org.apache.sling.models.annotations.Model;
49. import org.apache.sling.models.annotations.injectorspecific.ChildResource;
50. import org.apache.sling.models.annotations.injectorspecific.Self;
51.
52. /**
53.  * This model represents an IEX api stock data structure created from the StockDataImporter:
54.  * /content/stocks/
55.  * + <STOCK_SYMBOL> [sling:OrderedFolder]
56.  *   + lastTrade [nt:unstructured]
57.  *     - companyName = <value>
58.  *     - sector = <value>
59.  *     - lastTrade = <value>
60.  *     - ..
61. */
62.
63. @Model(adaptables=Resource.class, defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL)
64. public class StockModel{
65.
66.     @Self
67.     private Resource stock;
68.
69.     @ChildResource
70.     private Resource lastTrade;
71.
72.     @ChildResource
73.     @Named("lastTrade")
74.     private ValueMap lastTradeValues;
75.
76.
77.
78.     //Uses the resource to read the stock symbol
79.     public String getSymbol() {
80.         return stock.getName();
81.     }
82.
83.     // uses the ValueMap to read the stock data
84.     public double getLastTrade() {
85.         return lastTradeValues.get("lastTrade", Double.class);
86.     }
87.     public String getRequestDate() {
88.         return lastTradeValues.get("dayOfLastUpdate", String.class);
89.     }
90.     public String getRequestTime() {
91.         return lastTradeValues.get("timeOfUpdate", String.class);
92.     }
93.     public double getUpDown() {
94.         return lastTradeValues.get("upDown", Double.class);
95.     }
96.     public double getOpenPrice() {
97.         return lastTradeValues.get("openPrice", Double.class);
98.     }
99.     public double getRangeHigh() {
100.         return lastTradeValues.get("rangeHigh", Double.class);
101.     }
102.     public double getRangeLow() {
103.         return lastTradeValues.get("rangeLow", Double.class);
104.     }
105.     public int getVolume() {
106.         return lastTradeValues.get("volume", Integer.class);
107.     }
108.     public String getCompanyName() {
109.         return lastTradeValues.get("companyName", String.class);

```

```

110.        }
111.        public String getSector() {
112.            return lastTradeValues.get("sector", String.class);
113.        }
114.        public double get52weekHigh() {
115.            return lastTradeValues.get("week52High", Double.class);
116.        }
117.        public double get52weekLow() {
118.            return lastTradeValues.get("week52Low", Double.class);
119.        }
120.        public double getYtdPercentChange() {
121.            return lastTradeValues.get("ytdPercentageChange", Double.class);
122.        }
123.    }
124.

```



**Note:** This StockModel will work with the StockDataImporter that you will create later in this course. Because this data importer is not yet created, you will use a servlet and dummy data to simulate the input and output for the StockModel.

7. Copy the **SlingModelServlet.java** file from Exercises\_Folder/10\_Sling\_Models.
8. Right-click **com.adobe.training.core.servlets** and paste the file. The **SlingModelServlet.java** class is created.
9. Double-click **SlingModelServlet.java** to examine the code:

```

1. package com.adobe.training.core.servlets;
2.
3. import java.io.IOException;
4.
5. import javax.servlet.Servlet;
6. import javax.servlet.ServletException;
7.
8. import org.osgi.service.component.annotations.Component;
9.
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.resource.ResourceResolver;
14. import org.apache.sling.api.resource.ValueMap;
15. import org.apache.sling.api.servlets.SlingAllMethodsServlet;
16.
17. import org.slf4j.Logger;
18. import org.slf4j.LoggerFactory;
19.
20. import com.adobe.training.core.models.StockModel;
21. //import com.adobe.training.core.models.StockModelAdaptFromRequest;
22.
23. /**
24. * Example URI http://localhost:4502/content/trainingproject/en.model.html/content/stocks/ADBE
25. *
26. * To use this servlet, a content structure must be created:
27. * /content/stocks/
28. *   + ADBE [sling:OrderedFolder]
29. *     + lastTrade [nt:unstructured]

```

+

```

30. *      - lastTrade = "300"
31. *      - dayOfLastUpdate = "11/13/2016"
32. *      - companyName = "Adobe Systems Incorporated"
33. *      - week52High = "310"
34. */
35.
36. @Component(service = Servlet.class,
37.             property = {
38.                 "sling.servlet.resourceTypes=trainingproject/components/structure
39.                 /page",
40.                 "sling.servlet.selectors=model"
41.             })
42. public class SlingModelServlet extends SlingAllMethodsServlet{
43.     private final Logger logger = LoggerFactory.getLogger(this.getClass());
44.
45.     private static final long serialVersionUID = 1L;
46.
47.     @Override
48.     public void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException{
49.         response.setContentType("text/html");
50.         try {
51.             // Get the resource (a node in the JCR) using ResourceResolver from the request
52.             (ResourceResolver resourceResolver = request.getResourceResolver()) {
53.
54.                 //Specify the node of interest in the suffix on the request
55.                 String nodePath = request.getRequestPathInfo().getSuffix();
56.                 if(nodePath != null){
57.                     Resource resource = resourceResolver.getResource(nodePath);
58.
59.                     // Adapt resource properties to variables using ValueMap, and log their values
60.                     Resource parent = resource.getChild("lastTrade");
61.                     ValueMap valueMap=parent.adaptTo(ValueMap.class);
62.                     response.getOutputStream().println("<h3>");
63.                     response.getOutputStream().println(resource.getName() + " lastTrade node with Val
ueMap is");
64.                     response.getOutputStream().println("</h3><br />");
65.                     response.getOutputStream().println("(Last Trade) " + valueMap.get("lastTrade").toS
tring() + " (Requested Day) " + valueMap.get("dayOfLastUpdate").toString()
66.                                     + " (Company) " + valueMap.get("companyName").toString() + " (52 week Hig
h) " + valueMap.get("week52High").toString());
67.
68.                     //Adapt the resource to our model (part 1)
69.                     StockModel stockModel = resource.adaptTo(StockModel.class);
70.                     response.getOutputStream().println("<br /><h3>");
71.                     response.getOutputStream().println(stockModel.getSymbol() + " lastTrade node with
StockModel is");
72.                     response.getOutputStream().println("</h3><br />");
73.                     response.getOutputStream().println("(Last Trade) " + stockModel.getLastTrade() +
" (Requested Day) " + stockModel.getRequestDate()
74.                                     + " (Company) " + stockModel.getCompanyName() + " (52 week High) " + stockMod
el.get52weekHigh());
75.
76.                     //Adapt the request object to our model (part 2)
77.                     /*StockModelAdaptFromRequest stockModelAdaptFromRequest = request.adaptTo(StockMo
delAdaptFromRequest.class);
78.                     response.getOutputStream().println("<br /><h3>");
79.                     response.getOutputStream().println(stockModelAdaptFromRequest.getSymbol() + " las
tTrade node with StockModelAdaptFromRequest is");
80.                     response.getOutputStream().println("</h3><br />");
81.                     response.getOutputStream().println("(Last Trade) " + stockModelAdaptFromRequest.g
etLastTrade() + " (Requested Time) " + stockModelAdaptFromRequest.get

```

```

82.           + " (Company) " + stockModelAdaptFromRequest.getCompanyName() + " (52 week Hi
gh) " + stockModelAdaptFromRequest.get52weekHigh();*/
83.       }
84.       else {
85.           response.getWriter().println("Can't get the last trade node, enter a suffix in th
e URI");
86.       }
87.   } catch (Exception e) {
88.       response.getWriter().println("Can't read last trade node. make sure the suffix path e
xists!");
89.   }
90. }
91. response.getWriter().close();
92. }
93. }

```

10. Right-click **training.core**, and perform **Run As > Maven install** to build the project. The project runs successfully.

11. Verify the bundle installed successfully, as shown:

```

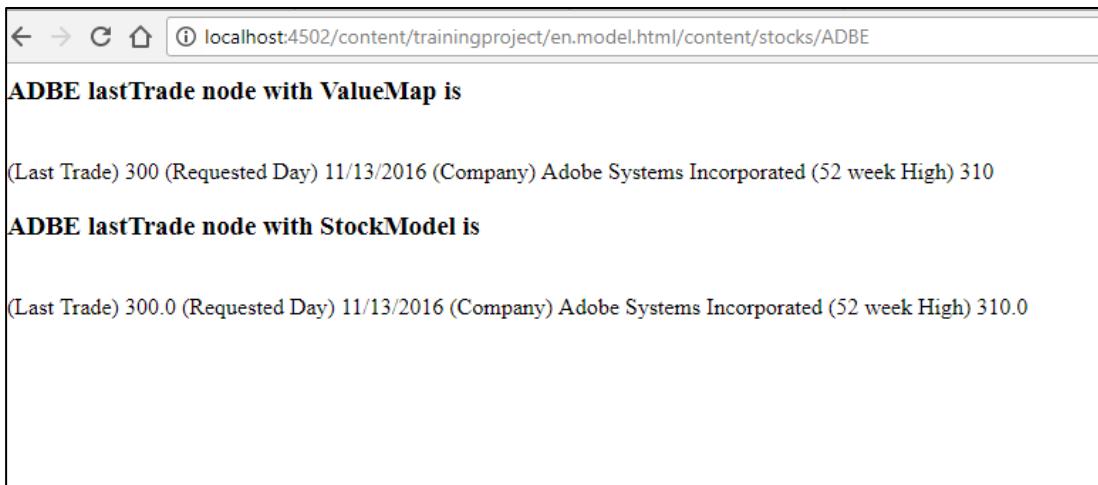
[INFO] Installing C:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar to C:\Users\prpurush\.m2\repository\com\adobe\trai
[INFO] Installing C:\Users\prpurush\Dev1\training\core\pom.xml to C:\Users\prpurush\.m2\repository\com\adobe\training.core\0.0.1-SNAPSHOT\trai
[INFO]
[INFO] --- maven-bundle-plugin:3.3.0: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:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar) to http://local
[INFO] Bundle installed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.366 s
[INFO] Finished at: 2018-03-29T16:10:08-07:00
[INFO] Final Memory: 31M/360M
[INFO] -----

```

12. Go to the following URL:

<http://localhost:4502/content/trainingproject/en.model.html/content/stocks/ADBE>

13. Verify the message returned by the model is displayed in the browser, as shown:



## Task 3: Adapt the request to a Sling Model

In this task you will adapt the request rather than the resource to get the same outcome.

1. Copy the **StockModelAdaptFromRequest.java** file from /Exercises\_Folder/10\_Sling\_Models.

 **Note:** The code is provided as part of the Exercise\_Files under /Exercise\_Files/10\_Sling\_Models/. Copy and paste the file from the exercise files referenced to **StockModelAdaptFromRequest.java** to Eclipse. Please do not copy the code from this exercise book. The code in the student guide is only for illustrative purposes.

2. Right-click **com.adobe.training.core.models** and paste the file. The StockModelAdaptFromRequest.java class is created.
3. Double-click **StockModelAdaptFromRequest.java** to open it in editor, as shown:

```
1. package com.adobe.training.core.models;
2.
3. import org.apache.sling.api.SlingHttpServletRequest;
4. import org.apache.sling.api.resource.Resource;
5. import org.apache.sling.api.resource.ValueMap;
6. import org.apache.sling.models.annotations.DefaultInjectionStrategy;
7. import org.apache.sling.models.annotations.Model;
8. import org.apache.sling.models.annotations.injectorspecific.ResourcePath;
9. import org.apache.sling.models.annotations.injectorspecific.Self;
10.
11. import javax.annotation.PostConstruct;
12.
13. /**
14. * This model represents an IEX api stock data structure created from the StockDataImporter:
15. * /content/stocks/
16. * + <STOCK_SYMBOL> [sling:OrderedFolder]
17. *     + lastTrade [nt:unstructured]
18. *         - companyName = <value>
19. *         - sector = <value>
20. *         - lastTrade = <value>
21. *         - ...
22. */
23.
24. @Model(adaptables=SlingHttpServletRequest.class, defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL)
25. public class StockModelAdaptFromRequest {
26.
27.     @Self
28.     private SlingHttpServletRequest request;
29.
30.     @ResourcePath(path = "/")
31.     private Resource root;
32.
33.     private ValueMap lastTradeValues;
34.     private Resource stock;
35.
36.     @PostConstruct
37.     private String stockContentPath() {
38.         String path = request.getRequestPathInfo().getSuffix();
39.
40.         stock = root.getChild(path);
41.         lastTradeValues = root.getChild(path).getChild("lastTrade").getValueMap();
42.         return path;
43.     }
44. }
```

```
43.    }
44.
45.    //Uses the resource to read the stock symbol
46.    public String getSymbol() {
47.        return stock.getName();
48.    }
49.
50.    // uses the ValueMap to read the stock data
51.    public double getLastTrade() {
52.        return lastTradeValues.get("lastTrade", Double.class);
53.    }
54.    public String getRequestDate() {
55.        return lastTradeValues.get("dayOfLastUpdate", String.class);
56.    }
57.    public String getRequestTime() {
58.        return lastTradeValues.get("timeOfUpdate", String.class);
59.    }
60.    public double getUpDown() {
61.        return lastTradeValues.get("upDown", Double.class);
62.    }
63.    public double getOpenPrice() {
64.        return lastTradeValues.get("openPrice", Double.class);
65.    }
66.    public double getRangeHigh() {
67.        return lastTradeValues.get("rangeHigh", Double.class);
68.    }
69.    public double getRangeLow() {
70.        return lastTradeValues.get("rangeLow", Double.class);
71.    }
72.    public int getVolume() {
73.        return lastTradeValues.get("volume", Integer.class);
74.    }
75.    public String getCompanyName() {
76.        return lastTradeValues.get("companyName", String.class);
77.    }
78.    public String getSector() {
79.        return lastTradeValues.get("sector", String.class);
80.    }
81.    public double get52weekHigh() {
82.        return lastTradeValues.get("week52High", Double.class);
83.    }
84.    public double get52weekLow() {
85.        return lastTradeValues.get("week52Low", Double.class);
86.    }
87.    public double getYtdPercentChange() {
88.        return lastTradeValues.get("ytdPercentageChange", Double.class);
89.    }
90. }
```



**Note:** The code is provided as part of the Exercise\_Files under /Exercise\_Files/10\_Sling\_Models/. Copy and paste the file from the exercise files referenced to SlingModelServlet\_P2.java to Eclipse. Please do not copy the code from this exercise book. The code in the student guide is only for illustrative purposes.

4. Edit **SlingModelServlet.java** class by uncommenting the following lines of code, as shown:

```

1. package com.adobe.training.core.servlets;
2.
3. import java.io.IOException;
4.
5. import javax.servlet.Servlet;
6. import javax.servlet.ServletException;
7.
8. import org.osgi.service.component.annotations.Component;
9.
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.resource.ResourceResolver;
14. import org.apache.sling.api.resource.ValueMap;
15. import org.apache.sling.api.servlets.SlingAllMethodsServlet;
16.
17. import org.slf4j.Logger;
18. import org.slf4j.LoggerFactory;
19.
20. import com.adobe.training.core.models.StockModel;
21. import com.adobe.training.core.models.StockModelAdaptFromRequest; // Lines 20 and 21 are highlighted with a red border
22.
23. /**
24. * Example URI http://localhost:4502/content/trainingproject/en.model.html/content/stocks/ADBE
25. *
26. * To use this servlet, a content structure must be created:
27. * /content/stocks/
28. *   + ADBE [sling:OrderedFolder]
29. *     + lastTrade [nt:unstructured]
30. *       - lastTrade = "300"
31. *       - dayOfLastUpdate = "11/13/2016"
32. *       - companyName = "Adobe Systems Incorporated"
33. *       - week52High = "310"
34. */
35.
36. @Component(service = Servlet.class,
37.             property = {
38.               "sling.servlet.resourceTypes=trainingproject/components/structure/page",
39.               "sling.servlet.selectors=model"
40.             })
41.
42. public class SlingModelServlet extends SlingAllMethodsServlet{
43.     private final Logger logger = LoggerFactory.getLogger(this.getClass());
44.
45.     private static final long serialVersionUID = 1L;
46.
47.     @Override
48.     public void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException{
49.         response.setContentType("text/html");
50.         try {
51.             // Get the resource (a node in the JCR) using ResourceResolver from the request
52.             ResourceResolver resourceResolver = request.getResourceResolver();
53.
54.             //Specify the node of interest in the suffix on the request
55.             String nodePath = request.getRequestPathInfo().getSuffix();
56.             if(nodePath != null){
57.                 Resource resource = resourceResolver.getResource(nodePath);
58.             }
59.         }
60.     }
61. }
```

```

59.          // Adapt resource properties to variables using ValueMap, and log their values
60.          Resource parent = resource.getChild("lastTrade");
61.          ValueMap valueMap=parent.adaptTo(ValueMap.class);
62.          response.getOutputStream().println("<h3>");
63.          response.getOutputStream().println(resource.getName() + " lastTrade node with Val
ueMap is");
64.          response.getOutputStream().println("</h3><br />");
65.          response.getOutputStream().println("(Last Trade) " + valueMap.get("lastTrade").toS
tring() + " (Requested Day) " + valueMap.get("dayOfLastUpdate").toString()
66.                      + " (Company) " + valueMap.get("companyName").toString() + " (52 week Hig
h) " + valueMap.get("week52High").toString());
67.
68.          //Adapt the resource to our model (part 1)
69.          StockModel stockModel = resource.adaptTo(StockModel.class);
70.          response.getOutputStream().println("<br /><h3>");
71.          response.getOutputStream().println(stockModel.getSymbol() + " lastTrade node with
StockModel is");
72.          response.getOutputStream().println("</h3><br />");
73.          response.getOutputStream().println("(Last Trade) " + stockModel.getLastTrade()
" (Requested Day) " + stockModel.getRequestDate()
74.                      + " (Company) " + stockModel.getCompanyName() + " (52 week High) " + stockMod
el.get52weekHigh());
75.
76.          //Adapt the request object to our model (part 2)
77.          StockModelAdaptFromRequest stockModelAdaptFromRequest = request.adaptTo(StockMode
lAdaptFromRequest.class);
78.          response.getOutputStream().println("<br /><h3>");
79.          response.getOutputStream().println(stockModelAdaptFromRequest.getSymbol() + " las
tTrade node with StockModelAdaptFromRequest is");
80.          response.getOutputStream().println("</h3><br />");
81.          response.getOutputStream().println("(Last Trade) " + stockModelAdaptFromRequest.g
etLastTrade() + " (Requested Time) " + stockModel.getRequestDate()
82.                      + " (Company) " + stockModelAdaptFromRequest.getCompanyName() + " (52 week Hi
gh) " + stockModelAdaptFromRequest.get52weekHigh());
83.      }
84.      else {
85.          response.getWriter().println("Can't get the last trade node, enter a suffix in th
e URI");
86.      }
87.      } catch (Exception e) {
88.          response.getWriter().println("Can't read last trade node. make sure the suffix patin
exists!");
89.          logger.error(e.getMessage());
90.      }
91.      response.getWriter().close();
92.  }
93. }
```

5. Click **Save**.

6. Right-click **training.core**, and perform **Run As > Maven install** to build the project. The project runs successfully.

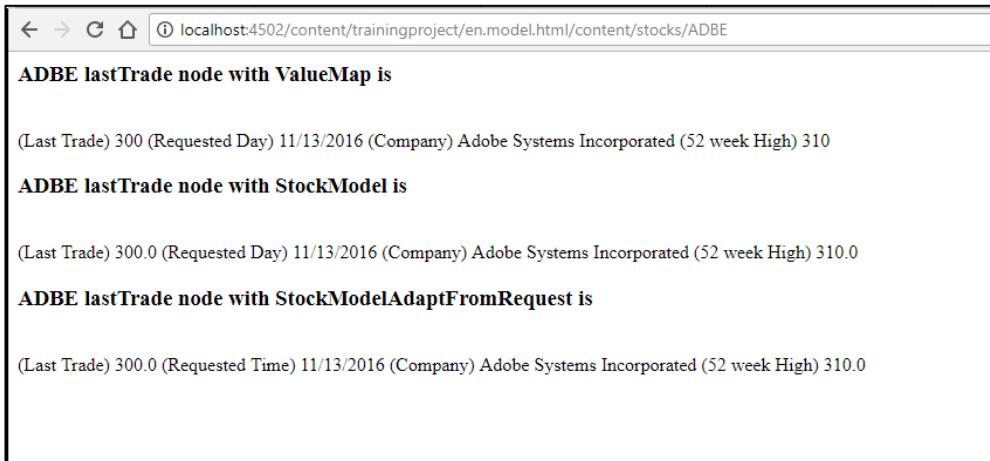
7. Verify the bundle installed successfully, as shown:

```
[INFO] Installing C:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar to C:\Users\prpurush\.m2\repository\com\adobe\trai
[INFO] Installing C:\Users\prpurush\Dev1\training\core\pom.xml to C:\Users\prpurush\.m2\repository\com\adobe\training.core\0.0.1-SNAPSHOT\trai
[INFO]
[INFO] --- maven-bundle-plugin:3.3.0: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:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar) to http://local
[INFO] Bundle installed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.366 s
[INFO] Finished at: 2018-03-29T16:10:08-07:00
[INFO] Final Memory: 31M/360M
[INFO] -----
```

8. Go to the following URL:

<http://localhost:4502/content/trainingproject/en.model.html/content/stocks/ADBE>

9. Verify the message returned by the model is displayed in the browser, as shown:



**Note:** Notice that you can adapt the request rather than the resource to our StockModel. The object you are adapting does not matter as long as the Sling Model is built to handle the object.

## Sling Model Exporter

The Sling Model Exporter feature allows new annotations to be added to Sling Models that define how the Model can be exported as a different Java object, or more commonly, serialized into a different format such as JSON.

Apache Sling provides a Jackson JSON exporter to cover the most common case of exporting Sling Models as JSON objects for consumption by programmatic web consumers, such as other web services and JavaScript applications.

Sling Model Exporter is perfect for leveraging Sling Models that already contain business logic that supports HTML renditions through HTL (or formerly JSP), and expose the same business representation as JSON for consumption by programmatic Web services or JavaScript applications.

Sling Model Exporter comes with a Sling provided Jackson Exporter that automatically serializes an *ordinary* Sling Model object into JSON. The Jackson Exporter, while quite configurable, at its core inspects the Sling Model object, and generates JSON using any *getter* methods as JSON keys, and the getter return values as the JSON values.

Sling Models objects can be exported to arbitrary Java objects through the Sling Models Exporter framework. Model objects can be programmatically exported by calling the ModelFactory method `exportModel()`. This method takes the following arguments:

- The model object
- An exporter name
- A target class
- A map of options

Apache Sling provides a Jackson JSON exporter to cover the most common case of exporting Sling Models as JSON objects for consumption by programmatic web consumers, such as other web services and JavaScript applications.

## Exercise 2: Extend a core component

In this exercise, you will upload and install a `trainingtitle` component. This will be the frontend HTL code for your component. You will then write a Sling Model to support the business logic.

1. From CRXDE Lite, click the Package icon. The **CRX Package Manager** page is displayed.

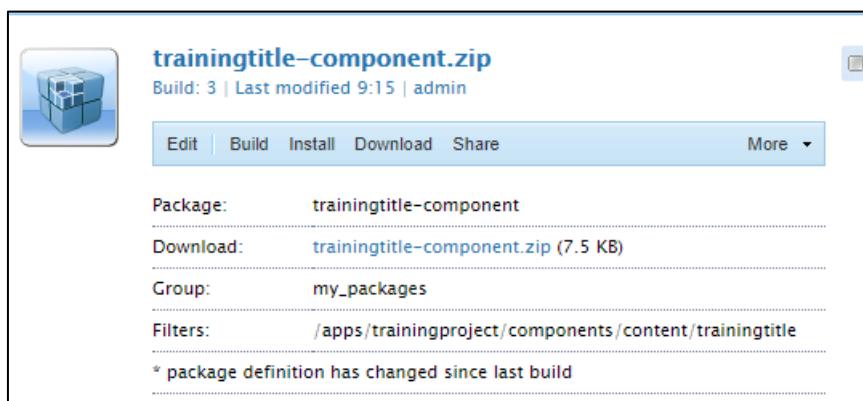
2. Click **Upload Package**, as shown. The **Upload Package** dialog box opens.



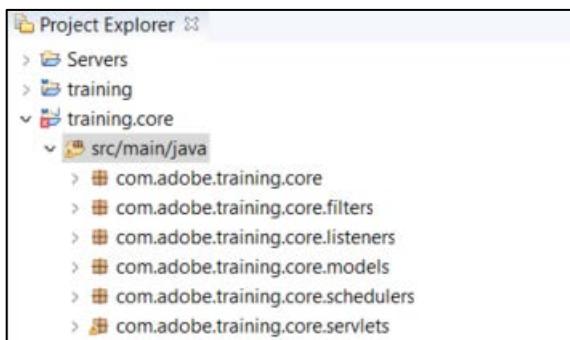
3. In the Upload Package dialog box, click Browse, and select the `trainingtitle-component.zip` package file from the Exercise\_Files\10\_Sling\_Models.

4. Click **OK**.

5. Verify the uploaded package is now available in **CRX Package Manager**, as shown:

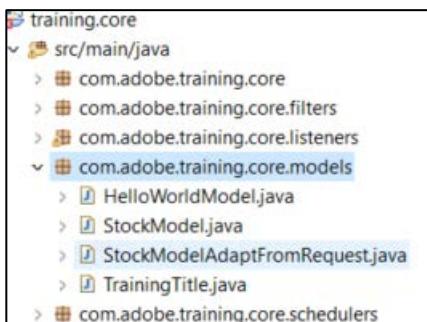


6. Click **Install**. The package is installed.
7. In Project Explorer, under training.ui.apps, select /src/main/content/jcr\_root [rep:root], right-click and select Import from Server. The Import from Repository dialog box appears.
8. Click **Finish**. The project is imported from the server.
9. In Project Explorer, navigate to **training.core > src/main/java**, as shown:



10. Copy the **TrainingTitle.java** file from Exercises\_Folder/10\_Sling\_Models.

11. Right-click **com.adobe.training.core.models** and paste the file. The **TrainingTitle.java** class is created, as shown:



12. Double-click **TrainingTitle.java** to open it in editor, as shown:

```

1. package com.adobe.training.core.models;
2.
3. import javax.annotation.PostConstruct;
4. import javax.inject.Inject;
5. import javax.inject.Named;
6.
7. import org.apache.sling.api.SlingHttpServletRequest;
8. import org.apache.sling.models.annotations.*;
9. import org.apache.sling.models.annotations.injectorspecific.InjectionStrategy;
10. import org.apache.sling.models.annotations.injectorspecific.ScriptVariable;
11. import org.apache.sling.models.annotations.injectorspecific.Self;
12. import org.apache.sling.models.annotations.injectorspecific.ValueMapValue;
13. import org.slf4j.Logger;
14.
15. import com.adobe.cq.export.json.ComponentExporter;
16. import com.day.cq.wcm.api.Page;
17.

```

```

18. @Model(adaptables=SlingHttpServletRequest.class,
19.         adapters= {ComponentExporter.class},
20.         resourceType="trainingproject/components/content/trainingtitle",
21.         defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL)
22. @Exporter(name = "jackson", extensions = "json")
23. public class TrainingTitle implements ComponentExporter{
24.
25.     @Self
26.     private SlingHttpServletRequest request;
27.
28.     @Inject
29.     @Named("log")
30.     private Logger logger;
31.
32.     @ScriptVariable
33.     private Page currentPage;
34.
35.     @ValueMapValue(name="subtitle", injectionStrategy=InjectionStrategy.OPTIONAL)
36.     private String subtitle;
37.
38.     @ValueMapValue(name="jcr:title", injectionStrategy=InjectionStrategy.OPTIONAL)
39.     private String title;
40.
41.     @ValueMapValue(name="type", injectionStrategy=InjectionStrategy.OPTIONAL)
42.     private String type;
43.
44.     @PostConstruct
45.     protected void initModel(){
46.         if(title == null){
47.             title = "";
48.         }
49.
50.         if(subtitle == null){
51.             subtitle = "";
52.         }
53.         logger.info("TrainingTitle InitModel");
54.     }
55.
56.     public String getText() {
57.         return title;
58.     }
59.
60.     public String getType() {
61.         return type;
62.     }
63.
64.     public String getSubtitle() {
65.         return subtitle;
66.     }
67.
68.     @Override
69.     public String getExportedType() {
70.         return request.getResource().getResourceType();
71.     }
72. }
```

13. Right-click **training.core**, and perform **Run As > Maven install** to build the project. The project runs successfully.

14. Verify the bundle installed successfully, as shown:

```
[INFO] Installing C:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar to C:\Users\prpurush\.m2\repository\com\adobe\training.core\0.0.1-SNAPSHOT\train
[INFO] Installing C:\Users\prpurush\Dev1\training\core\pom.xml to C:\Users\prpurush\.m2\repository\com\adobe\training.core\0.0.1-SNAPSHOT\train
[INFO]
[INFO] --- maven-bundle-plugin:3.3.0: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:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar) to http://local
[INFO] Bundle installed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.366 s
[INFO] Finished at: 2018-03-29T16:10:08-07:00
[INFO] Final Memory: 31M/360M
[INFO] -----
```

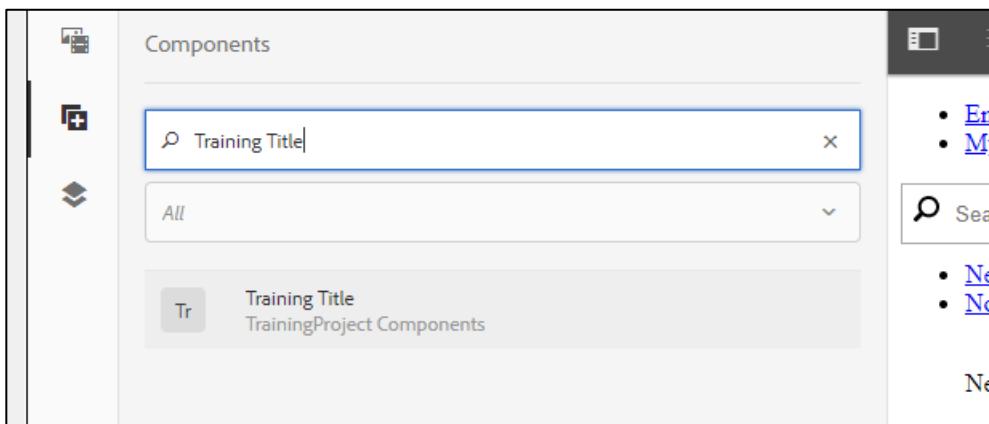
15. Go the **Sites** Console <http://localhost:4502/sites.html/content>.

16. Navigate to **Sites > TrainingProject Site**, and click the thumbnail on **English** and select the page, as shown:

The screenshot shows the AEM Sites Console interface. The URL in the address bar is `localhost:4502/sites.html/content/trainingproject`. The main area displays a tree view of site structures. The 'TrainingProject Site' node is currently selected. To its right, there are two columns: one for language (English, en) and one for preview. The English preview shows a sample page titled 'My Page!' with some placeholder text. The interface includes standard navigation buttons (back, forward, search, etc.) and toolbars for creating new pages, editing, and viewing properties.

17. Click **Edit (e)**. The page opens in a new tab for editing.

18. In the left pane, select the **Components** tab. Use the **Filter** option to find **Training Title** component, as shown:



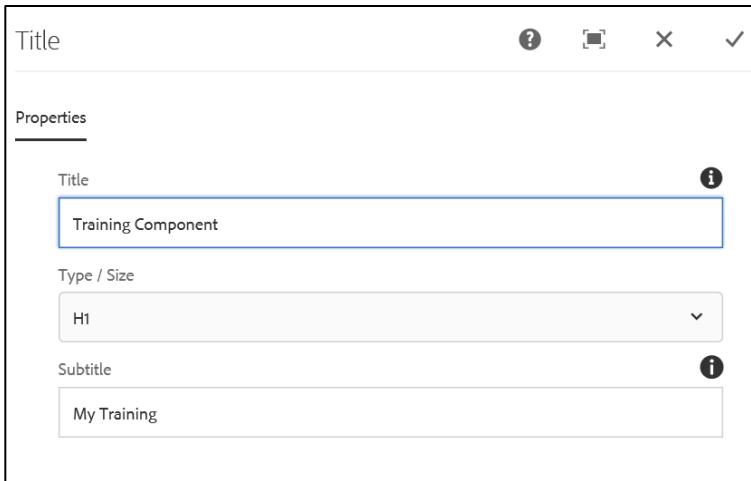
19. Drag the **Training Title** on the **Drag components here** container, as shown:

The screenshot shows a page editor interface. At the top, it says 'Sling Servlet injected this title'. Below that is a large black rectangular area containing text about Lorem Ipsum. Underneath the black area, there's developer info: 'Created by Scott Reynolds. + Hobbies include: [swimming, climbing]. Preferred programming language in AEM is HTL'. It also shows 'HelloWorldModel says:' followed by 'Hello World!', 'This is instance: c43e6ceb-7c7a-468d-a2ad-8f9f1ad0fedc', and 'Resource type is: trainingproject/components/content/helloworld'. At the bottom of the page, there's a red-bordered box containing the text 'Training Title'.

20. Double-click the **Training Title** components box to open it.

21. Add a title and subtitle, as shown:

+



22. Click **Done** (the checkmark symbol). The changes are saved.

23. Observe the component exported as JSON by using the URL, as shown:

<http://localhost:4502/content/trainingproject/en.model.json>

```
localhost:4502/content/trainingproject/en.model.json

{
  "designPath": "/libs/settings/wcm/designs/default",
  "title": "English",
  "lastModifiedDate": 1524089437628,
  "templateName": "content-page",
  "cssClassNames": "page basicpage",
  "language": "en",
  "itemsOrder": [
    "root",
    "image"
  ],
  "items": {
    "root": {
      "columnCount": 12,
      "gridClassNames": "aem-Grid aem-Grid--12 aem-Grid--default--12",
      "itemsOrder": [
        "languagenavigation",
        "search",
        "navigation",
        "breadcrumb",
        "responsivegrid"
      ],
      "items": {
        "languagenavigation": {
          "columnClassNames": "aem-GridColumn aem-GridColumn--default--12",
          "items": [
            {
              "children": [],
              "level": 0,
              "active": true,
              "title": "English",
              "locale": "en",
              "country": "",
              "language": "en",
              "url": "/content/trainingproject/en.html",
              "path": "/content/trainingproject/en",
              "lastModified": 1524089437628,
              "title": "English"
            }
          ]
        }
      }
    }
  }
}
```

Note this is a JSON for the entire page, but it contains the TrainingTitle sling model with the 3 getters.

24. View only the json output of the component by references the resource directly:

<http://localhost:4502/content/trainingproject/en/jcr:content/root/responsivegrid/trainingtitle.model.json>, as shown:



The screenshot shows a browser window with the URL <http://localhost:4502/content/trainingproject/en/jcr:content/root/responsivegrid/trainingtitle.model.json> in the address bar. The page content displays a JSON object representing a component configuration. The JSON structure is as follows:

```
{  
    "subtitle": "My Training ",  
    "type": "h1",  
    "text": "Training Component",  
    ":type": "trainingproject/components/content/trainingtitle"  
}
```

## Query Index

Apache 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.

**Note:** If a query reads more than 10,000 nodes in memory, 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 by using the system property `oak.queryLimitInMemory`.

Query Indices are defined under the `oak:index` node.

### Supported Query Languages

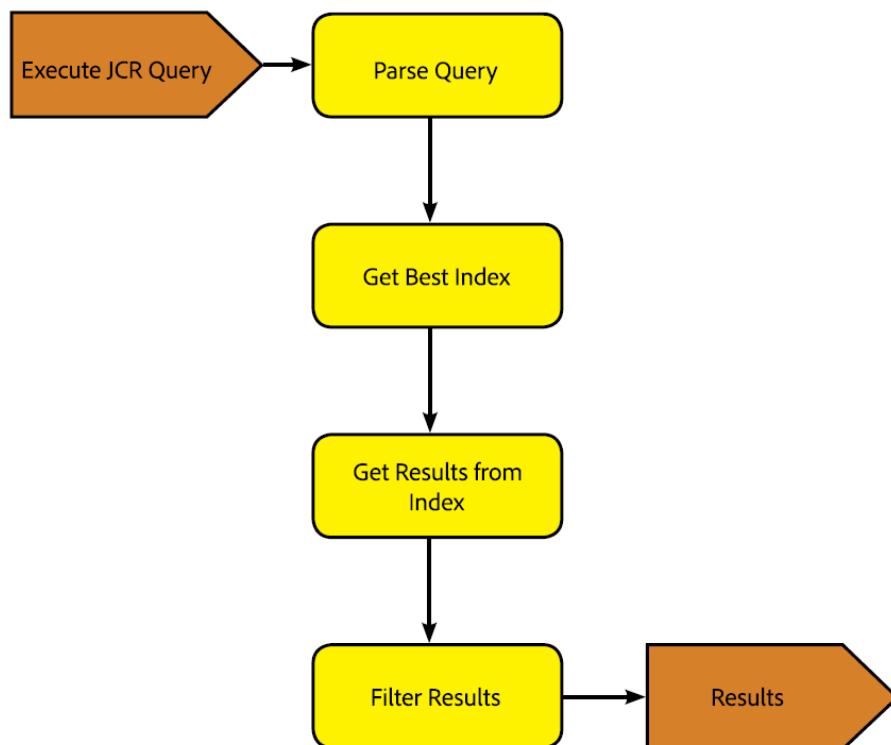
The Oak query engine supports the following languages:

- XPath
- SQL
- SQL-2

---

## 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 AEM. 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 the 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. After completed, the results from the most cost-effective index are retrieved. Finally, the results are filtered, 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 the data is read in memory 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, the so-called native constraints are supported by Oak. These constraints are passed directly to the full-text index. The full-text index 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.

### Configuring The Property Index

To configure the Property Index:

1. Open **CRXDE Lite** by going to <http://localhost:4502/crx/de/index.jsp>
2. Create a new node under `oak:index`.
3. Name the node `PropertyIndex`, and set the node type to `oak:QueryIndexDefinition`.
4. Click **OK** then set the following properties for the new node:

Name	Type	Value
type	String	property
propertyNames	Name	jcr:uuid

The above node will index the `jcr:uuid` property, whose job is to expose the universally unique identifier (UUID) of the node it is attached to.

5. Click **Save All** to save the changes—the `PropertyIndex` node will have three properties on its tab now.

## The Lucene Full-text Index

A full-text indexer based on Apache Lucene is available in AEM. If a full-text index is configured, then all the queries with a full-text condition use the full-text index, no matter if there are other conditions that are indexed and 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 the features that Lucene does, and it traverses all the 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:

1. Open **CRXDE Lite** and create a new node under `oak:index`.
2. Name the node **LuceneIndex** and set the node type to `oak:QueryIndexDefinition`.
3. Add the following properties to the node:

Name	Type	Value
<code>type</code>	String	lucene
<code>async</code>	String	async

4. Save the changes.

## Configuring 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 AEM occurs at the repository level.
- It can be configured to work as an embedded server with the AEM instance, or as a remote server.

To configure AEM 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 in the Web Console.
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 in the Web Console under **OSGi > Configuration**.
6. Click the **Edit** icon in the **Oak Solr embedded server configuration**, and update the configuration. The Solr home directory configuration will look for a folder with the same name in the AEM 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:
  - 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

When removing an index, it is always recommended to temporarily disable the index by setting the **type** property to **disabled** and do testing to ensure that your application functions correctly before actually deleting it.

## Indexing Tools

AEM also integrates two indexing tools present in AEM as part of the Adobe Consulting Services Commons toolset.

- Explain Query: A tool designed to help administrators understand how queries are executed.
- Oak Index Manager: A Web User Interface for maintaining existing indexes.

## 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 of the Query Tool :

- 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, as shown:

The screenshot shows the Adobe Experience Manager Query Performance interface. At the top, there are tabs for 'SLOW QUERIES', 'POPULAR QUERIES', and 'EXPLAIN QUERY'. The 'EXPLAIN QUERY' tab is selected. Below it, there are fields for 'Language \*' (set to 'XPath') and 'Query \*' (containing the query '/jcr:root/content/geometrixx/en/products/element(\*,nt:unstructured){@sling:resourceType=geometrixx/components/title}'). There are also two checkboxes: 'Include Execution Time' (unchecked) and 'Include Node Count' (unchecked). A large 'Explain' button is at the bottom.

A modal dialog titled 'Query Explanation' is displayed. It contains three sections: 'Indexes Used' (listing 'sling:resourceType'), 'Execution Plan' (showing the query plan: '[nt:unstructured] as [a] /\* property sling:resourceType = geometrixx%2Fcomponents%2Ftitle where ([a][sling:resourceType] = 'geometrixx/components/title') and (isdescendantnode([a], [/content/geometrixx/en/products])) /\*/'), and 'Logs' (displaying the execution logs). The logs show the process of finding an index definition for the query, evaluating plans for various indexes like Lucene, and handling the old index format.

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.

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.

Access DataLayer in Adobe Experience Manager using Sling

## 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.

You can access the Explain Query and Oak Index Manager tools by clicking 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. Example of value of a property—nodes whose `jcr:created` property is after 2007-03-14T00:00:00.000Z.
- Length of a property. Example of length of a property—nodes whose `jcr:data` property is longer than 100 KB.
- Existence of a property. Example of existence of a property—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:

```
NodeIterator 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]
```

## Search Performance

Fastest JCR search methodologies:

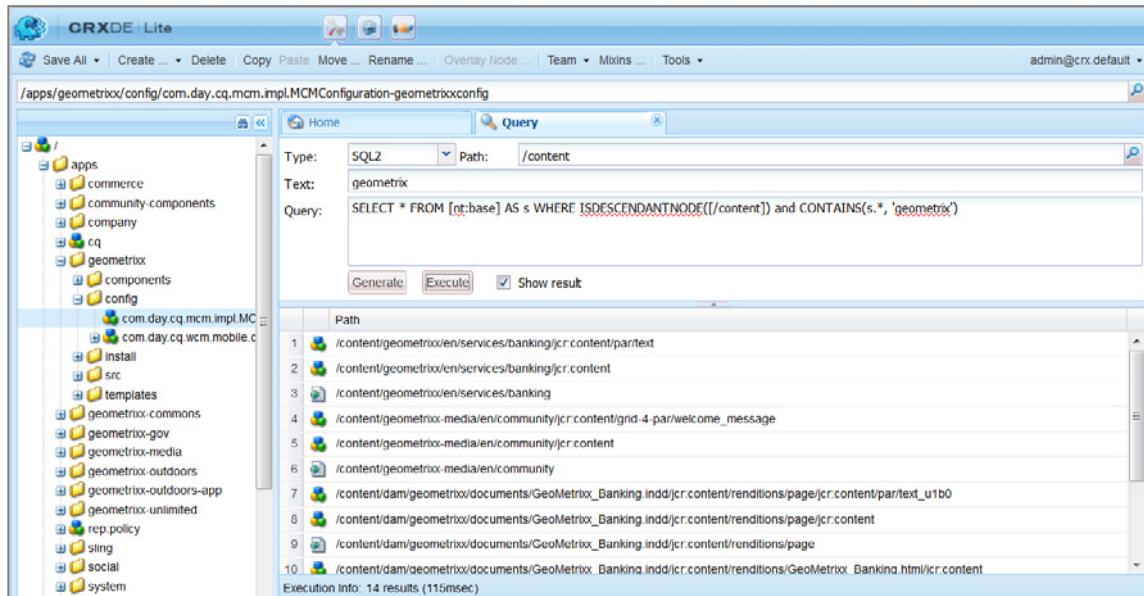
- Constraints on properties, Node types, full-text
- Typically O(n), where n is the number of results, versus to the total number of nodes
- Constraints on the path

JCR methodologies that need some planning:

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

## Testing Queries

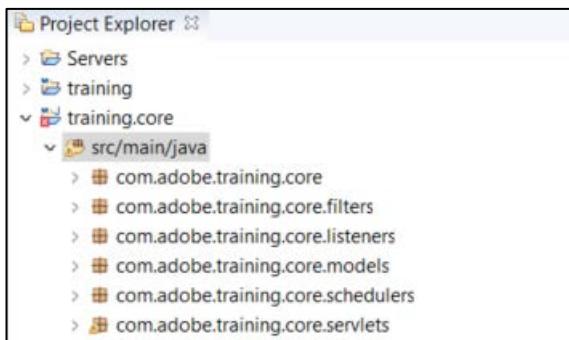
You can test your queries using CRXDE Lite. Access the Query Tool on the Tools menu from the top toolbar in CRXDE Lite, as shown:



## Exercise 3: Search resources with queries

In this exercise, you will do SQL query to search for a full text phrase in a website and then display the pages that contain that phrase.

1. In Project Explorer, navigate to training.core > src/main/java, as shown:



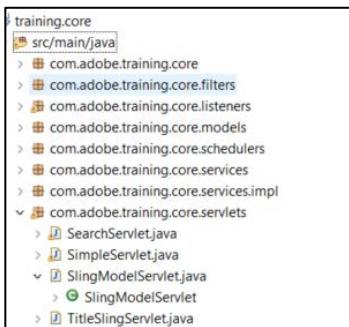
2. Copy the **SearchServlet.java** file from /Exercises\_Folder/10\_Sling\_Models.



**Note:** The code is provided as part of the Exercise\_Files under /Exercise\_Files/10\_Sling\_Models/. Copy and paste the file from the exercise files referenced to SearchServlet.java to Eclipse. Please do not copy the code from this exercise book. The code in the student guide is only for illustrative purposes.

---

3. Right-click **com.adobe.training.core.servlets** and paste the file. The **SearchServlet.java** class is created, as shown:



4. Double-click **SearchServlet.java** to open it in editor, as shown:

```
1. package com.adobe.training.core.servlets;
2.
3. import com.day.cq.wcm.api.PageManager;
4. import com.google.gson.Gson;
5. import org.apache.sling.api.SlingHttpServletRequest;
6. import org.apache.sling.api.SlingHttpServletResponse;
7. import org.apache.sling.api.resource.Resource;
8. import org.apache.sling.api.resource.ResourceResolver;
9. import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
10.
11. import org.osgi.service.component.annotations.Component;
12.
13. import javax.jcr.query.Query;
14.
15. import javax.servlet.Servlet;
16. import javax.servlet.ServletException;
17. import java.io.IOException;
18. import java.util.ArrayList;
19. import java.util.Iterator;
20.
21. /**
22. *
23. * Example URI: http://localhost:4502/content/trainingproject/en.search.html?q=Lorem&wcmode=
disabled
24. *
25. */
26. @Component(service = Servlet.class,
27.             property = {"sling.servlet.resourceTypes=trainingproject/components/structure/page
",
28.                         "sling.servlet.selectors=search"})
29.
30. public class SearchServlet extends SlingSafeMethodsServlet {
31.
32.     private static final long serialVersionUID = 3169795937693969416L;
33.
34.     @Override
35.     public final void doGet(final SlingHttpServletRequest request, final SlingHttpServletResponse
onse response)
36.             throws ServletException, IOException {
37.         response.setHeader("Content-Type", "application/json");
38.
39.         ArrayList resultArray = new ArrayList();
40.         Gson gson = new Gson();
41.
42.         try (ResourceResolver rr = request.getResourceResolver()) {
43.
```

```

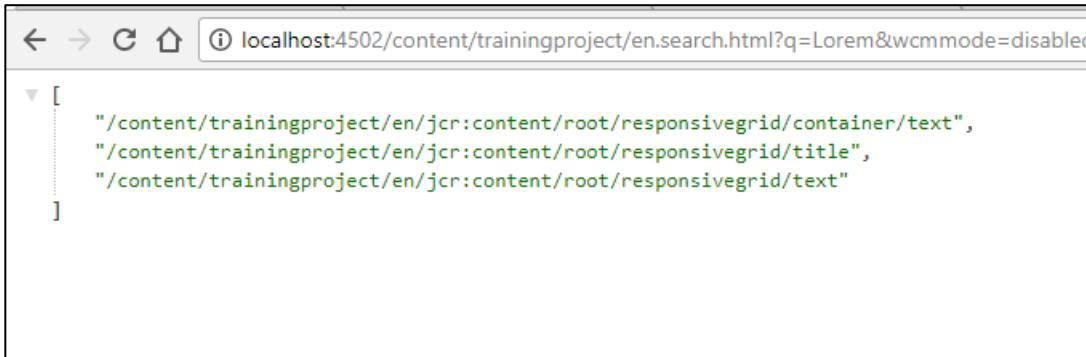
44.          // Path of the node, which triggers this servlet
45.          Resource requestingResource = request.getResource();
46.
47.          // Then adapt the resource tree into a page tree
48.
49.          PageManager pageManager = rr.adaptTo(PageManager.class);
50.
51.          // Now we can find the page, that was initiating the search
52.          String queryingPagePath = pageManager.getContainingPage(requestingResource).getPa
    th();
53.
54.          String queryTerm = (request.getParameter("q") != null) ? request.getParameter("q"
) : "";
55.
56.          String QUERY_STRING = "SELECT * " +
57.              "FROM [nt:unstructured] AS node " +
58.              "WHERE ISDESCENDANTNODE([" + queryingPagePath + "]) " +
59.              "and CONTAINS(node.*,'"+ queryTerm + "')";
60.
61.          Iterator<Resource> resourcesIterator = rr.findResources(QUERY_STRING, Query.JCR_S
QL2);
62.
63.          //Add the search results into the json array
64.          while (resourcesIterator.hasNext()) {
65.              Resource foundResource = resourcesIterator.next();
66.              resultArray.add(foundResource.getPath());
67.          }
68.
69.          String json = gson.toJson(resultArray);
70.
71.          response.getWriter().print(json);
72.          response.getWriter().close();
73.
74.      } catch (Exception e) {
75.          e.printStackTrace();
76.      }
77.  }
78. }
```

5. Right-click **training.core**, and perform **Run As > Maven install** to build the project. The project runs successfully.

6. Verify the bundle installed successfully, as shown:

```
[INFO] Installing C:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar to C:\Users\prpurush\.m2\repository\com\adobe\t
[INFO] Installing C:\Users\prpurush\Dev1\training\core\pom.xml to C:\Users\prpurush\.m2\repository\com\adobe\training.core\0.0.1-SNAPSHOT\t
[INFO]
[INFO] --- maven-bundle-plugin:3.3.0: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:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar) to http://local
[INFO] Bundle installed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 6.366 s
[INFO] Finished at: 2018-03-29T16:10:08-07:00
[INFO] Final Memory: 31M/360M
[INFO] -----
```

7. Go to <http://localhost:4502/content/trainingproject/en.search.html?q=Lorem&wcmmode=disabled> and verify the output, as shown:



# Event Handling in Adobe Experience Manager



## Introduction

Adobe Experience Manager (AEM) provides multiple ways of handling events to cater to changes in the repository. Some of the approaches includes JCR level with observation manager, Sling level with event handlers and jobs, AEM level with workflows and launchers. In AEM, Sling event handler approach is concise and easy to implement.

## Objectives

After completing this course, you will be able to:

- Explain OSGi events
- Describe Sling Jobs
- Handle OSGi events
- Create Job consumer
- Describe Sling scheduler
- Clean up nodes with a Sling scheduler
- Discuss event modeling
- Create a resource listener

# Sling Events

---

Events are used to trigger jobs or workflows. Apache Sling enables you to manage these events within your application. Apache Sling provides support for event handling jobs, and scheduling. Apache 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 or 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 the event listeners in the registry.

Various types of events can be handled by OSGi API:

- 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

## Listening to OSGi Events

The event mechanism is a mechanism that must:

- Implement the EventHandler interface.
- Subscribe by service registration by 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.

To listen to OSGi events in Apache 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, register an org.osgi.service.event.EventHandler service with ReplicationAction.Event:

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

or

- ```
6. @scr.property name="event.topics"  
7. 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.    }
```

## Exercise 1: Handle OSGi Events

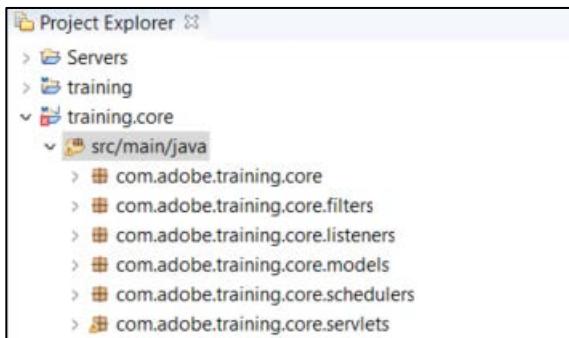
In this exercise, you will write a class to handle events for the replication action, which might be a publish or unpublish event. When the publish or unpublish action happens, this event handler will be triggered.

This exercise includes two tasks:

1. Create a new Java class
2. Deploy the project

### Task 1: Create a new Java class

1. Launch **Eclipse** by double-clicking the **Eclipse** shortcut on the desktop. The Eclipse Launcher opens.
2. Specify the Workspace as **C:\Workspace** and click **Launch**. The Eclipse Workspace opens.
3. In Project Explorer, navigate to **training.core > src/main/java**, as shown:

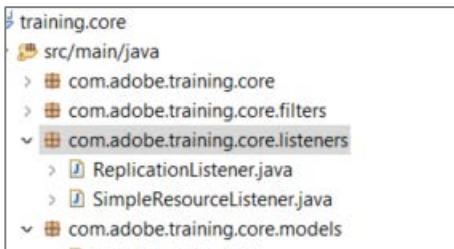


4. Copy the file **ReplicationListener.java** from **Exercise\_Files/11\_Event\_Handling/**.



**Note:** The code is provided as part of the Exercise\_Files under /Exercise\_Files/11\_Event\_Handling/. Copy and paste the file from the exercise files referenced to ReplicationListener.java to Eclipse. Please do not copy the code from this exercise book. The code in the student guide is for illustrative purposes only.

5. In Eclipse, right-click **com.adobe.training.core.listeners** and paste the file. The **ReplicationListener.java** class is created, as shown:



6. Double-click **ReplicationListener.java**. The file opens in the editor.

```
1. package com.adobe.training.core.listeners;
2.
3. import java.util.HashMap;
4.
5. import org.osgi.service.component.annotations.Component;
6. import org.osgi.service.component.annotations.Reference;
7. import org.osgi.framework.Constants;
8. import org.osgi.service.event.Event;
9. import org.osgi.service.event.EventHandler;
10. import org.osgi.service.event.EventConstants;
11. import org.slf4j.Logger;
12. import org.slf4j.LoggerFactory;
13. import com.day.cq.replication.ReplicationAction;
14. import com.day.cq.replication.ReplicationActionType;
15.
16. import org.apache.sling.event.jobs.JobManager;
17.
18. @Component( immediate = true,
19.             service = EventHandler.class,
20.             property = {Constants.SERVICE_DESCRIPTION + "=Replication Listener kicks off a jo
b",
21.                         EventConstants.EVENT_TOPIC + "=" + 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 46
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.         logger.info("PATH: " + action.getPath().toString());
36.         if (action.getType().equals(ReplicationActionType.ACTIVATE)) {
37.
38.             if (action.getPath() != null)
39.             {
40.                 try {
41.                     // Create a properties map that contains things we want to pass through t
he job

```

```

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()+"' was added to the Job Manager");
47.
48.             } catch (Exception e) {
49.                 logger.error("===== ERROR CREATING JOB : NO PAYLOAD WAS DEFINED")
50.                 ;
51.             }
52.         }
53.     }
54. }
55. }
```

7. Examine the method **handleEvent()** and observe the log message, as shown:

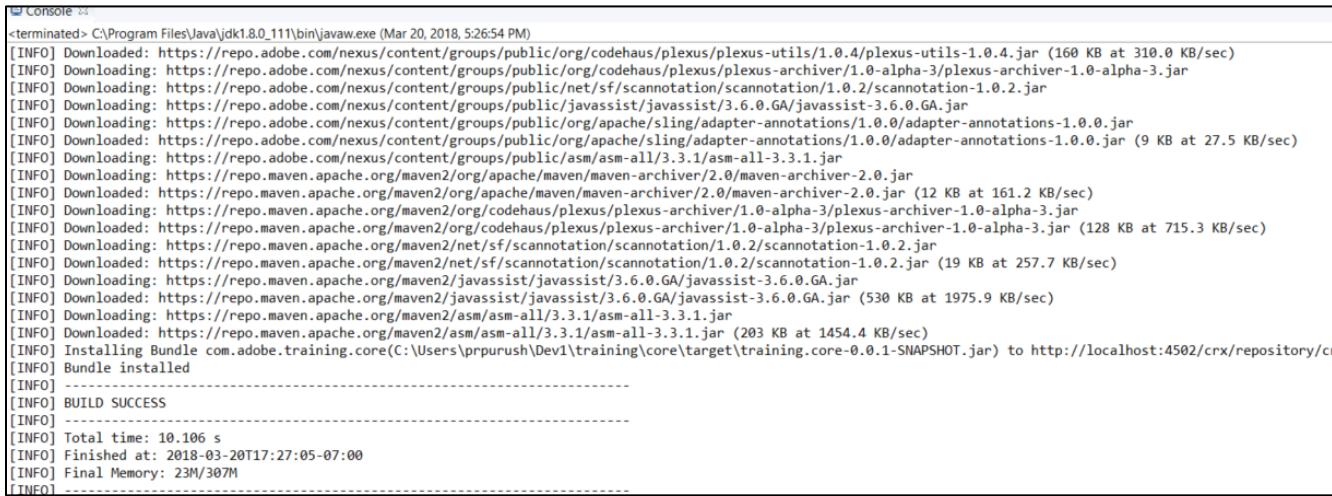
```

public void handleEvent(final Event event) {
    ReplicationAction action = ReplicationAction.fromEvent(event);
    logger.info("PATH: " + action.getPath().toString());
    if (action.getType().equals(ReplicationActionType.ACTIVATE)) {

        if (action.getPath() != null)
        {
            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");
            }
        }
    }
}
```

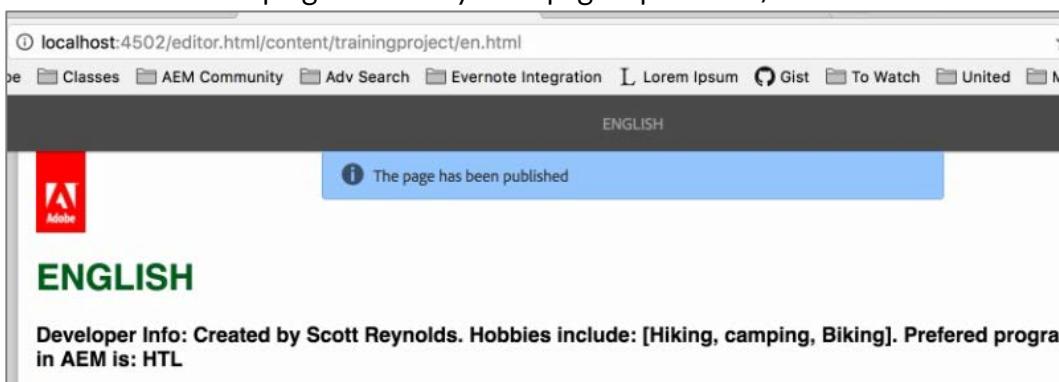
## Task 2: Deploy and test the class

1. In Project Explorer, right-click **training.core** and choose **Run As > Maven install**. The build starts.
2. Verify the bundle installed successfully, as shown:



```
<terminated> C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Mar 20, 2018, 5:26:54 PM)
[INFO] Downloaded: https://repo.adobe.com/nexus/content/groups/public/org/codehaus/plexus/plexus-utils/1.0.4/plexus-utils-1.0.4.jar (160 KB at 310.0 KB/sec)
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar
[INFO] Downloaded: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar (9 KB at 27.5 KB/sec)
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/asm/asm-all/3.3.1/asm-all-3.3.1.jar
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.0/maven-archiver-2.0.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.0/maven-archiver-2.0.jar (12 KB at 161.2 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar (128 KB at 715.3 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar (19 KB at 257.7 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar (530 KB at 1975.9 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/asm/asm-all/3.3.1/asm-all-3.3.1.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/asm/asm-all/3.3.1/asm-all-3.3.1.jar (203 KB at 1454.4 KB/sec)
[INFO] Installing Bundle com.adobe.training.core(C:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar) to http://localhost:4502/crx/repository/cr
[INFO] Bundle installed
-----
[INFO] BUILD SUCCESS
-----
[INFO]
[INFO] Total time: 10.106 s
[INFO] Finished at: 2018-03-20T17:27:05-07:00
[INFO] Final Memory: 23M/307M
[INFO]
```

3. Log in to AEM, navigate to **Sites**, select **TrainingProject Site**, and click the thumbnail on **English**. The **English** page is selected.
4. Click **Edit (e)** in the action bar. The **English** page opens.
5. Click the Page Information icon at the top left and select **Publish Page**. The **Publish** page appears with **All Assets** and **asset.jpg**.
6. Click **Publish** at the top right and verify if the page is published, as shown:



7. On your desktop, navigate to `/crx-quickstart/logs/` and open `project-trainingproject.log`.

8. Scroll down and examine the log message, as shown:

```
[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* [0:0:0:0:0:0:1] [148558327994] 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: HTML
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
```

This indicates your class is successful.

9. Navigate to the **AEM Web Console** and Select **OSGi > Events** from the menu. The Events are listed.

10. Notice all the events in the event log, as shown:

Received	Event Topic	Event Properties
4/5/2018 1:17:20 PM	org/apache/sling/api/resource/Resource/REMOVED	path /var/eventing/jobs/as event.topics org/apache/sling/api/ userid sling-event
4/5/2018 1:17:16 PM	org/osgi/framework/ServiceEvent/MODIFIED	Service (id=50, objectClass=org.osg)
4/5/2018 1:16:49 PM	org/apache/sling/event/notification/job/FAILED	:time 0 replicationContent Replication event.job.retrydelay 60000 cq:type ACTIVATE slingevent:application 7181f121- jcr:created java.util.G slingevent:created java.util.G event.job.queueName com_day_0 event.job.createdTime java.util.G

11. Search for **training** and locate the following event entry in the **AEM Web Console**, as shown:

- `event.topics=com/day/cq/replication`
- `paths=/content/trainingproject/en`
- `type=ACTIVATE`

```
resourceType sling:Folder
modificationDate java.util.GregorianCalendar[time=1522959168356,areFieldsSet=true,areAllFieldsSet=true,lenient=true,zone=sun.util.calendar.ZoneInfo[id="America/Los
[Id=America/Los_Angeles,offset=-28800000,dstSavings=3600000,useDaylight=true,startYear=0,startMode=3,startMonth=2,startDay=8,startDayOfWeek=1,second=0]
event.topics com/day/cq/replication
paths /content/trainingproject/en
type ACTIVATE
```

# Working with Sling Scheduler

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. The Sling Scheduler makes use of the open source Quartz library.

The scheduler can be used in two ways, by registering the job through the scheduler API and by leveraging the whiteboard pattern that is supported by the scheduler. 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.

# Sling Jobs

A job is a special event that must be processed exactly once. The Sling Jobs Processing adds some overhead, so in some cases it might be better to use just the Commons Scheduler Service or the Commons Thread Pool for asynchronous execution of code. While older versions of the job handling were based on sending and receiving events through the OSGi event admin, newer versions provide enhanced support through special Java interface.

## Scheduling Jobs

The following sections of scheduling jobs describe various methods:

- Scheduling at periodic times

Instead of specifying a Quartz expression for the time interval, in simpler cases, you 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);`

## 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. 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

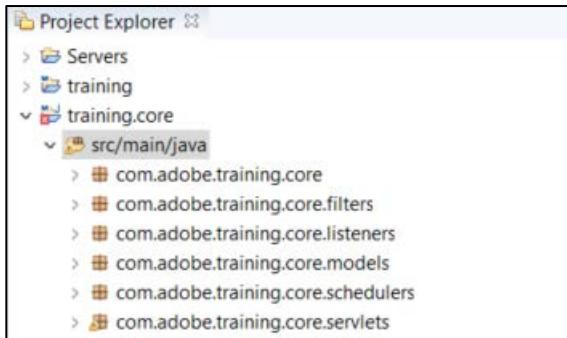
Event handling can be done at any of the following levels:

- JCR level, with observation
- Sling level, with event handlers and jobs
- AEM level, with workflows and launchers

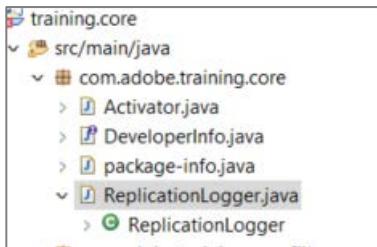
## Exercise 2: Create a job consumer for a topic

In the last exercise, you added a Sling Job with the JobManager. In this exercise, you will process the Sling Job that was put in the Job Queue.

1. In Project Explorer, navigate to **training.core > src/main/java**, as shown:



2. Copy the file **ReplicationLogger.java** from **/Exercise\_Files/11\_Event\_Handling/**.
3. In Eclipse, right-click **com.adobe.training.core** and paste the file. The **ReplicationLogger.java** class is created, as shown:



 **Note:** The code is provided as part of the Exercise\_Files under **/Exercise\_Files/11\_Event\_Handling/**. Copy and paste the file from the exercise files referenced to **ReplicationLogger.java** to Eclipse. Please do not copy the code from this exercise book. The code in the student guide is for illustrative purposes only.

4. Double-click **ReplicationLogger.java**. The file opens in editor, as shown:

```

1. package com.adobe.training.core;
2.
3. import org.osgi.service.component.annotations.Component;
4. import org.osgi.service.component.annotations.Reference;
5.
6. import org.apache.sling.api.resource.LoginException;
7. import org.apache.sling.api.resource.ResourceResolver;
8. import org.apache.sling.api.resource.ResourceResolverFactory;
9. import org.apache.sling.event.jobs.Job;
10.
11. import org.apache.sling.event.jobs.consumer.JobConsumer;
12.
13. import org.slf4j.Logger;
14. import org.slf4j.LoggerFactory;
15. import com.day.cq.wcm.api.Page;
16. import com.day.cq.wcm.api.PageManager;
17.
18. import java.util.HashMap;
19. import java.util.Map;
20.
21. @Component(service = JobConsumer.class,
22.             property = JobConsumer.PROPERTY_TOPICS + "=com/adobe/training/core/replicationjob"
23. )
24.
25. public class ReplicationLogger implements JobConsumer {
26.     private final Logger logger = LoggerFactory.getLogger(getClass());
27.
28.     @Reference
29.     private ResourceResolverFactory resourceResolverFactory;
30.
31.     @Override
32.     public JobResult process(final Job job) {
33.
34.         final String pagePath = job.getProperty("PAGE_PATH").toString();
35.
36.         Map<String, Object> serviceParams = new HashMap<>();
37.         serviceParams.put(ResourceResolverFactory.SUBSERVICE, "training");
38.
39.         try (ResourceResolver resourceResolver = resourceResolverFactory
40.              .getServiceResourceResolver(serviceParams)) {
41.             // Create a Page object to log its title
42.             final PageManager pm = resourceResolver.adaptTo(PageManager.class);
43.
44.             if (pagePath != null) {
45.                 final Page page = pm.getContainingPage(pagePath);
46.                 logger.info("***** ACTIVATION OF PAGE : {}", page.getTitle());
47.
48.             }
49.             resourceResolver.close();
50.         } catch (LoginException e) {
51.             logger.error("Exception with ResourceResolver: ", e);
52.             e.printStackTrace();
53.             return JobConsumer.JobResult.FAILED;
54.         }
55.     }

```

5. Right-click **training.core** and select **Run As > Maven install**. The project is built.
6. Verify that the component is available in AEM at <http://localhost:4502/system/console/components> and search for the component by its name, as shown:

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

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.

7. To publish (activate) a page, select the page in AEM, and select **Quick Publish**. The page is published.
8. Navigate to **/crx-quickstart/logs/** and open **project-trainingproject.log**.
9. You should see logged messages giving the titles of pages as they are activated, as shown:

```
com.adobe.training.core.Service [titleservlet,9749, [javax.servlet.Servlet]] ServiceEvent UNREGISTERING
com.adobe.training.core.BundleEvent STOPPING
com.adobe.training.core.Service [9711, [org.osgi.service.cm.ManagedService]] ServiceEvent UNREGISTERING
com.adobe.training.core.BundleEvent STOPPED
com.adobe.training.core.BundleEvent UNRESOLVED
com.adobe.training.core.BundleEvent UPDATED
com.adobe.training.core.BundleEvent RESOLVED
com.adobe.training.core.BundleEvent ADDED
com.adobe.training.core.Service [1104, [org.osgi.service.cm.ManagedService]] ServiceEvent REGISTERED
com.adobe.training.core.Activator #####@#Bundle Started#####
com.adobe.training.core.Service [com.adobe.training.core.ReplicationLogger,11105, [org.apache.sling.event.jobs.consumer.JobConsumer]] ServiceEvent REGISTERED
com.adobe.training.core.Service [com.adobe.training.core.filters.LoggingFilter,11106, [javax.servlet.Filter]] ServiceEvent REGISTERED
com.adobe.training.core.Service [com.adobe.training.core.listeners.ReplicationListener,11108, [org.osgi.service.event.EventHandler]] ServiceEvent REGISTERED
com.adobe.training.core.Service [com.adobe.training.core.listeners.SimpleResourceListener,11109, [org.osgi.service.event.EventHandler]] ServiceEvent REGISTERED
com.adobe.training.core.Service [com.adobe.training.core.schedulers.SimpleScheduledTask,11118, [java.lang.Runnable]] ServiceEvent REGISTERED
com.adobe.training.core.Service [TrainingDeveloperInfo,11119, [com.adobe.training.core.DeveloperInfo]] ServiceEvent REGISTERED
com.adobe.training.core.Service [ServletResourceProvider for Servlets at /libs/trainingproject/components/structure/page/txt.GET.servlet],11121, [org.apache.sling.spi.resource.provider.ResourceProvider]
com.adobe.training.core.Service [com.adobe.training.core.servlets.SimpleServlet,11120, [javax.servlet.Servlet]] ServiceEvent REGISTERED
com.adobe.training.core.Service [ServletResourceProvider for Servlets at /bin/trainingproject/titleServlet, /bin/trainingproject/titleServlet.servlet],11123, [org.apache.sling.spi.resource.provider.ResourceProvider]
com.adobe.training.core.Service [ServletResourceProvider for Servlets at /bin/trainingproject/titleServlet, /bin/trainingproject/titleServlet.servlet],11151, [org.apache.sling.spi.resource.provider.ResourceProvider]
com.adobe.training.core.Service [TrainingTitleServlet,11122, [javax.servlet.Servlet]] ServiceEvent REGISTERED
com.adobe.training.core.BundleEvent STARTED
c6f695b3-428e-4aa4-a84d-ad65eb43368f-(apache-sling-job-thread-pool-15-<main queue>(com/adobe/training/core/replicationjob)) com.adobe.training.core.ReplicationLogger **** ACTIVATION OF PAGE
c6f695b3-428e-4aa4-a84d-ad65eb43368f-(apache-sling-job-thread-pool-17-<main queue>(com/adobe/training/core/replicationjob)) com.adobe.training.core.ReplicationLogger **** ACTIVATION OF PAGE
be.training.core.listeners.ReplicationListener PATH /content/trainingproject/en
be.training.core.listeners.ReplicationListener *****Topic: 'com/adobe/training/core/replicationjob' with payload: '/content/trainingproject/en' was added to the Job Manager
c6f695b3-428e-4aa4-a84d-ad65eb43368f-(apache-sling-job-thread-pool)-25-<main queue>(com/adobe/training/core/replicationjob)) com.adobe.training.core.ReplicationLogger **** ACTIVATION OF PAGE
```

10. Go to <http://localhost:4502/system/console/slingevent>. The events are displayed.

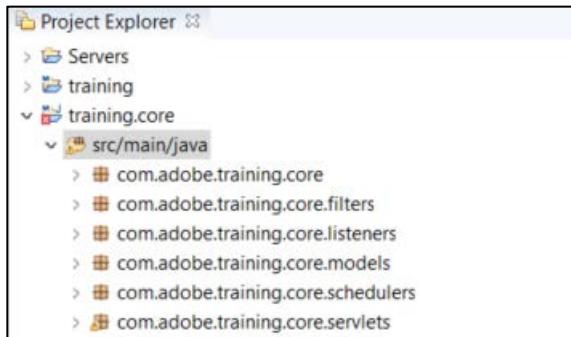
11. Search for **replicationjob** using browser search. The **Topic Statistics** for **replicationjob** is returned.

Average Waiting Time	42 ms
<b>Topic Statistics: com/adobe/training/core/<b>replicationjob</b></b>	
Last Activated	16:13:21:756 2018-Apr-05
Last Finished	16:13:21:769 2018-Apr-05
Finished Jobs	3
Failed Jobs	0
Cancelled Jobs	1
Processed Jobs	4
Average Processing Time	12 ms
Average Waiting Time	152 min 44 secs

## Exercise 3: Clean up nodes with a Sling scheduler

In this exercise, you will write a job that periodically deletes the temporary nodes in the repository. The configuration will be stored in a configuration node, which you can see and edit in the AEM Web Console.

1. In Project Explorer, navigate to **training.core > src/main/java**, as shown:

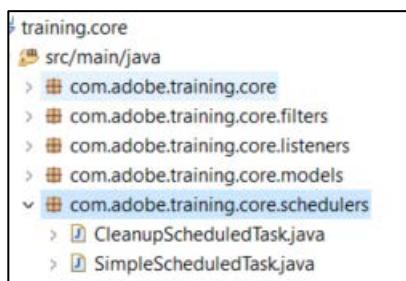


2. Copy the file **CleanupScheduledTask.java** from [/Exercise\\_Files/11\\_Event\\_Handling/](#).



**Note:** The code is provided as part of the Exercise\_Files under /Exercise\_Files/11\_Event\_Handling/. Copy and paste the file from the exercise files referenced to CleanupScheduledTask.java to Eclipse. Please do not copy the code from this exercise book. The code in the student guide is for illustrative purposes only.

3. Right-click **com.adobe.training.core.schedulers** and paste the file. The **CleanupScheduledTask.java** class is created, as shown:



4. Double-click **CleanupScheduledTask.java**. The file opens in editor, as shown:

```

1. package com.adobe.training.core.schedulers;
2.
3. import org.apache.sling.api.resource.*;
4. import org.osgi.service.component.annotations.Component;
5. import org.osgi.service.component.annotations.Activate;
6. import org.osgi.service.component.annotations.Modified;
7. import org.osgi.service.component.annotations.Reference;
8. import org.osgi.service.metatype.annotations.AttributeDefinition;
9. import org.osgi.service.metatype.annotations.AttributeType;
10. import org.osgi.service.metatype.annotations.Designate;
11. import org.osgi.service.metatype.annotations.ObjectClassDefinition;
12.
13. import org.slf4j.Logger;
14. import org.slf4j.LoggerFactory;
15.
16. import java.util.HashMap;
17. import java.util.Map;
18.
19. @Component(service = Runnable.class, immediate = true)
20.
21. @Designate(ocd = CleanupScheduledTask.Configuration.class, factory=true)
22. public class CleanupScheduledTask implements Runnable {
23.     private final Logger logger = LoggerFactory.getLogger(getClass());
24.
25.     @Reference
26.     private ResourceResolverFactory resolverFactory;
27.
28.     private String cleanupPath;
29.
30.     @Activate
31.     protected void activate(Configuration config) {
32.         cleanupPath = config.cleanupPath();
33.     }
34.
35.     @Modified
36.     protected void modified(Configuration config) {
37.         logger.info("Configuration modified");
38.     }
39.
40.     @ObjectClassDefinition(name = "Training Cleanup Service")
41.     public @interface Configuration {
42.         @AttributeDefinition(
43.             name = "Cleanup path",
44.             description = "Path to the node which the scheduler removes at a configured i
        nterval",
45.             type = AttributeType.STRING
46.         )
47.         String cleanupPath() default "/mypathtraining";
48.
49.         @AttributeDefinition(
50.             name = "Cron like expression",
51.             description = "Run every so often as defined in the cron job expression.",
52.             type = AttributeType.STRING
53.         )
54.         String scheduler_expression() default "0/30 * * * * ?";
55.     }
56.
57.     @Override
58.     public void run() {

```

```

59.         //allows us to get the resourceResolver based on the current session
60.         Map<String, Object> params = new HashMap<>();
61.         params.put(ResourceResolverFactory.SUBSERVICE, "training");
62.
63.         logger.info("!!!Cleanup Task checking: {}", cleanupPath);
64.
65.         try (ResourceResolver resourceResolver = resolverFactory.getServiceResourceResolver(p
66.             arams)){
67.
68.             Resource resource = resourceResolver.getResource(cleanupPath);
69.
70.             if (resource != null) {
71.                 resourceResolver.delete(resource);
72.                 resourceResolver.commit();
73.                 logger.info("!!!node deleted");
74.             }
75.         } catch (LoginException e) {
76.             logger.error("!!!resource does not exist", e);
77.         } catch (PersistenceException e) {
78.             e.printStackTrace();
79.         }
80.     }
81. }
```

5. Right-click **training.core**, and select **Run As > Maven install**. The project is built.
6. Refresh your browser where you are working with your AEM instance. The page is reloaded.
7. On your desktop, navigate to **/crx-quickstart/logs/** and open **project-trainingproject.log**.
8. Inspect the log output (**project-trainingproject.log**), as shown:

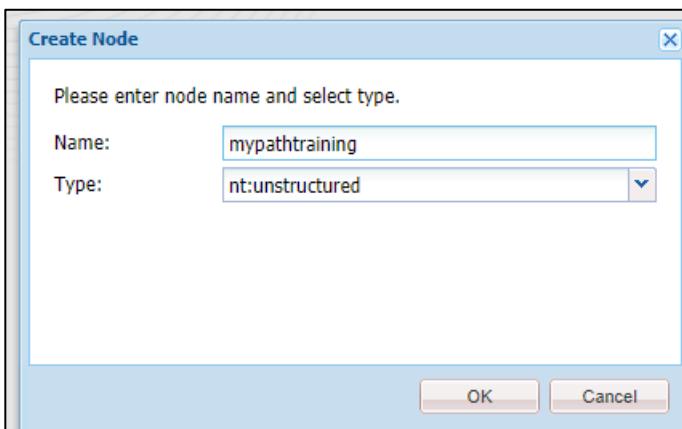
```

-22c1-4788-8d6b-92570f197b3c.7021] com.adobe.training.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mypathtraining
```



**Note:** Now you know the scheduler is running. However, you need to test to see if it will delete, or cleanup **/mypathtraining** nodes.

9. In CRXDE Lite, right-click the root node (/), and select **Create > Create Node** named **mypathtraining**, as shown:



10. Click **OK**. The node is created.
11. Click **Save All** to save the changes and wait 30 seconds.
12. Refresh **CRXDE Lite**. The node gets deleted.
13. Verify it is deleted by the cleanup service in the custom log file **project-trainingproject.log**, as shown:

```
ining.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mypathtraining
ining.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mypathtraining
ining.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mycustompath
ining.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mypathtraining
ining.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mypathtraining
ining.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mycustompath
ining.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mypathtraining
ining.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mypathtraining
ining.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mycustompath
ining.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mypathtraining
ining.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mycustompath
ining.core.schedulers.CleanupScheduledTask !!!node deleted
```

14. In the AEM Web Console, go to **OSGi > Configuration** (<http://localhost:4502/system/console/configMgr>) and search for: **Training Cleanup Service**, as shown:



Notice the configuration has a + icon next to it. This means that it is a configuration factory, and you can have multiple instances of this service.

15. Click on **Training Cleanup Service** to view the details, as shown:

Training Cleanup Service	
Cleanup path	/mypathtraining ⚠ Path to the node which the scheduler removes at a configured interval (cleanupPath)
Cron like expression	0/30 * * * * ? ⚠ Run every so often as defined in the cron job expression. (scheduler.expression)
Configuration Information	
Persistent Identity (PID)	[Temporary PID replaced by real PID upon save]
Factory Persistent Identifier (Factory PID)	com.adobe.training.core.schedulers.CleanupScheduledTask
Configuration Binding	Unbound or new configuration

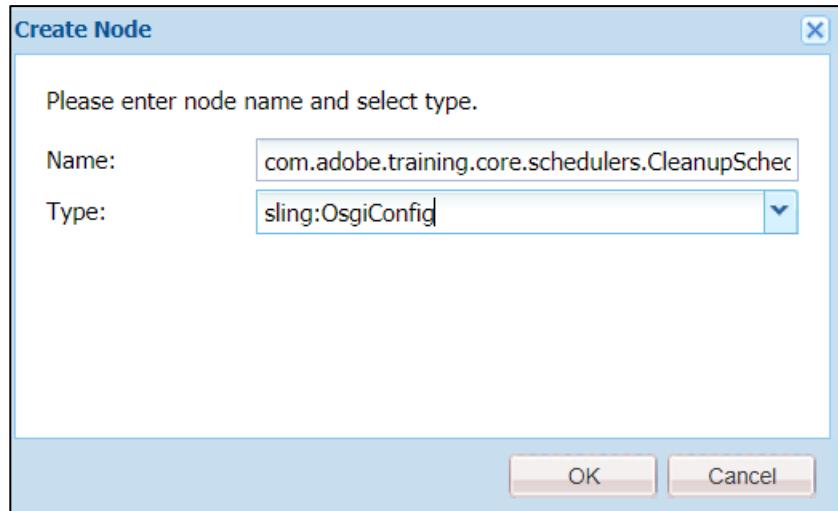


**Note:** When you open the config, you will notice the PID is not set, but rather a Factory PID is provided. This means that the actual PID is assigned when a configuration is made for this factory.

16. In CRXDE Lite, navigate to /apps/trainingproject/config.author.

17. Right-click **config.author** and select **Create > Create Node**, and enter the following values, as shown:

- Name: **com.adobe.training.core.schedulers.CleanupScheduledTask-mycustompath**
- Select **sling:OsgiConfig** from **Type** drop-down menu.



**Note:** mycustompath was added as a unique identifier (it can be anything) to the PID. This is needed to make an instance of the config factory.

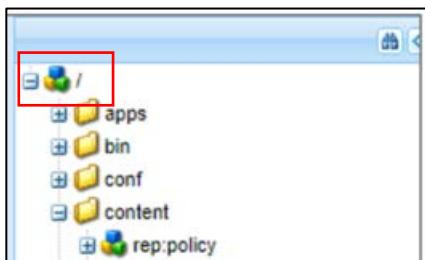
18. To the new node, add these properties, as shown:

- Name: **cleanupPath**  
Type: **(String)**  
Value: **/mycustompath**
- Name: **scheduler.expression**  
Type: **(String)**  
Value: **0/30 \* \* \* ?**

Properties			Access Control	Replication	Console	Build Info
Name	Type	Value				
1 cleanupPath	String	/mycustompath				
2 jcr:created	Date	2018-04-06T20:16:03.724-07:00				
3 jcr:createdBy	String	admin				
4 jcr:primaryType	Name	sling:OsgiConfig				
5 scheduler.expression	String	0/30 * * * ?				

19. Click **Save All** to save the changes.

20. To test your configuration, right-click the root node (/), as shown and select **Create > Create Node**.



21. Enter the following values for the node:

- Name: **mycustompath**
- Type: **nt:unstructured**

22. Click **Save All**. The **mycustompath** node is created.

23. Wait 30 seconds and refresh **CRXDE Lite**. Observe the node is deleted.

24. Observe the node being deleted in the logs ~/**crx-quickstart/logs/project-trainingproject.log**, as shown:

```
com.adobe.training.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mycustompath  
com.adobe.training.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mypathtraining  
com.adobe.training.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mypathtraining  
com.adobe.training.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mycustompath  
com.adobe.training.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mypathtraining  
com.adobe.training.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mypathtraining  
com.adobe.training.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mycustompath  
com.adobe.training.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mypathtraining  
com.adobe.training.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mycustompath  
com.adobe.training.core.schedulers.CleanupScheduledTask !!!Cleanup Task checking: /mypathtraining  
com.adobe.training.core.schedulers.CleanupScheduledTask !!!node deleted
```

## Extra Credit: Add more Cleanup paths

1. To add more cleanup paths, copy and paste the OSGi config node and rename the unique identifier, for example:
  - a./apps/trainingproject/config.author/com.adobe.training.core.schedulers.CleanupScheduledTask-mysecondpath
  - b.Update the cleanupPath property:  
  
cleanupPath (String) /mysecondpath
2. Save the changes.
3. Observe how both the paths are not being cleaned up.

# Event Modeling

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: `EventEventIterator.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

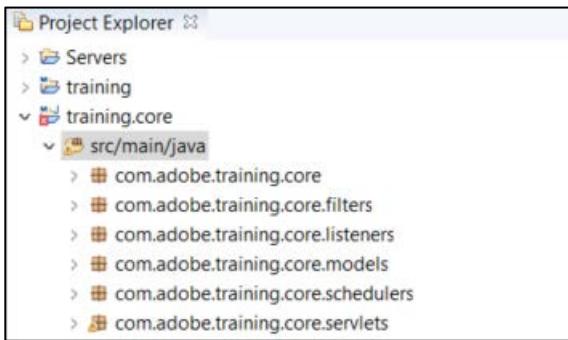
You can set the user data using:

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

## Exercise 4: Create a resource listener

In this exercise, you will create a resource listener that listens for resource changes made to the jcr:title property of a page, and displays the notification accordingly.

1. In Project Explorer, navigate to **training.core > src/main/java**, as shown:



2. Copy the file **TitlePropertyListener.java** from **/Exercise\_Files/11\_Event\_Handling**.



**Note:** The code is provided as part of the Exercise\_Files under /Exercise\_Files/11\_Event\_Handling/. Copy and paste the file from the exercise files referenced to TitlePropertyListener.java to Eclipse. Please do not copy the code from this student guide. The code in the student guide is for illustrative purposes only.

3. Right-click **com.adobe.training.core.listeners** and paste the file. The **TitlePropertyListener.java** class is created, as shown:



4. Double-click **TitlePropertyListener.java**. The file opens in editor, as shown:

```

1. package com.adobe.training.core.listeners;
2.
3. import org.apache.sling.api.resource.*;
4. import org.osgi.service.component.annotations.Component;
5. import org.osgi.service.component.annotations.Reference;
6.
7. import org.apache.sling.api.resource.observation.ResourceChange;
8. import org.apache.sling.api.resource.observation.ResourceChangeListener;
9.
10. import java.util.HashMap;
11. import java.util.List;
12. import java.util.Map;
13.
14. import org.slf4j.Logger;
15. import org.slf4j.LoggerFactory;
16.
17. @Component( immediate = true,
18.             name = "Title Property Listener",
19.             property = {"resource.paths=/content/trainingproject/fr",
20.                         "resource.change.types=CHANGED"} )
21.
22. public class TitlePropertyListener implements ResourceChangeListener{
23.
24.     private final Logger logger = LoggerFactory.getLogger(getClass());
25.
26.     @Reference
27.     private ResourceResolverFactory resourceResolverFactory;
28.
29.     public void onChange(List<ResourceChange> changes) {
30.
31.         for (final ResourceChange change : changes) {
32.
33.             logger.info("Change type: {}", change);
34.
35.             Map<String, Object> serviceParams = new HashMap<String, Object>(){{
36.                 put(ResourceResolverFactory.SERVICE_NAME, "training");
37.             }};
38.             try (ResourceResolver rr = resourceResolverFactory.getServiceResourceResolver(serviceParams)) {
39.                 logger.error("Repository login success");
40.
41.                 String JCR_TITLE_PROP = "jcr:title";
42.
43.                 Resource changedRes = rr.getResource(change.getPath());
44.
45.                 String titleProperty = (changedRes != null) ? changedRes.getValueMap().get(JCR_TITLE_PROP, String.class) : "";
46.
47.                 if (titleProperty != null && !titleProperty.isEmpty() && !titleProperty.endsWith("!")) {
48.                     ModifiableValueMap modChangedResource = changedRes.adaptTo(ModifiableValueMap.class);
49.                     modChangedResource.put(JCR_TITLE_PROP, titleProperty + "!");
50.                     rr.commit();
51.                     logger.info("*****Property updated: {}", change.getPath());
52.                 }
53.             } catch(PersistenceException e ){
54.                 logger.error("Failed to write the resource change.");
55.             }
56.         }
57.     }
58. }
```

```

55.         } catch(LoginException e ){
56.             logger.error("Repository login failed.");
57.         }
58.     }
59. }
60. }
```

5. Right-click **training.core** and select **Run As > Maven install**. The project is built.

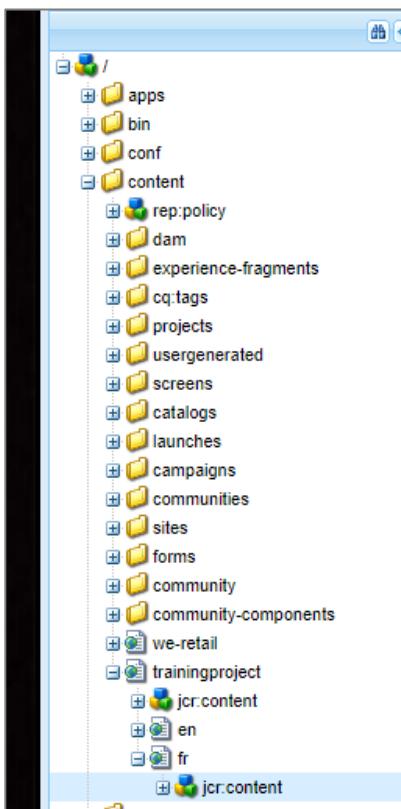
6. Verify that the component is available in AEM at: <http://localhost:4502/system/console/components>, as shown:

The screenshot shows the AEM System Console Components view. A component named "Title Property Listener" (PID: 3249) is selected. The component details are displayed in a table:

	Value
Bundle	com.adobe.training.core (547)
Implementation Class	com.adobe.training.core.listeners.TitlePropertyListener
Default State	enabled
Activation	immediate
Configuration Policy	optional
Service Type	singleton
Services	org.apache.sling.api.resource.observation.ResourceChangeListener
PID	Title Property Listener
Reference resourceResolverFactory	Satisfied Service Name: org.apache.sling.api.resource.ResourceResolverFactory Cardinality: 1..1 Policy: static Policy Option: reluctant Bound Service ID 2007 (Apache Sling Resource Resolver Factory)
Properties	component.id = 3249 component.name = Title Property Listener resource.change.types = CHANGED resource.paths = /content/trainingproject/fr

Below the component details, another component "TrainingDeveloperInfo" (PID: 3254) is listed.

7. In CRXDE Lite, navigate to /apps/content/trainingproject/fr/jcr:content, as shown:



8. Change the value of the property **jcr:title** to **Test**, as shown:

Properties			Access Control	Replication	Console	Build Info
Name	Type	Value				
3 cq:template	String	/conf/trainingproject/settings/wcm/templates/content-page				
4 jcr:created	Date	2018-04-05T21:22:31.890-07:00				
5 jcr:createdBy	String	admin				
6 jcr:primaryType	Name	cq:PageContent				
7 jcr:title	String	Test				
8 pageTitle	String	Nouveau Project				
9 sling:resourceType	String	trainingproject/components/structure/page				

9. Click **Save All**. The changes are saved.

10. Refresh CRXDE Lite to see if ! is added by **com.adobe.training.core.TitlePropertyListener** component with the value **Test**, as shown:

Properties		Access Control	Replication	Console	Build Info
	Name ▾	Type	Value		
3	cq:template	String	/conf/trainingproject/settings/wcm/templates/content-page		
4	jcr:created	Date	2018-04-05T21:22:31.890-07:00		
5	jcr:createdBy	String	admin		
6	jcr:primaryType	Name	cq:PageContent		
7	jcr:title	String	Test!		
8	pageTitle	String	Nouveau Project		
9	sling:resourceType	String	trainingproject/components/structure/page		

11. Go to the French page in our **TrainingProject** Site,  
<http://localhost:4502/editor.html/content/trainingproject/fr.html>.

12. Click to the Page information icon > **Open Properties**. The **Page Properties** window (with the name **Test!**) opens.

13. Change the **Title**, as shown:

The screenshot shows the 'Page Properties' dialog box for a page named 'Test!'. The 'Title and Tags' section is expanded, showing a 'Title \*' input field containing 'My Page'. Below it is a 'Tags' input field with a single tag selected. There is also a checked checkbox labeled 'Hide in Navigation'. The 'More Titles and Description' section is also visible, showing a 'Page Title' input field containing 'Nouveau Project'.

14. Click **Save & Close**. The page is reloaded, as shown:

The screenshot shows the AEM Site Structure interface. The left sidebar lists various site components: Screens, Campaigns, Community Sites, We.Retail, and the current site, TrainingProject Site. The 'Screens' component is expanded, showing two language variants: English (en) and French (fr), each represented by a thumbnail and a title. The other components are collapsed, indicated by a right-pointing arrow icon.

## References

For more information on the events in the AEM Web Console:

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

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

# Deep Dive into Adobe Experience Manager APIs



## Introduction

APIs are at the highest layer of the Adobe Experience Manager (AEM) architecture stack, and are most closely related to various business tasks a typical content producer would create in AEM. You can use APIs in AEM to develop a custom data importer service that helps import data from a site.

To address some business requirements, you may need to import external data into your AEM site. For example, you can import data from an external social media site. You can also use AEM APIs to create a workflow for AEM, and then programmatically invoke the workflow.

## Objectives

After completing this course, you will be able to:

- Describe polling importers
- Create polling importers
- Programmatically create an AEM page
- Create AEM pages based on a CSV file input
- Describe AEM projects
- Create a project and project templates
- Execute the workflow
- Build and test a workflow
- Customize the process step in workflow

## Polling Importers

---

The polling 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 represents the content of the remote resource.

The polling importer is used to support:

- The auto-blogging feature, which automatically creates blog posts from an external RSS or Atom feed
- iCalendar subscriptions, which automatically creates calendar events from an external ICS file or subscribes to or from other calendars

The polling importer is found in the WCM Administrative interface under Tools > **Importers** > **Feed Importer**. You can double-click the Feed Importer node to configure standard polling importers for RSS, Atom, Calendar, IMAP, and POP3.

To implement customized polling importer, use the interface, `com.day.cq.pollingimporter.Importer` and develop an OSGi bundle.

## Exercise 1: Create a polling importer

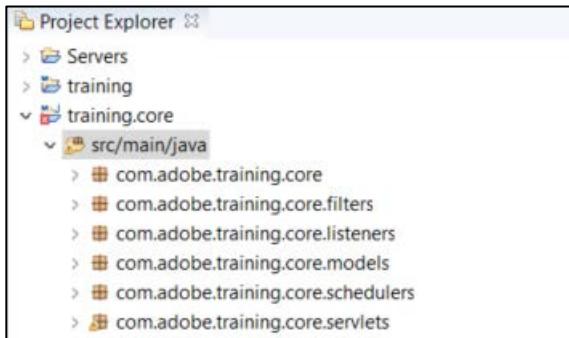
In this exercise, you will create a polling importer to import stock information to JCR.

This exercise includes two tasks:

1. Create a polling importer
2. Deploy and test the polling importer

### Task 1: Create a polling importer

1. Double-click the **Eclipse** shortcut on the desktop. The Eclipse Launcher opens.
2. Specify the workspace as **C:\Workspace**, and click **Launch**. The Eclipse Workspace opens.
3. In Project Explorer, navigate to **training.core > src/main/java**, as shown:



4. Copy the file **StockDataImporter.java** from **/Exercise\_Files/12\_Deep\_Dive\_into\_AEM\_APIs/**.



**Note:** The code is provided as part of the Exercise\_Files under /Exercise\_Files/12\_Deep\_Dive\_into\_AEM\_APIs/. Copy and paste the file from the exercise files referenced to **StockDataImporter.java** to Eclipse. Please do not copy the code from this student guide. The code in the student guide is for illustrative purposes only.

5. In Eclipse, right-click **com.adobe.training.core** and paste the file. The **StockDataImporter.java** class is created, as shown:



6. Double-click **StockDataImporter.java**. The file opens in editor, as shown:

```
56. package com.adobe.training.core;
57.
58. import java.io.IOException;
59. import java.io.InputStream;
60. import java.io.InputStreamReader;
61. import java.net.MalformedURLException;
62. import java.net.URL;
63. import java.time.Instant;
64. import java.time.LocalDateTime;
65. import java.time.ZoneId;
66. import java.time.ZonedDateTime;
67. import java.time.format.DateTimeFormatter;
68.
69. import javax.jcr.Node;
70. import javax.jcr.RepositoryException;
71. import javax.jcr.Session;
72. import javax.net.ssl.HttpsURLConnection;
73.
74. import org.apache.sling.api.resource.Resource;
75. import org.apache.sling.jcr.api.SlingRepository;
76. import org.osgi.service.component.annotations.Component;
77. import org.osgi.service.component.annotations.Reference;
78. import org.slf4j.Logger;
79. import org.slf4j.LoggerFactory;
80.
81. import com.day.cq.commons.jcr.JcrUtil;
82. import com.day.cq.polling.importer.ImportException;
83. import com.day.cq.polling.importer.Importer;
84. import com.google.gson.JsonElement;
85. import com.google.gson.JsonObject;
86. import com.google.gson.JsonParser;
87.
88. /**
89. * This class imports from IEX api and creates the data structure example below:
90. *
91. * /content/stocks/
92. * + <STOCK_SYMBOL> [sling:OrderedFolder]
93. *   + lastTrade [nt:unstructured]
94. *     - companyName = <value>
95. *     - sector = <value>
96. *     - lastTrade = <value>
97. *     - timeOfUpdate = <value>
```

```

98. *           - dayOfLastUpdate = <value>
99. *           - openPrice = <value>
100. *           - rangeHigh = <value>
101. *           - rangeLow = <value>
102. *           - volume = <value>
103. *           - upDownPrice = <value>
104. *           - week52High = <value>
105. *           - week52Low = <value>
106. *           - ytdChange = <value>
107. *
108. */
109.
110. @Component(service = Importer.class, property = "importer.scheme=stock")
111.
112. public class StockDataImporter implements Importer {
113.     private final Logger logger = LoggerFactory.getLogger(getClass());
114.
115.     //controls the property NAME values to be written to the JCR in lines 170-180:
116.     private static final String COMPANY = "companyName";
117.     private static final String SECTOR = "sector";
118.     private static final String LASTTRADE = "lastTrade";
119.     private static final String UPDATETIME = "timeOfUpdate";
120.     private static final String DAYOFUPDATE = "dayOfLastUpdate";
121.     private static final String OPENPRICE = "openPrice";
122.     private static final String RANGEHIGH = "rangeHigh";
123.     private static final String RANGELOW = "rangeLow";
124.     private static final String VOLUME = "volume";
125.     private static final String UPDOWN = "upDown";
126.     private static final String WEEK52HIGH = "week52High";
127.     private static final String WEEK52LOW = "week52Low";
128.     private static final String YTDCHANGE = "ytdPercentageChange";
129.
130.     @Reference
131.     private SlingRepository repo;
132.
133.     @Override
134.     public void importData(final String scheme, final String dataSource, final Resource
e resource)
135.             throws ImportException {
136.             try {
137.                 // dataSource will be interpreted as the stock symbol
138.
139.                 //BW - Updated using the IEX api - https://iextrading.com/developer/docs/#/
quote
140.                 String iexApiUrl = "https://api.iextrading.com/1.0/stock/" + dataSource +
"/quote";
141.                 URL sourceUrl = new URL(iexApiUrl);
142.                 HttpsURLConnection request = (HttpsURLConnection) sourceUrl.openConnection();
143.                 request.connect();
144.
145.                 // Convert IEX data return to a JSON object
146.                 JsonParser parser = new JsonParser();
147.                 JsonElement root = parser.parse(new InputStreamReader((InputStream) request
.getContent()));
148.                 //JsonObject will give access to quote data via keys
149.                 JsonObject allQuoteData = root.getAsJsonObject();
150.
151.                 logger.info("Last trade for stock symbol {} was {}", dataSource, allQuoteD
ata.get("latestPrice"));
152.

```

```

153.             writeToRepository(dataSource, allQuoteData, resource);
154.
155.         }
156.         catch (MalformedURLException e) {
157.             logger.error("MalformedURLException", e);
158.         }
159.         catch (IOException e) {
160.             logger.error("IOException", e);
161.         }
162.         catch (RepositoryException e) {
163.             logger.error("RepositoryException", e);
164.         }
165.
166.     }
167.
168.     /**
169.      * Creates the stock data structure
170.      *
171.      * + <STOCK_SYMBOL> [sling:OrderedFolder]
172.      *   + lastTrade [nt:unstructured]
173.      *     - companyName = <value>
174.      *     - sector = <value>
175.      *     - lastTrade = <value>
176.      *     - timeOfUpdate = <value>
177.      *     - dayOfLastUpdate = <value>
178.      *     - openPrice = <value>
179.      *     - rangeHigh = <value>
180.      *     - rangeLow = <value>
181.      *     - volume = <value>
182.      *     - upDownPrice = <value>
183.      *     - week52High = <value>
184.      *     - week52Low = <value>
185.      *     - ytdChange = <value>
186.      */
187.     private void writeToRepository(final String stockSymbol, final JSONObject quoteDat
a, final Resource resource) throws RepositoryException {
188.
189.         logger.info("Stock Symbol: " + stockSymbol);
190.         logger.info("JsonObject to Write: " + quoteData.toString());
191.
192.         Session session= repo.loginService("training",null);
193.
194.         //NOTE: /content/stocks has to exist in the JCR to successfully adapt resour
ce to Node here.
195.         Node parent = resource.adaptTo(Node.class);
196.
197.         if(parent == null) {
198.             logger.warn("The node " + resource.getPath() + " does not exist. Autocreat
ing.");
199.             parent = JcrUtil.createPath(resource.getPath(), "sling:OrderedFolder", ses
sion);
200.         }
201.
202.         if (parent != null) {
203.             Node stockPageNode = JcrUtil.createPath(parent.getPath() + "/" + stockSymb
ol, "sling:OrderedFolder", session);
204.             Node lastTradeNode = JcrUtil.createPath(stockPageNode.getPath() + "/lastTr
ade", "nt:unstructured", session);
205.             //use java.time api for Date & Time. Set time to EST for our application
206.             ZoneId timeZone = ZoneId.of("America/New_York");

```

```

207.             long latestUpdateTime = quoteData.get("latestUpdate").getAsLong();
208.             LocalDateTime timePerLatestUpdate = LocalDateTime.ofInstant(Instant.ofEpoch
   hMilli(latestUpdateTime),
   timeZone);
210.             ZonedDateTime timeWithZone = ZonedDateTime.of(timePerLatestUpdate, timeZone);
   e);
211.             DateTimeFormatter timeFormatter = DateTimeFormatter.ofPattern("hh:mm a zzz");
   );
212.             //will store timeOfUpdate as: Hour:Minute AM/PM, TimeZone e.g. 11:34
   AM, EDT
213.             String iexUpdateTimeOfDay = timeWithZone.format(timeFormatter);
214.             DateTimeFormatter dayFormatter = DateTimeFormatter.ofPattern("E MMMM d, yyyy");
   yy);
215.             String dayOfUpdate = timeWithZone.format(dayFormatter);
216.             lastTradeNode.setProperty(COMPANY, quoteData.get("companyName").getAsString());
   g());
217.             lastTradeNode.setProperty(SECTOR, quoteData.get("sector").getAsString());
218.             /*
   * latestPrice & latestUpdate from IEX both STOP at 4pm Eastern Time (New
   York)
219.             * so - effectively, updates stop at this point and do not resume until
   the next trading day
220.             */
221.             lastTradeNode.setProperty(LASTTRADE, quoteData.get("latestPrice").getAsDouble());
222.             lastTradeNode.setProperty(UPDATETIME, iexUpdateTimeOfDay);
223.             lastTradeNode.setProperty(DAYOFUPDATE, dayOfUpdate);
224.             lastTradeNode.setProperty(OPENPRICE, quoteData.get("open").getAsDouble());
225.
226.             lastTradeNode.setProperty(RANGEHIGH, quoteData.get("high").getAsDouble());
227.             lastTradeNode.setProperty(RANGELOW, quoteData.get("low").getAsDouble());
228.             lastTradeNode.setProperty(VOLUME, quoteData.get("latestVolume").getAsLong());
   );
229.             lastTradeNode.setProperty(UPDOWN, quoteData.get("change").getAsDouble());
230.             lastTradeNode.setProperty(WEEK52HIGH, quoteData.get("week52High").getAsDouble());
   ble());
231.             lastTradeNode.setProperty(WEEK52LOW, quoteData.get("week52Low").getAsDouble());
   e());
232.             lastTradeNode.setProperty(YTDCHANGE, quoteData.get("ytdChange").getAsDouble());
   e());
233.         } else {
234.             logger.error("##### JCR resource: " + resource.getPath() + " needs to be
   created for Importer to write data");
235.         }
236.         session.save();
237.         session.logout();
238.     }
239.
240.     @Override
241.     public void importData(String scheme, String dataSource, Resource target, String login,
   password) throws ImportException {
242.         importData(scheme, dataSource, target);
243.
244.     }
245. }
246.

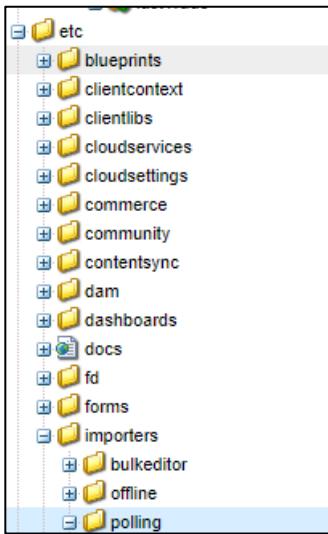
```

## Task 2: Deploy and test the polling importer

1. In Project Explorer, right-click **training.core** and choose **Run As > Maven install**. The build starts.
2. Verify the bundle installed successfully, as shown:

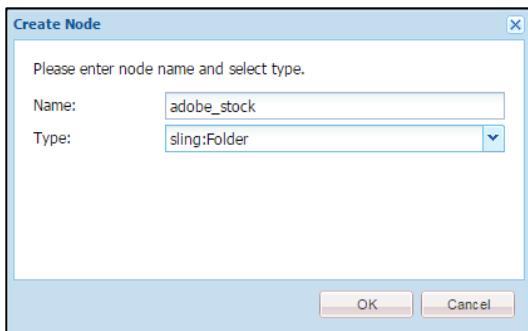
```
<terminated> C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Mar 20, 2018, 5:26:54 PM)
[INFO] Downloaded: https://repo.adobe.com/nexus/content/groups/public/org/codehaus/plexus/plexus-utils/1.0.4/plexus-utils-1.0.4.jar (160 KB at 310.0 KB/sec)
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar
[INFO] Downloaded: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar (9 KB at 27.5 KB/sec)
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/asm/asm-all/3.3.1/asm-all-3.3.1.jar
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.0/maven-archiver-2.0.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.0/maven-archiver-2.0.jar (12 KB at 161.2 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar (128 KB at 715.3 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar (19 KB at 257.7 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar (530 KB at 1975.9 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/asm/asm-all/3.3.1/asm-all-3.3.1.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/asm/asm-all/3.3.1/asm-all-3.3.1.jar (203 KB at 1454.4 KB/sec)
[INFO] Installing Bundle com.adobe.training.core(C:\Users\pprprush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar) to http://localhost:4502/crx/repository/cm
[INFO] Bundle installed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 10.106 s
[INFO] Finished at: 2018-03-20T17:27:05-07:00
[INFO] Final Memory: 23M/307M
[INFO] -----
```

3. Open CRXDE Lite <http://localhost:4502/crx/de/index.jsp> and navigate to **etc > importers > polling**, as shown:



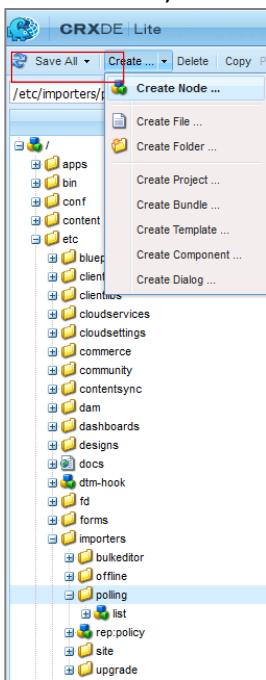
4. Right-click **polling**, and select **Create > Create node**. Provide the following values:

- Select **sling:folder** from the **Type** drop-down menu.
- Enter **adobe\_stock** for the **Name** field.



5. Click **OK**. The node **adobe\_stock** is created.

6. Click **Save All**, as shown:



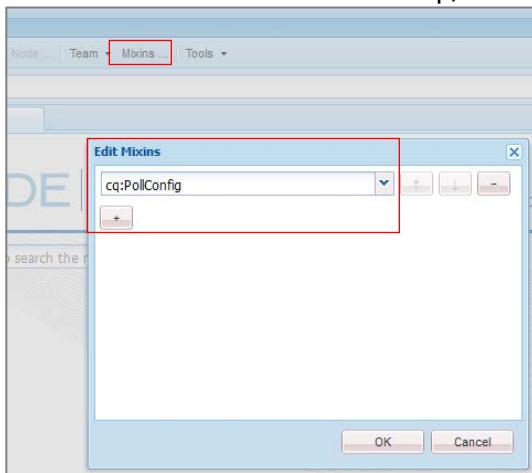
7. On the **adobe\_stock** folder node, add the following properties, as shown:

- Name: **interval**  
Type: **String**  
Value: **300**
- Name: **source**  
Type: **String**  
Value: **stock:ADBE**
- Name: **target**  
Type: **String**  
Value: **/content/stocks**

The screenshot shows the 'Properties' tab of the AEM dialog. The table lists the following properties:

Name	Type	Value	Protected	Mandatory	Multiple
1 interval	String	300	false	false	false
2 jcr:created	Date	2018-04-12T19:50:50.929...	true	false	false
3 jcr:createdBy	String	admin	true	false	false
4 jcr:mixinTypes	Name[]	cq:PollConfig	true	false	true
5 jcr:primaryType	Name	sling:Folder	true	true	false
6 source	String	stock:ADBE	false	true	false
7 target	String	/content/stocks	false	false	false

8. Click **Mixins** in the toolbar at the top, click **+**, and select **cq:PollConfig** from the drop-down menu, as shown:



This helps JCR identify a Mixins configuration.

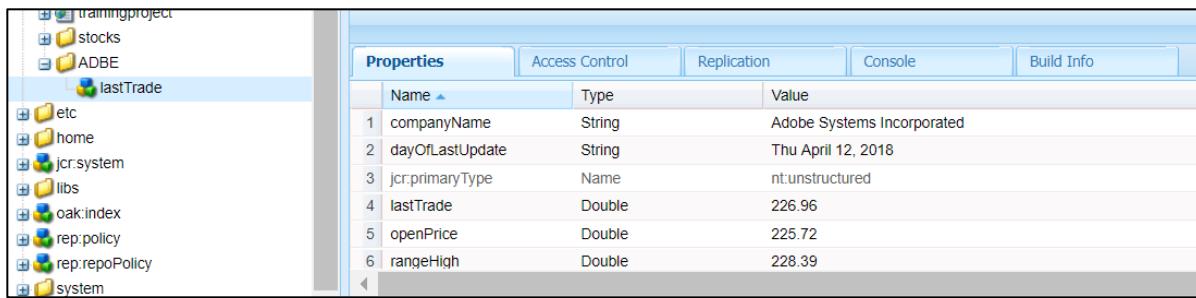
9. Click **OK**. The Mixins configuration is created.

10. Click **Save All** to save the properties created on the **adobe\_stock** node.

 **Note:** The configurations you created will provide the information to the custom polling importer you created, StockDataImporter.java class. As a result, in 300 seconds (5 minutes), you should see a new node created at the target path (/content) provided in the node.

11. Refresh **/content** in **CRXDE Lite**, and notice that there is a **ADBE** node created.

Notice that there is a **lastTrade** node under the **ADBE** node, and the properties on the **lastTrade** node is the data you imported. This data is shown in the screenshot below:



The screenshot shows the CRXDE Lite interface with the left sidebar displaying a tree view of the repository structure. The 'stocks' folder contains an 'ADBE' folder, which in turn contains a 'lastTrade' node. The right panel shows the 'Properties' tab selected, displaying a table of properties for the 'lastTrade' node. The table has columns for Name, Type, and Value.

Name	Type	Value
1 companyName	String	Adobe Systems Incorporated
2 dayOfLastUpdate	String	Thu April 12, 2018
3 jcr:primaryType	Name	nt:unstructured
4 lastTrade	Double	226.96
5 openPrice	Double	225.72
6 rangeHigh	Double	228.39

# Creating AEM Pages and Assets Programmatically

Following are the best practices for data migration are:

- Create pages dynamically
- Create assets dynamically
- Identify cost benefit

## Create Pages Dynamically

AEM has 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.

## Create 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.

## Identify 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 employees to enter the data into a format that is compatible with AEM.
- Automated migration (using APIs): Export data as XML from the legacy system, write a script to convert the data into a JCR-friendly XML, and import that package into JCR.
- Hybrid migration (a combination of manual and automated): Import the bulk of the data through XML, and hire employees to update the data manually. For example, you can follow this method to add metadata to modify the data.

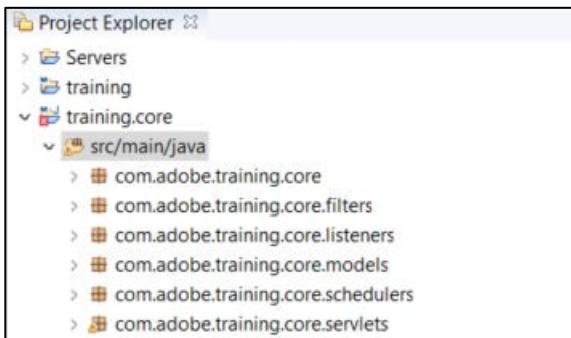
## Exercise 2: Create AEM pages programmatically

In this exercise, you will create a page programmatically, with a servlet and a resource from JCR. This exercise includes two tasks:

1. Create a page creator
2. Deploy and test the page creator

### Task 1: Create a page creator

1. Double-click the **Eclipse** shortcut on the desktop. The Eclipse Launcher opens.
2. Specify the Workspace as **C:\Workspace** and click **Launch**. The Eclipse Workspace opens.
3. In Project Explorer, navigate to **training.core > src/main/java**, as shown:

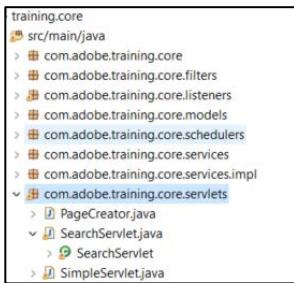


4. Copy the file `PageCreator.java` from `/Exercise_Files/12_Deep_Dive_into_AEM_APIs/`.



**Note:** The code is provided as part of Exercise\_Files under `/Exercise_Files/12_Deep_Dive_into_AEM_APIs/`. Copy and paste the file from the exercise files referenced to `PageCreator.java` to Eclipse. Please do not copy the code from this student guide. The code in the student guide is for illustrative purposes only.

5. In Eclipse, right-click **com.adobe.training.core.servlets**, and paste the file. The **PageCreator.java** class is created, as shown:



6. Double-click **PageCreator.java**. The file opens in editor, as shown:

```
1. package com.adobe.training.core.servlets;
2.
3. import java.io.IOException;
4. import java.util.HashMap;
5.
6. import javax.jcr.Session;
7. import javax.servlet.Servlet;
8. import javax.servlet.ServletException;
9.
10. import com.google.gson.Gson;
11. import org.osgi.service.component.annotations.Component;
12. import org.osgi.service.component.annotations.Reference;
13.
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.
19. import org.slf4j.Logger;
20. import org.slf4j.LoggerFactory;
21.
22. import com.day.cq.replication.ReplicationActionType;
23. import com.day.cq.replication.Replicator;
24. import com.day.cq.tagging.Tag;
25. import com.day.cq.tagging.TagManager;
26. import com.day.cq.wcm.api.Page;
27. import com.day.cq.wcm.api.PageManager;
28.
29. /**
30. * This Servlet ingests a comma delimited string in the form of:
31. *
32. * New Page Path, Page Title, Page Tag, Auto Publish, Template
33. *
34. * And outputs the AEM page created.
35. *
36. * To test, you must create a content node with a resourceType=trainingproject/tools/pagecreator
37. *
38. * /content/pagecreator {sling:resourceType=trainingproject/tools/pagecreator}
39. *
40. * Example cURL Command:
41. * $ curl -u admin:admin -X POST http://localhost:4512/content/pagecreator.json -
F importer="/content/trainingproject/en/community,Our Community,/apps/trainingproject/templates/page-home,/etc/tags/we-retail/activity/biking,TRUE"
```

```

42. *
43. */
44.
45.
46. @Component(service = Servlet.class,
47.             property = {"sling.servlet.resourceTypes=trainingproject/tools/pagecreator",
48.                         "sling.servlet.methods=POST"
49.                     })
50.
51. public class PageCreator extends SlingAllMethodsServlet{
52.     private final Logger logger = LoggerFactory.getLogger(this.getClass());
53.
54.     private static final long serialVersionUID = 1L;
55.
56.     @Reference
57.     private Replicator replicator;
58.
59.     private Resource resource;
60.
61.     @Override
62.     public void doPost(SlingHttpServletRequest request, SlingHttpServletResponse response) throws
63.             ServletException, IOException{
64.         response.setHeader("Content-Type", "application/json");
65.         HashMap<String, String> resultObject = new HashMap<String, String>();
66.         Gson jsonResponse = new Gson();
67.
68.         resource = request.getResource();
69.
70.         String param = request.getParameter("importer");
71.         String[] nextPage = param.split(",");
72.         try {
73.             resultObject = createTrainingPage(nextPage[0], nextPage[1], nextPage[2], nextPage[3],
74.                     nextPage[4]);
75.         } catch (Exception e) {
76.             resultObject.put("Status", "Could not properly parse");
77.             logger.error("Failure to create page: " + e);
78.         }
79.         if(resultObject != null){
80.             //Write the result to the page
81.             response.getWriter().print(jsonResponse.toJson(resultObject));
82.             response.getWriter().close();
83.         }
84.
85.         /** Helper method to create the page based on available input
86.          *
87.          * @param path JCR location of the page to be created
88.          * @param title Page Title
89.          * @param template AEM Template this page should be created from. The template must exist
90.          * in the JCR already.
91.          * @param tag Tag must already be created in AEM. The tag will be in the form of a path.
92.          * Ex /etc/tags/marketing/interest
93.          * @param publish boolean to publish the page
94.          * @return HashMap
95.          */
96.
97.         private HashMap<String, String> createTrainingPage(String path, String title, String tem-
late, String tag, String publish) throws Exception {
98.             HashMap<String, String> pageInfo = new HashMap<String, String>();
99.             if (path != null) {

```

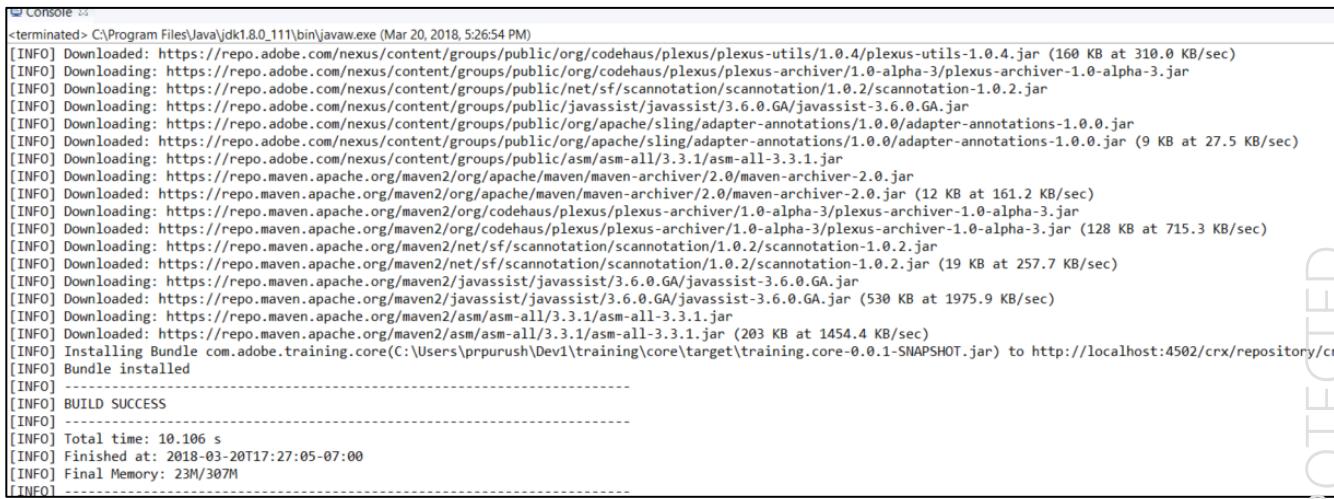
```

98.         //Parse the path to get the pageNodeName and parentPath
99.         int lastSlash = path.lastIndexOf("/");
100.        String pageNodeName = path.substring(lastSlash + 1);
101.        String parentPath = path.substring(0, lastSlash);
102.
103.        //Set a default template if none is given
104.        if (template == null || template.isEmpty()) {
105.            template = "/apps/trainingproject/templates/page-content";
106.        }
107.
108.        //Create page
109.        PageManager pageManager = resource.getResourceResolver().adaptTo(PageManager.class);
110.
111.        if (pageManager != null && !parentPath.isEmpty() && !pageNodeName.isEmpty())
112.        {
113.            Page p = pageManager.create(parentPath,
114.                pageNodeName,
115.                template,
116.                title);
117.
118.            //Add a tag to the page
119.            if (tag != null && !tag.isEmpty())
120.            {
121.                //TagManager can be retrieved via adaptTo
122.                TagManager tm = resource.getResourceResolver().adaptTo(TagManager.class);
123.
124.                if (tm != null)
125.                {
126.                    tm.setTags(p.getContentResource(),
127.                        new Tag[]{tm.resolve(tag)},
128.                        true);
129.                }
130.
131.            }
132.            //Replicator is exposed as a service
133.            replicator.replicate(resource.getResourceResolver().adaptTo(Session.class),
134.                ReplicationActionType.ACTIVATE,
135.                p.getPath());
136.
137.
138.            pageInfo.put("Status", "Successful");
139.            pageInfo.put("Location", p.getPath());
140.            pageInfo.put("Title", p.getTitle());
141.            pageInfo.put("Template Used", p.getTemplate().getPath());
142.            pageInfo.put("Tagged with", p.getTags()[0].getTitle());
143.            pageInfo.put("Was Published", String.valueOf(publishPage));
144.        }
145.    }
146.
147.    if(pageInfo.isEmpty())
148.    {
149.        pageInfo.put("Status", "Could not create a page");
150.    }
151.    return pageInfo;
152.}

```

## Task 2: Deploy and test the page creator

1. In Project Explorer, right-click **training.core** and choose **Run As > Maven install**. The build starts.
2. Verify the bundle installed successfully, as shown:



```
Console 14
<terminated> C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Mar 20, 2018, 5:26:54 PM)
[INFO] Downloaded: https://repo.adobe.com/nexus/content/groups/public/org/codehaus/plexus/plexus-utils/1.0.4/plexus-utils-1.0.4.jar (160 KB at 310.0 KB/sec)
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar
[INFO] Downloaded: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar (9 KB at 27.5 KB/sec)
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/asm/asm-all/3.3.1/asm-all-3.3.1.jar
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.0/maven-archiver-2.0.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.0/maven-archiver-2.0.jar (12 KB at 161.2 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar (128 KB at 715.3 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar (19 KB at 257.7 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar (530 KB at 1975.9 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/asm/asm-all/3.3.1/asm-all-3.3.1.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/asm/asm-all/3.3.1/asm-all-3.3.1.jar (203 KB at 1454.4 KB/sec)
[INFO] Installing Bundle com.adobe.training.core(C:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar) to http://localhost:4502/crx/repository/crx/bundles/com.adobe.training.core
[INFO] Bundle installed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 10.106 s
[INFO] Finished at: 2018-03-20T17:27:05-07:00
[INFO] Final Memory: 23M/307M
[INFO] -----
```

3. Open CRXDE Lite <http://localhost:4502/crx/de/index.jsp> and navigate to **/content**.
4. Right-click **content**, and select **Create > Create node**. Provide the following values:
  - a. Enter **pagecreator** for the **Name** field.
  - b. Select **nt:unstructured** from the **Type** drop-down menu.
5. Click **OK**. The node **pagecreator** is created.
6. Click **Save All**.

7. On the **pagecreator** folder node, add the following property, as shown:

- Name: **sling:resourceType**
- Type: **String**
- Value: **trainingproject/tools/pagecreator**

Properties			
Name	Type	Value	
1 jcr:primaryType	Name	nt:unstructured	
2 sling:resourceType	String	trainingproject/tools/pagecreator	

8. Click **Save All** to save the properties created on the **pagecreator** node.



**Note:** You have content that you can post and a servlet that will be triggered. You can now test with a cURL command.

9. Copy the first command from /Exercise\_Files/12\_Deep\_Dive\_into\_AEM\_APIs/pagecreator-curl-command.txt.

10. Open your command terminal, and run the following command:

```
C:\Users\prpurush>curl -u admin:admin -X POST http://localhost:4502/content/pagecreator.json -F imp  
orter="/content/trainingproject/en/community,Our Community,/apps/trainingproject/templates/page-hom  
e,/etc/tags/we-retail/activity/biking,TRUE"  
{"Status": "Successful", "Title": "Our Community", "Template Used": "/apps/trainingproject/templates/pag  
e-home", "Tagged with": "Biking", "Was Published": "true", "Location": "/content/trainingproject/en/commu  
nity"}
```



**Note:** If you are on a Mac, you can format the returned json by adding | json\_pp.

11. Verify the newly created page in **Sites > TrainingProject Site > English**, as shown:

The screenshot shows the AEM Sites interface. On the left, there's a navigation tree with categories like Screens, Campaigns, Community Sites, We.Retail, TrainingProject Site, and stocks. Under TrainingProject Site, there's a folder named 'en' which contains a page titled 'My Page!' and another page titled 'Our Community'. The 'Our Community' page is selected, indicated by a blue checkmark icon. The right side of the screen displays the properties of the selected page. The properties table includes columns for Title, Name, Template, Modified, and Modified By. The Title is 'Our Community', Name is 'community', Template is 'TrainingProject Site Home Page', Modified is '2 minutes ago', and Modified By is 'Administrator'. There's also a preview area showing a grid-based layout.



**Note:** If you want to use the browser rather than cURL, you can install the http-pagecreator.zip content package and request for <http://localhost:4502/content/pagecreator.html>.

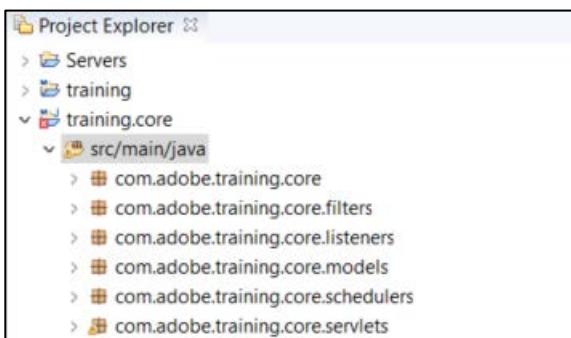
## Exercise 3: Programmatically create an AEM page

In this exercise, you will create a website programmatically, with a servlet and a csv file. This exercise includes two tasks:

1. Create a CSV page creator
2. Deploy and test the CSV page creator

### Task 1: Create a CSV page creator

1. Double-click the **Eclipse** shortcut on the desktop. The Eclipse Launcher opens.
2. Specify the Workspace as **C:\Workspace** and click **Launch**. The Eclipse Workspace opens.
3. In Project Explorer, navigate to **training.core > src/main/java**, as shown:

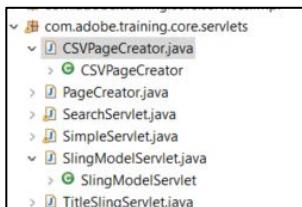


4. Copy the file **CSVPageCreator.java** from **/Exercise\_Files/12\_Deep\_Dive\_into\_AEM\_APIs/**.



**Note:** The code is provided as part of the Exercise\_Files under /Exercise\_Files/12\_Deep\_Dive\_into\_AEM\_APIs/. Copy and paste the file from the exercise files referenced to CSVPageCreator.java to Eclipse. Please do not copy the code from this student guide. The code in the student guide is for illustrative purposes only.

5. In Eclipse, right-click **com.adobe.training.core.servlets** and paste the file. The **CSVPageCreator.java** class is created, as shown:



6. Double-click **CSVPageCreator.java**. The file opens in editor.

```
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. import java.util.HashMap;
9.
10. import javax.jcr.Session;
11. import javax.servlet.Servlet;
12. import javax.servlet.ServletException;
13.
14. import com.google.gson.Gson;
15.
16. import org.osgi.service.component.annotations.Reference;
17. import org.osgi.service.component.annotations.Component;
18.
19. import org.apache.sling.api.SlingHttpServletRequest;
20. import org.apache.sling.api.SlingHttpServletResponse;
21. import org.apache.sling.api.resource.Resource;
22. import org.apache.sling.api.servlets.SlingAllMethodsServlet;
23.
24. import org.slf4j.Logger;
25. import org.slf4j.LoggerFactory;
26.
27. import com.day.cq.replication.ReplicationActionType;
28. import com.day.cq.replication.Replicator;
29. import com.day.cq.tagging.Tag;
30. import com.day.cq.tagging.TagManager;
31. import com.day.cq.wcm.api.Page;
32. import com.day.cq.wcm.api.PageManager;
33.
34. /**
35. * This Servlet ingests a .csv file with the following columns:
36. *
37. * New Page Path, Page Title, Page Tag, Auto Publish, Template
38. *
39. * And outputs AEM pages created.
40. *
41. * To test, you must create a content node with a resourceType=trainingproject/tools/pagecreator
42. *
43. * /content/pagecreator {sling:resourceType=trainingproject/tools/pagecreator}
44. *
45. * Example cURL Command:
```

```

46. * $ curl -u admin:admin -X POST http://localhost:4512/content/pagecreator.csv.json -
  F importer=@PageCreator.csv
47. *
48. */
49.
50. @Component( service = Servlet.class,
51.               property = {
52.                 "sling.servlet.resourceTypes=trainingproject/tools/pagecreator",
53.                 "sling.servlet.selectors=csv",
54.                 "sling.servlet.methods=POST"
55.               }
56.             )
57. public class CSVPageCreator extends SlingAllMethodsServlet {
58.
59.     private static final long serialVersionUID = 1L;
60.     private final Logger logger = LoggerFactory.getLogger(getClass());
61.
62.     @Reference
63.     private Replicator replicator;
64.
65.     private Resource resource;
66.
67.     public void doPost(SlingHttpServletRequest request, SlingHttpServletResponse response) th
rows ServletException, IOException {
68.         response.setHeader("Content-Type", "application/json");
69.         HashMap<String, HashMap<String, String>> resultObject = new HashMap<String, HashMap<S
tring, String>>();
70.         Gson jsonResponse = new Gson();
71.         resource = request.getResource();
72.
73.         String param = request.getParameter("importer");
74.         byte[] input = param.getBytes();
75.         InputStream stream = new ByteArrayInputStream(input);
76.         try {
77.             resultObject = readCSV(stream);
78.         } catch (Exception e) {
79.             logger.error("Failure to Read CSV: " + e);
80.         }
81.
82.         if (resultObject != null) {
83.             //Write the result to the page
84.             response.getWriter().print(jsonResponse.toJson(resultObject));
85.             response.getWriter().close();
86.         }
87.     }
88.
89. /**
90. * Reads the CSV file. The CSV file MUST be in the form of:
91. * <p>
92. * JCR path, Page Title, Page Template, AEM Tag, Publish boolean
93. *
94. * @param stream Stream from the CSV
95. * @return JSON object that contains the results of the page creation process
96. */
97. private HashMap<String, HashMap<String, String>> readCSV(InputStream stream) throws IOExc
eption, Exception {
98.     HashMap<String, HashMap<String, String>> out = new HashMap<String, HashMap<String, St
ring>>();
99.     BufferedReader br = new BufferedReader(new InputStreamReader(stream));
100.
101.    String line;

```

```

102.         String[] newPage;
103.         HashMap<String, String> createdPageObject = null;
104.         //Read each line of the CSV
105.         while ((line = br.readLine()) != null) {
106.             newPage = line.split(",");
107.             String aemTag = null;
108.             String publishFlag = null;
109.             String aemTemplatePath = null;
110.
111.             //If the line has a template, tag, publish flag, set those variables
112.             if (newPage.length == 5) {
113.                 aemTemplatePath = newPage[2];
114.                 aemTag = newPage[3];
115.                 publishFlag = newPage[4];
116.             } else if (newPage.length == 4) {
117.                 aemTemplatePath = newPage[2];
118.                 aemTag = newPage[3];
119.             } else if (newPage.length == 3) {
120.                 publishFlag = newPage[2];
121.             }
122.
123.             //As long as there is a path and title, the page can be created
124.             if ((newPage.length > 1)
125.                 && !newPage[0].isEmpty()
126.                 && !newPage[1].isEmpty()) {
127.                 String path = newPage[0];
128.                 String title = newPage[1];
129.                 try {
130.                     createdPageObject = createTrainingPage(path, title, aemTemplatePath,
131.                     aemTag, publishFlag);
132.                 } catch (Exception e) {
133.                     logger.error(path + " not created successfully: " + e);
134.                 }
135.                 //add the status of the row into the json array
136.                 out.put(title, createdPageObject);
137.                 createdPageObject = null;
138.             } else {
139.                 createdPageObject = new HashMap<String, String>();
140.                 createdPageObject.put("Status", "Could not properly parse");
141.                 out.put(line, createdPageObject);
142.                 createdPageObject = null;
143.             }
144.         }
145.         br.close();
146.         return out;
147.     }
148.
149. /**
150. * Helper method to create the page based on available input
151. *
152. * @param path      JCR location of the page to be created
153. * @param title    Page Title
154. * @param template AEM Template this page should be created from. The template must
155. *                  exist in the JCR already.
156. * @param tag      Tag must already be created in AEM. The tag will be in the form
157. *                  of a path. Ex /etc/tags/marketing/interest
158. * @param publish  boolean to publish the page
159. */
160. private HashMap<String, String> createTrainingPage(String path, String title, String
template, String tag, String publish) throws Exception {

```

```

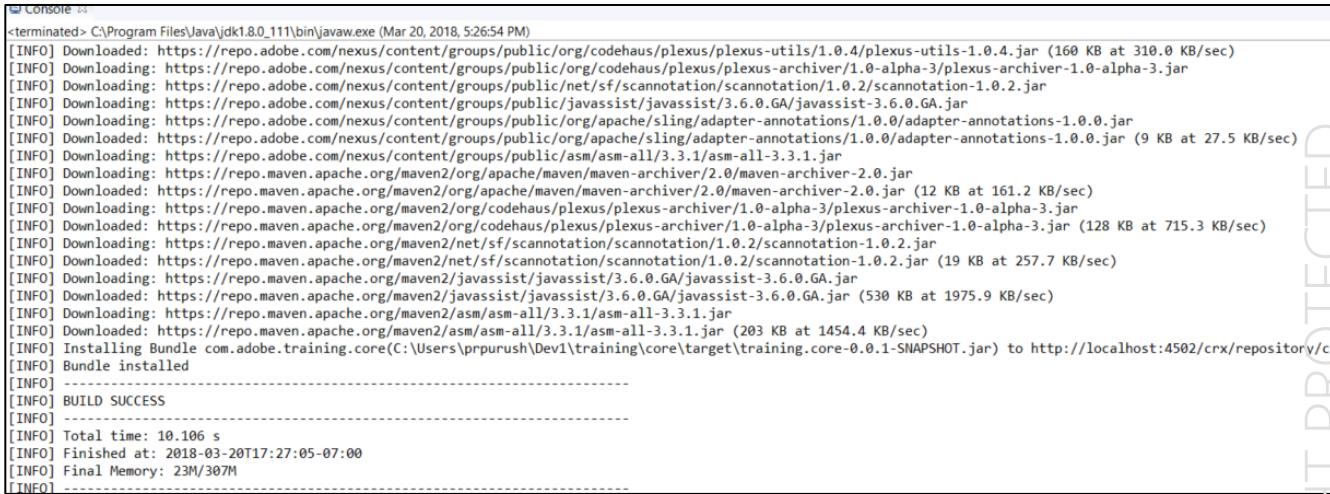
159.         HashMap<String, String> pageInfo = new HashMap<String, String>();
160.
161.        if (path != null) {
162.            //Parse the path to get the pageNodeName and parentPath
163.            int lastSlash = path.lastIndexOf("/");
164.            String pageNodeName = path.substring(lastSlash + 1);
165.            String parentPath = path.substring(0, lastSlash);
166.
167.            //Set a default template if none is given
168.            if (template == null || template.isEmpty()) {
169.                template = "/apps/trainingproject/templates/page-content";
170.            }
171.
172.            //Create page
173.            PageManager pageManager = resource.getResourceResolver().adaptTo(PageManager.class);
174.
175.            if (pageManager != null && !parentPath.isEmpty() && !pageNodeName.isEmpty())
176.            {
177.                Page p = pageManager.create(parentPath,
178.                    pageNodeName,
179.                    template,
180.                    title);
181.
182.                //Add a tag to the page
183.                if (tag != null && !tag.isEmpty()) {
184.                    //TagManager can be retrieved via adaptTo
185.                    TagManager tm = resource.getResourceResolver().adaptTo(TagManager.class);
186.                    if (tm != null) {
187.                        tm.setTags(p.getContentResource(),
188.                            new Tag[]{tm.resolve(tag)},
189.                            true);
190.                    }
191.
192.
193.                //Publish page if requested
194.                boolean publishPage = Boolean.parseBoolean(publish);
195.                if (publishPage) {
196.                    //Replicator is exposed as a service
197.                    replicator.replicate(resource.getResourceResolver().adaptTo(Session.class),
198.                        ReplicationActionType.ACTIVATE,
199.                        p.getPath());
200.                }
201.
202.                pageInfo.put("Status", "Successful");
203.                pageInfo.put("Location", p.getPath());
204.                pageInfo.put("Title", p.getTitle());
205.                pageInfo.put("Template Used", p.getTemplate().getPath());
206.                pageInfo.put("Tagged with", p.getTags()[0].getTitle());
207.                pageInfo.put("Was Published", String.valueOf(publishPage));
208.            }
209.        }
210.
211.        if(pageInfo.isEmpty()) {
212.            pageInfo.put("Status", "Could not create a page");
213.        }
214.        return pageInfo;

```

```
| 215.     } }
```

## Task 2: Deploy and test the CSV page creator

1. In Project Explorer, right-click training.core and select Run As > Maven install. The build starts.
2. Verify the bundle installed successfully, as shown:



```
Console >
<terminated> C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Mar 20, 2018, 5:26:54 PM)
[INFO] Downloaded: https://repo.adobe.com/nexus/content/groups/public/org/codehaus/plexus/plexus-utils/1.0.4/plexus-utils-1.0.4.jar (160 KB at 310.0 KB/sec)
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar
[INFO] Downloaded: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar (9 KB at 27.5 KB/sec)
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/asm/as-all/3.3.1/asm-all-3.3.1.jar
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/archiver/2.0/maven-archiver-2.0.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/archiver/2.0/maven-archiver-2.0.jar (12 KB at 161.2 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar (128 KB at 715.3 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar (19 KB at 257.7 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar (530 KB at 1975.9 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/asm/as-all/3.3.1/asm-all-3.3.1.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/asm/as-all/3.3.1/asm-all-3.3.1.jar (203 KB at 1454.4 KB/sec)
[INFO] Installing Bundle com.adobe.training.core(C:/Users/prpurush/Dev1/training/core/target/training.core-0.0.1-SNAPSHOT.jar) to http://localhost:4502/crx/repository/cr
[INFO] Bundle installed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 10.106 s
[INFO] Finished at: 2018-03-20T17:27:05-07:00
[INFO] Final Memory: 23M/307M
[INFO] -----
```



**Note:** Now that your new servlet accepts and processes CSV files, you can test with the cURL command.

3. Copy the command from /Exercise\_Files/12\_Deep\_Dive\_into\_AEM\_APIs/pagecreator-curl-command.txt.
4. Open your command terminal, and type the location of CSV files, as follows:

➤ CD C:\Exercise\_Files\12\_Deep\_Dive\_into\_AEM\_APIs

5. Try the servlet with bad data first. Execute the following statement, as shown:

```
curl -u admin:admin -X POST  
http://localhost:4502/content/pagecreator.csv.json -F  
importer=@PageCreator-badData.csv
```

```
C:\Dev\E&C_6.4\Exercise_Files\12_Deep_Dive_into_AEM_APIs>curl -u admin:admin -X POST http://localhost:4502/content/pagecreator.csv.json -F importer=@PageCreator-badData.csv  
{ "Women":{ "Status": "Successful", "Title": "Women", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Women", "Was Published": "true", "Location": "/content/trainingproject/en/products/women"}, "Khaki Shorts":{ "Status": "Successful", "Title": "Khaki Shorts", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Shorts", "Was Published": "false", "Location": "/content/trainingproject/en/products/women/shorts/khakis"}, "Products":{ "Status": "Successful", "Title": "Products", "Template Used": "/apps/trainingproject/templates/page-home", "Tagged with": "Apparel", "Was Published": "true", "Location": "/content/trainingproject/en/products"}, "T-shirts,,/etc/tags/we-retail/apparel/shirt":{ "Status": "Could not properly parse"}, "Shirts":{ "Status": "Successful", "Title": "Shirts", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Shirt", "Was Published": "true", "Location": "/content/trainingproject/en/products/women/shirts"}, "Polo Shirts,,/etc/tags/we-retail/apparel/shirt":{ "Status": "Could not properly parse"}, "Jean Shorts":{ "Status": "Successful", "Title": "Jean Shorts", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Shorts", "Was Published": "false", "Location": "/content/trainingproject/en/products/women/shorts/jeans"}, "Shorts":{ "Status": "Successful", "Title": "Shorts", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Shorts", "Was Published": "true", "Location": "/content/trainingproject/en/products/women/shorts"}}
```



**Note:** If you are on a Mac, you can format the returned json by adding | json\_pp.

6. Verify that all the pages are not created in AEM: **Sites > TrainingProject Site** > click the thumbnail on English page, as shown:

The screenshot shows the AEM authoring interface with the 'TrainingProject Site' selected. On the left, there's a tree view of sites and screens. In the center, there's a preview of an 'English' page titled 'My Page!' with some placeholder text. On the right, detailed properties for this page are listed:

Property	Value
Title	English
Name	en
Template	Content Page
Modified	a day ago
Modified By	Administrator
Page Title	New Project
Language	English
Published	7 days ago
Published By	Administrator

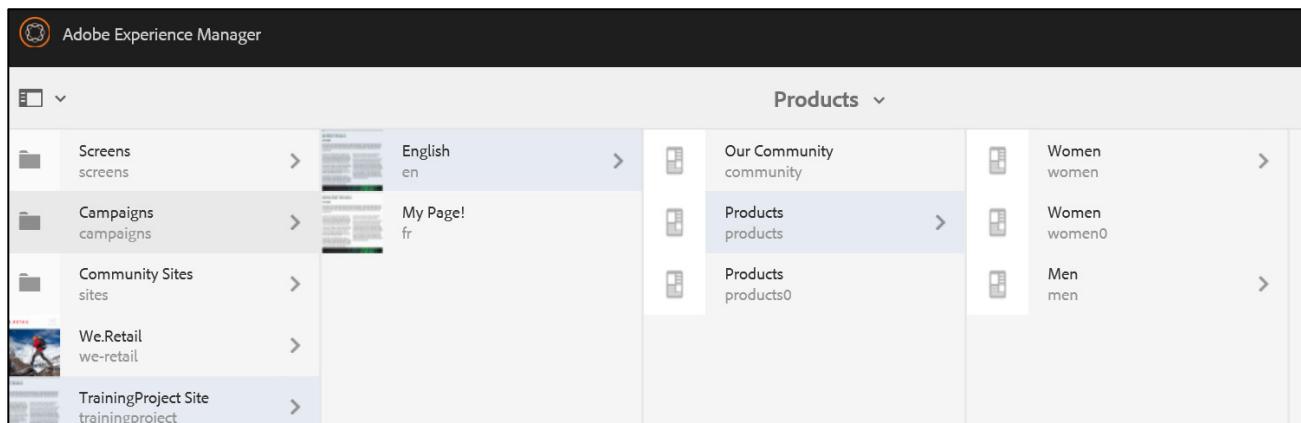
At the bottom right, it says 'Publication Pending #9 in the queue.'

7. Try the servlet with well formatted data. Execute the following statement, as shown:

```
curl -u admin:admin -X POST  
http://localhost:4502/content/pagecreator.csv.json -F  
importer=@PageCreator.csv
```

```
C:\Dev\E&C_6.4\Exercise_Files\12_Deep_Dive_into_AEM_APIs>curl -u admin:admin -X POST http://localhost:4502/content/pagecreator.csv.json -F importer=@PageCreator.csv  
{"women": [{"Status": "Successful", "Title": "Women", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Women", "Was Published": "true", "Location": "/content/trainingproject/en/products/women0"}, {"Khaki Shorts": {"Status": "Successful", "Title": "Khaki Shorts", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Shorts", "Was Published": "false", "Location": "/content/trainingproject/en/products/men/shorts/kakis"}, "Products": {"Status": "Successful", "Title": "Products", "Template Used": "/apps/trainingproject/templates/page-home", "Tagged with": "Apparel", "Was Published": "true", "Location": "/content/trainingproject/en/products0"}, {"Stretchy Pants": {"Status": "Successful", "Title": "Stretchy Pants", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Pants", "Was Published": "false", "Location": "/content/trainingproject/en/products/men/pants/stretchypants"}, "Pants": {"Status": "Successful", "Title": "Pants", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Pants", "Was Published": "true", "Location": "/content/trainingproject/en/products/men/pants"}, {"Jeans": {"Status": "Successful", "Title": "Jeans", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Pants", "Was Published": "false", "Location": "/content/trainingproject/en/products/men/pants/jeans"}, "Insulated Coats": {"Status": "Successful", "Title": "Insulated Coats", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Coat", "Was Published": "false", "Location": "/content/trainingproject/en/products/men/coats/insulated"}, "Coats": {"Status": "Successful", "Title": "Coats", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Coat", "Was Published": "true", "Location": "/content/trainingproject/en/products/men/coats"}, {"Shirts": {"Status": "Successful", "Title": "Shirts", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Shirt", "Was Published": "true", "Location": "/content/trainingproject/en/products/men/shirts"}, {"Jean Shorts": {"Status": "Successful", "Title": "Jean Shorts", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Shorts", "Was Published": "false", "Location": "/content/trainingproject/en/products/men/shorts/jeans"}, {"Rain Coats": {"Status": "Successful", "Title": "Rain Coats", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Coat", "Was Published": "false", "Location": "/content/trainingproject/en/products/men/coats/raincoat"}, {"T-shirts": {"Status": "Successful", "Title": "T-shirts", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Shirt", "Was Published": "false", "Location": "/content/trainingproject/en/products/men/shirts/tshirt"}, {"Men": {"Status": "Successful", "Title": "Men", "Template Used": "/apps/trainingproject/templates/page-content", "Tagged with": "Men", "Was Published": "true"}]}]}
```

8. Verify the newly created pages in AEM: **Sites > TrainingProject Site > English > Products**, as shown:



**Note:** If you want to use the browser rather than cURL, you can install the `http-pagecreator.zip` content package and request for <http://localhost:4502/content/pagecreator.html>.

## AEM Projects

The [AEM Projects](#) feature helps manage and group all workflows and tasks associated with creating content in AEM Sites and Assets.

### Project Template

AEM Projects comes with several out-of-the box ([OOTB](#)) [project templates](#). When creating a new project, authors can choose from these available templates. For large AEM implementations with unique business requirements, developers can create custom project templates. With these custom templates, developers can configure the project dashboard, hook into custom workflows, and create additional business roles for a project.

You should create project templates under source control, and they should be beneath your application folder under `/apps`. Ideally, they should be placed in a subfolder with the naming convention of `*/projects/templates/<my-template>`. This naming convention ensures that new custom templates will automatically become available to authors when creating a project.

The configuration of available project templates is set at the `/content/projects/jcr:content` node by the `cq:allowedTemplates` property. By default, this is a regular expression: `/(apps|libs)/*/*projects/templates/*`.

The root node of a project template will have `jcr:primaryType` of `cq:Template`. Beneath the root node, there are three nodes: gadgets, roles, and workflows. These nodes are all `nt:unstructured`.

- Gadgets

The child nodes of the gadgets node control which project tiles populate the project's dashboard when you create a new project. [The project tiles](#) are simple cards that populate the workplace of a project.

- Roles

There are three [default roles](#) for every project: Observers, Editors, and Owners. By adding child nodes beneath the roles node, you can add additional business specific project roles for the template. You can then tie these roles to specific workflows associated with the project.

- Workflows

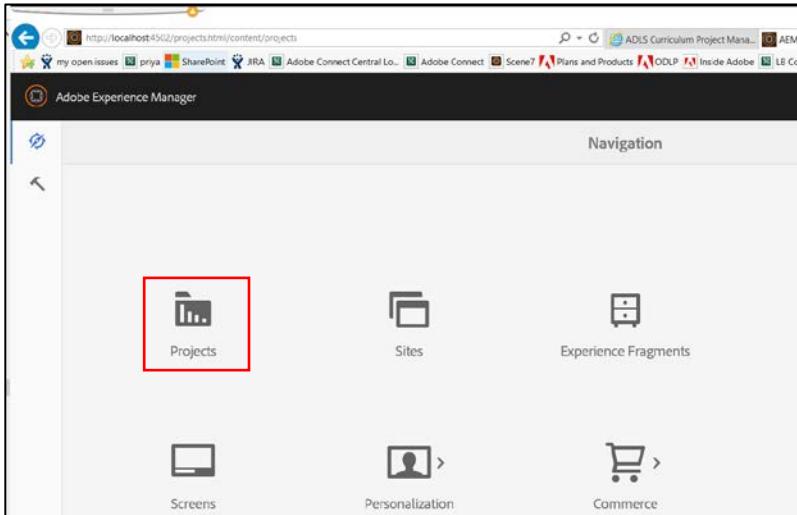
Workflows enable you to automate AEM activities. Workflows consist of a series of steps that are executed in a specific order. Each step performs a distinct activity, such as activating a page or sending an email message. One of the reasons for creating a custom project template is that it gives you the ability to configure the available workflows for the project. These can be OOTB workflows or custom workflows. Beneath the **workflows** node there needs to be a **models** node (`nt:unstructured`) and a child node to specify the available workflow models.

The root node of the project template will be of type `cq:Template`. On this node, you can configure the `jcr:title` and `jcr:description` properties that will be displayed in the Create Project wizard. There is also a property called `wizard` that points to a form that will populate the properties of the project.

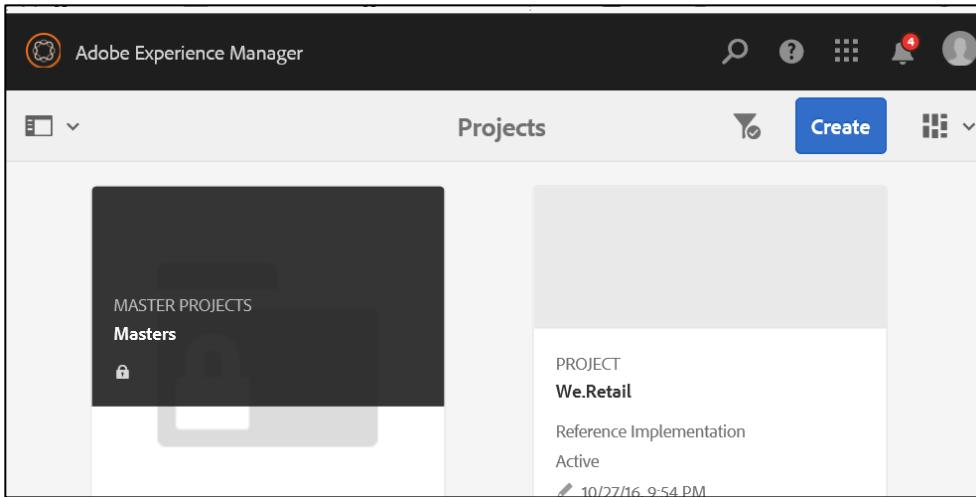
## Exercise 4: Create a simple project

In this exercise, you will create a simple project.

1. In AEM, go to the **Navigation** screen and click **Projects**, as shown:

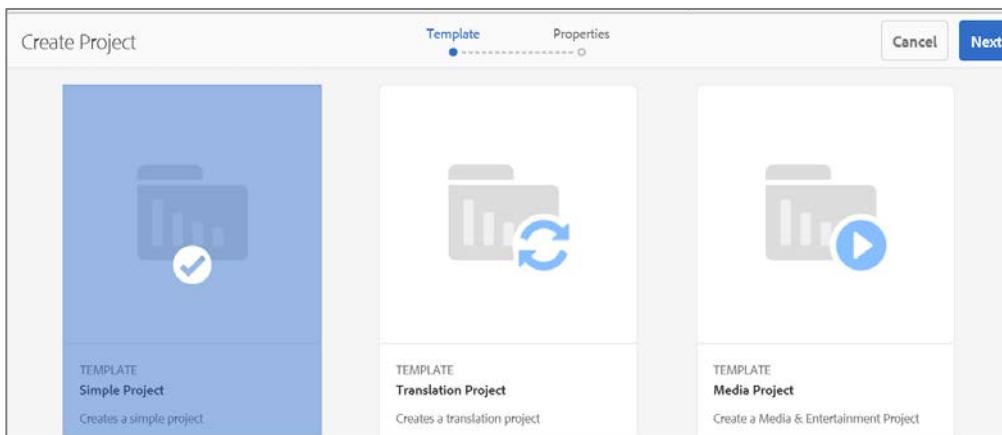


The **Projects** console opens, as shown:



+

2. In the **Projects** console, click **Create > Project**. The **Create Project** page opens.
3. Select **Simple Project** from the template, as shown:



4. Click **Next**. The **Properties** window opens.
5. Enter the following values for the simple project, as shown:
  - a. Enter **My Simple Project** in the **Title** field.
  - b. Enter **Carlene Avery** in the **User** field, ensure the dropdown beside it is set to **Editors**, and click **Add**.
  - c. Enter **Iris Mccoy** in the **User** field, ensure the dropdown beside it is set to **Observers**, and click **Add**.

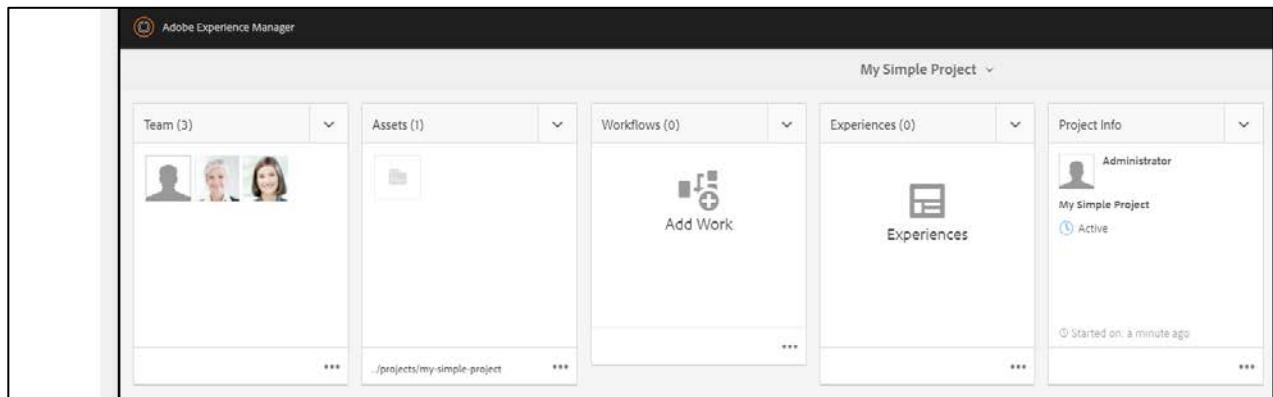
The screenshot shows the 'Properties' dialog box for a project titled 'My Simple Project'. The dialog has tabs for 'Template' and 'Properties', with 'Properties' selected. It includes sections for 'Basic' and 'Advanced' settings.

- Title**: My Simple Project
- Description**: (empty)
- Start Date**: (empty)
- Due Date**: (empty)
- User**:
  - Administrator: Carlene Avery (selected as Editors)
  - Owners: (empty)
  - Add: (button)
- Members**:
  - Administrators: Carlene Avery (selected as Editors)
  - Observers: Iris Mccoy (selected as Observers)

A red box highlights the list of users under the 'Members' section.

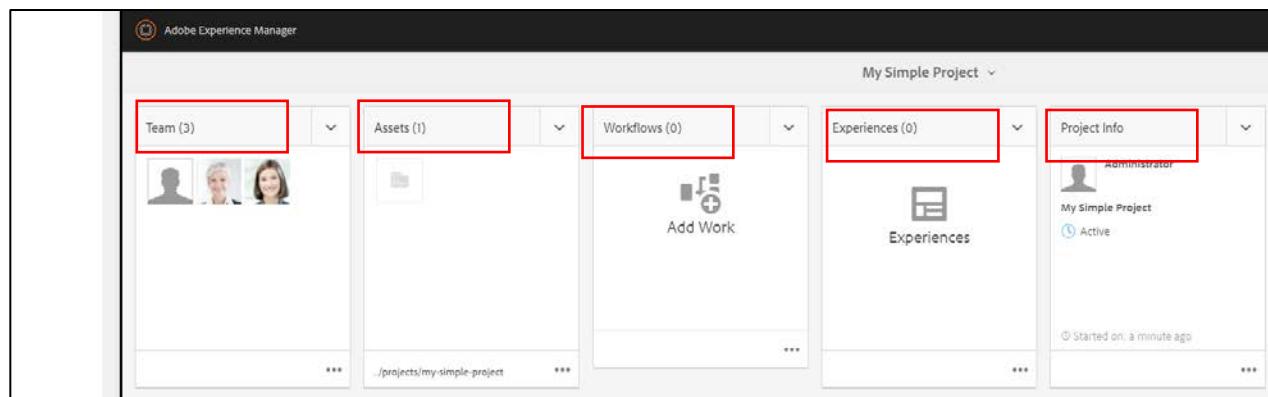
6. Click **Create** in the top right corner. The **Success** pop-up window opens.

7. Click **Open**. The project is created, as shown:



8. Observe different types of tiles, as shown:

- **Team** enables you to add or edit members of this project.
- **Assets** is a working area for assets in this project.
- **Workflows** enables you to run project workflows based on the team for this project.
- **Experiences** are all the pages that are being used in this project.
- **Project info** shows all the metadata information of this project.



## Exercise 5: Create a project template

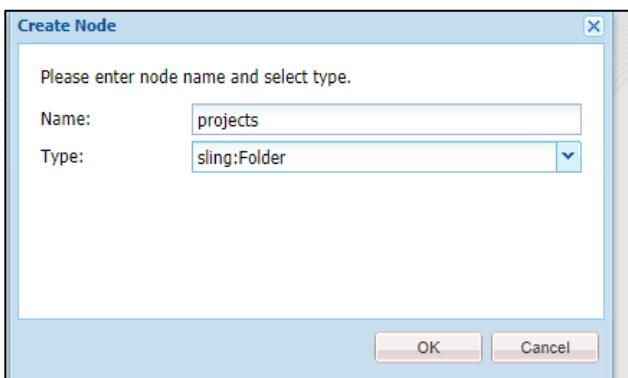
When creating custom business processes in AEM, a custom project might be required. In this exercise, you will create a project template, remove unnecessary tiles from the template, add a custom workflow, and add a new group called reviewers.



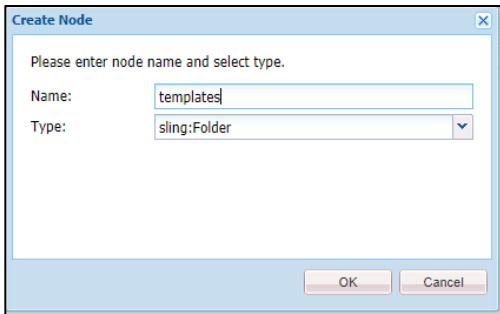
**Note:** A content package is given in Exercise\_Files (training-project-template.zip) with the completed project template. You can either install this content package or go through the exercise on your own. If you install the content package, go to step 19, the testing part of this exercise.

1. In CRXDE Lite, navigate to `/apps/trainingproject`, right-click **trainingproject**, and click **Create > Create node**. Enter the following values, as shown:

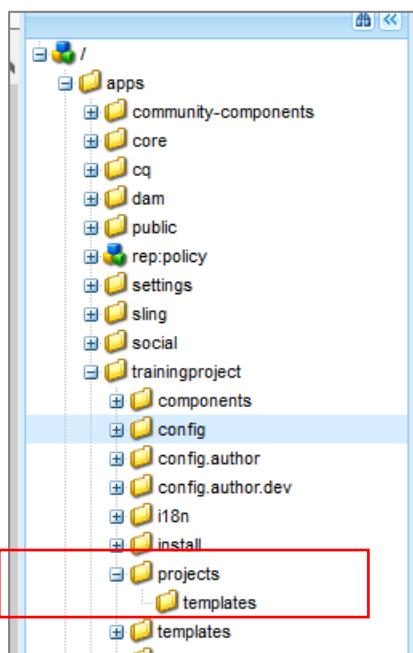
- Enter **projects** in the **Name** field.
- Select **sling:Folder** from the **Type** drop-down menu.



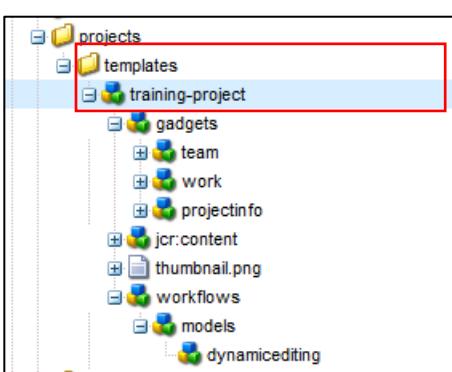
2. Click **OK**. The **projects** node is created.
3. Click **Save All**. Your changes are saved.
4. Right-click **projects**, and click **Create > Create node**. Enter the following values, as shown:
  - Enter **templates** in the **Name** field.
  - Select **sling:Folder** from the **Type** drop-down menu.



5. Click **OK**. The node is created.
6. Click **Save All**. Your changes are saved, as shown:



7. Copy `/libs/cq/core/content/projects/templates/default` to `/apps/trainingproject/projects/templates`.
8. Right-click **default**, and rename to **training-project**, as shown:



9. Click **Save All**. Your changes are saved.

10. Edit the following properties of the node **training-project** with the following values, as shown:

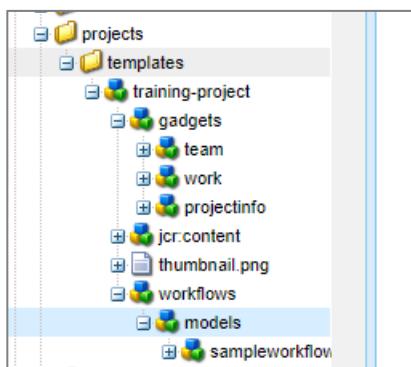
- jcr:title- Enter **Training Publishing Project** for the **Value** field.
- jcr:description- Enter **Create an ACL Project** for the **Value** field.

Properties		Access Control	Replication	Console	Build Info
Name	Type	Value			
1 includeInCreateProject	Boolean	true			
2 jcr:created	Date	2018-04-05T19:53:58.520-07:00			
3 jcr:createdBy	String	admin			
4 jcr:description	String	Create an ACL Project			
5 jcr:primaryType	Name	cq:Template			
6 jcr:title	String	Training Publishing Project			
7 ranking	Long	1			
8 wizard	String	/libs/cq/core/content/projects/wizard/steps/defaultproject.html			

11. Click **Save All**. Your changes are saved.

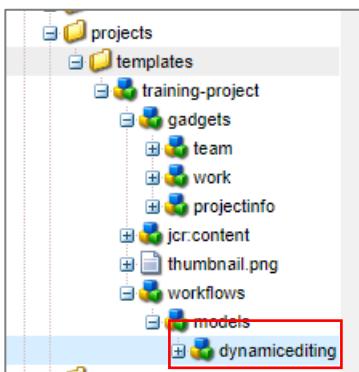
12. Delete the following under **training-project** :

- gadgets/experiences
- gadgets/asset
- Workflows/models/launch
- Wokflow/models/landing
- Workflow/models/email



13. Click **Save All**. Your changes are saved.

14. Rename workflows/models/sampleworkflow to dynamicediting, as shown:



15. Edit the property, as shown:

- modelId- Enter `/etc/workflow/models/dynamicediting/jcr:content/model` in the Value field.

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				



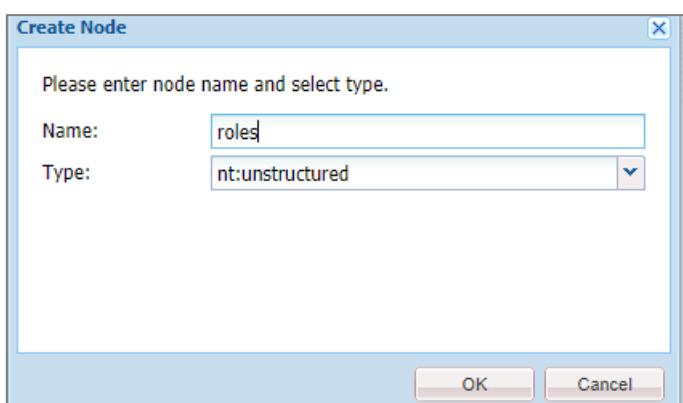
**Note:** This workflow named **dynamicediting** (the value you entered for the modelId in the current step) does not exist yet, you will create it later.

16. Click **Save All**. Your changes are saved.

17. Right-click **training-project** node, and click **Create > Create node**. The **Create Node** dialog box opens.

18. In the **Create Node** dialog box, enter the following values, as shown:

- Name: **roles**
- Select **nt:unstructured** from the Type dropdown.

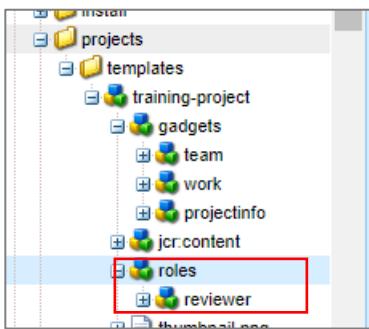


+

19. Click **OK**. The **roles** node is created.

20. Click **Save All**. Your changes are saved.

21. Navigate to `/libs/cq/core/content/projects/templates/retail-product-photoshoot/roles`, and copy the reviewer node to the /roles node under the training-project node, as shown:



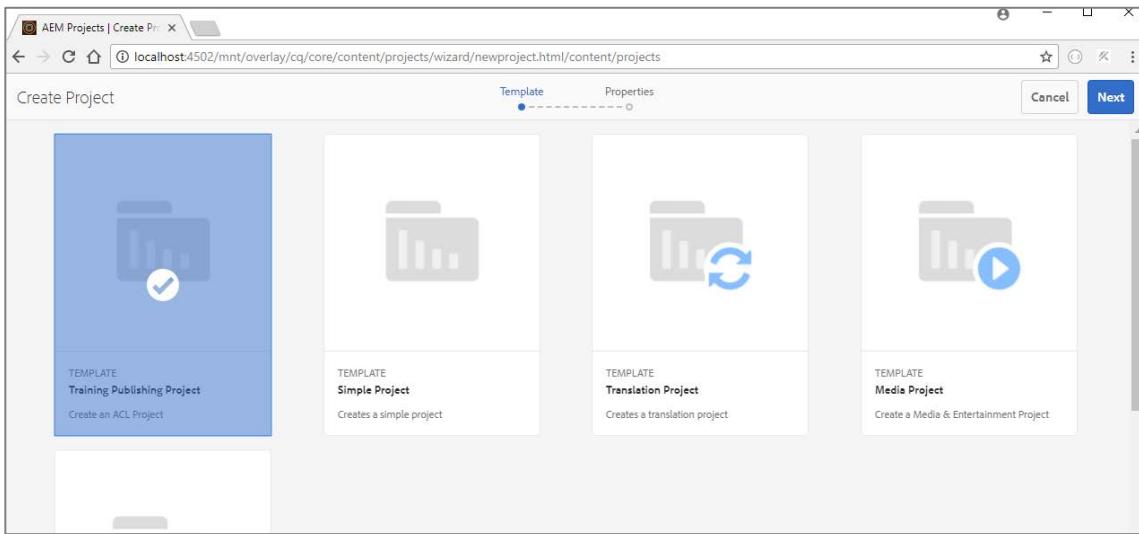
22. Click **Save All**. Your changes are saved.

23. Navigate to the **AEM > Projects**. The **Projects** Console opens, as shown:

A screenshot of the AEM Projects Console. The left sidebar shows 'MASTER PROJECTS' with a list item 'Masters'. The main area shows a project card for 'PROJECT We.Retail' with details: 'Reference Implementation' and 'Active'. A timestamp '10/27/16 9:54 PM' is also visible.

24. Select **Create > Project**. The template opens.

25. Select Training Publishing Project, and click Next. The Create Project window opens.



26. Enter the values for the project, as shown:

- Enter **Pages for Publishing** for the **Title** field.
- Enter **Carlene Avery** in the **User** field, ensure the dropdown beside it is set to **Editors**, and click **Add**.
- Enter **Iris Mccoy** in the **User** field, ensure the dropdown beside it is set to **Reviewers**, and click **Add**.

The screenshot shows a 'Properties' dialog box with the following fields:

- Title**: Pages for Publishing
- Description**: (empty)
- Start Date**: (empty)
- Due Date**: (empty)
- User**: Carlene Avery (Editors) - Add button
- Members** table:
 

	Administrator	Owners
	Iris McCoy	iris.mccoy@we-retail.net
	Carlene Avery	cavery@we-retail.net
	Reviewers	
	Editors	

27.Click **Create** in the top right corner. The **Success** pop-up window opens.

28.Click **Open** in the pop-up window. Observe the changes of the custom project, as shown:

The screenshot shows the Adobe Experience Manager interface with the following components:

- Team (3)**: Shows three user profiles.
- Workflows (0)**: Shows an "Add Work" button.
- Project Info**: Shows the following details:
  - Administrator
  - Pages for Publishing
  - Active
  - Started on: a few seconds ago

## Executing an Existing Workflow

AEM 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 workflow goes through a series of steps until its completion.

## Defining Workflow Models

Based on your business requirements, you can modify an existing workflow or create 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.

In AEM, there are a number of steps available for workflows, such as Participant, Process, Create Task, Delete Node, Dialog Participant, Dynamic Participant, and Form Participant. Each step can contain any number of actions and associated conditions. For example, a step in a publish workflow may involve the step- *approval from an editor*. Two of the most commonly used workflow steps are *Participant* and *Process*.

### Participant Step

The Participant step requires manual intervention by a person to advance the workflow. It enables you to assign a step to a user or a group of users. If the workflow is assigned to just one user, 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, all those users need to complete the task.

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 and immediate, and if you want to specify specific blocks of time, you can select 1h, 6h, 12h, and 24h. The timeout handler controls the workflow when the step times out.

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.

## Process Step

The Process step involves automatic actions that are executed by the system if specific conditions are met. This step:

- Executes an ECMA script or calls an OSGi service to automate the process.
- Offers the following built-in processes:
  - Workflow control processes: Control the behavior of the workflow and do not perform any action on content
  - Basic processes: Delete a node, or logs a debug message
  - WCM processes: Perform WCM-related tasks, such as activating a page or confirming registration
  - Versioning processes: Perform version-related tasks, such as creating versions of the payload
  - DAM processes: Perform DAM-related tasks, such as creating thumbnails, creating sub-assets, and extracting metadata
  - Collaboration processes: Are related to the collaboration features of AEM, such as the collaboration with social communities.

## Developing Custom Steps

You can extend workflow steps with scripts to provide more functionality and control. You can create customized process steps by using the following methods:

- Java class bundles: Create a bundle with the Java class, and then deploy the bundle into the OSGi container by using the Web console.
- ECMA scripts: Scripts are located in JCR 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.

## Creating a 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.

## 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 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>), as shown:

Workflow Launchers							
	Description	Event Type	Nodetype	Condition	Workflow	Enabled	Exclude
<input type="checkbox"/>	Globbing /content/dam(/.*)renditions/original						
<input type="checkbox"/>	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"/>	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/scnItems	/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 will 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 workflowPath: `/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 to which the launcher applies

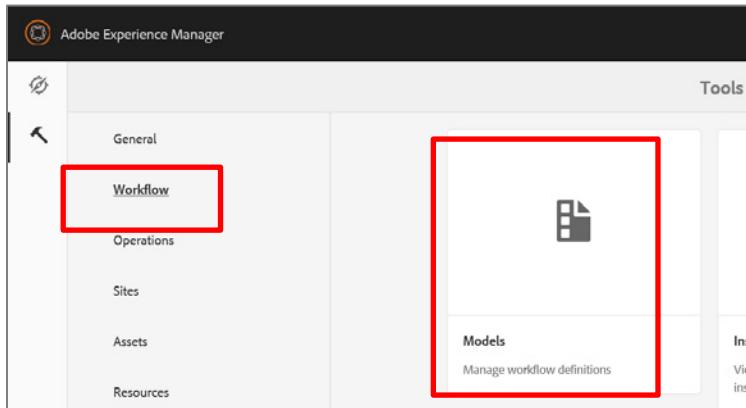
The following table differentiates Workflow Launchers and JCR Observations or Sling events:

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
The 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

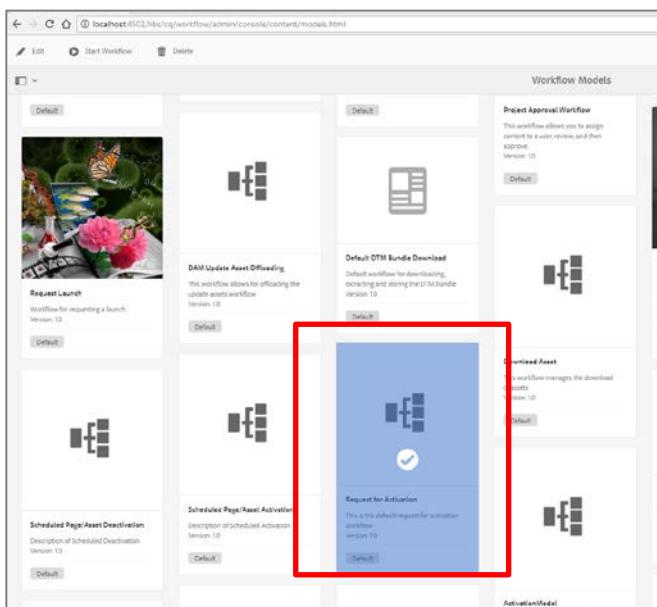
## Exercise 6: Execute a workflow

In this exercise, you will 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.

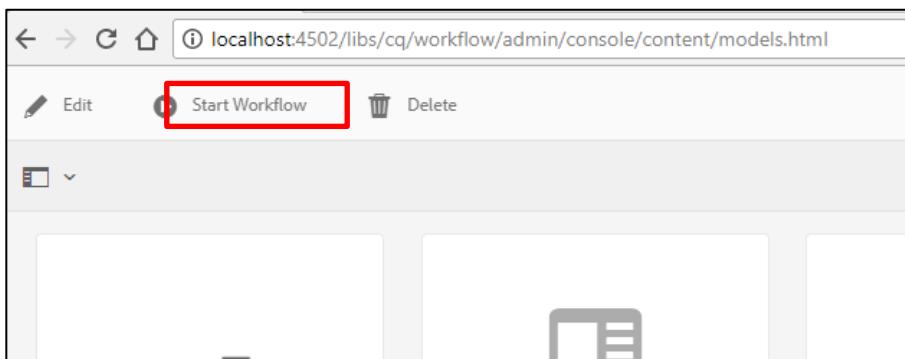
1. In AEM, navigate to Tools > Workflow > Models, as shown:



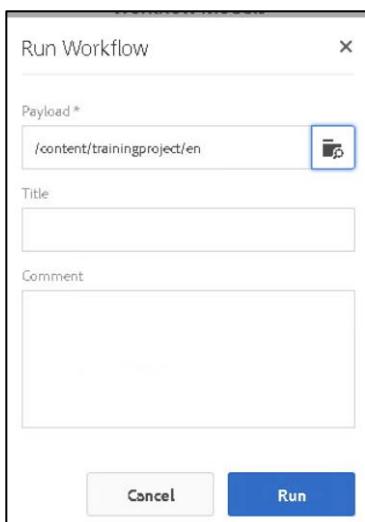
2. Scroll down to locate the **Request for Activation** workflow/card.
3. Click the **checkmark** on the **Request for Activation** workflow/card. The **Request for Activation** workflow/card is selected. A blue highlight appears, as shown.



4. Click **Start Workflow** in the top left corner, as shown. The **Run Workflow** dialog box opens.

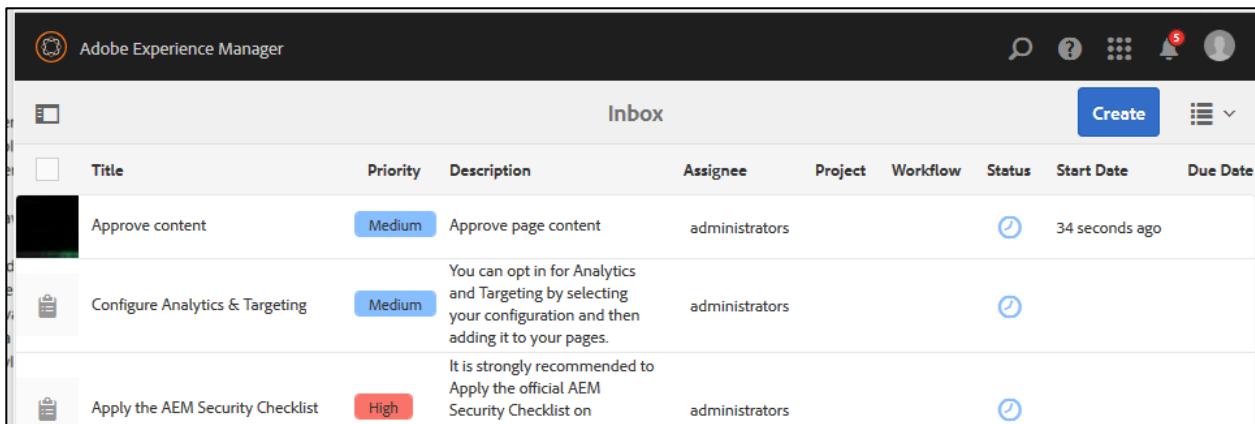


5. Enter the page location, **/content/trainingproject/en** in the mandatory **Payload** field, as shown:



6. Click **Run**. The workflow process will begin.

7. Click the **Inbox** at the top right corner. A notification will appear in the **Inbox** (top right) to approve page content, as shown:





**Note:** The page will not be published until it is approved. Recall the process is of two steps:

1. Edit an initial version of the page.
2. Publish the page.

Each of these steps requires approval.

## Exercise 7: Build and test a workflow

In this exercise, you will create a workflow with two tasks - editing and approving. After the content is approved, it will be published. If it is not approved, it will go back to the editor for a proper feedback loop.

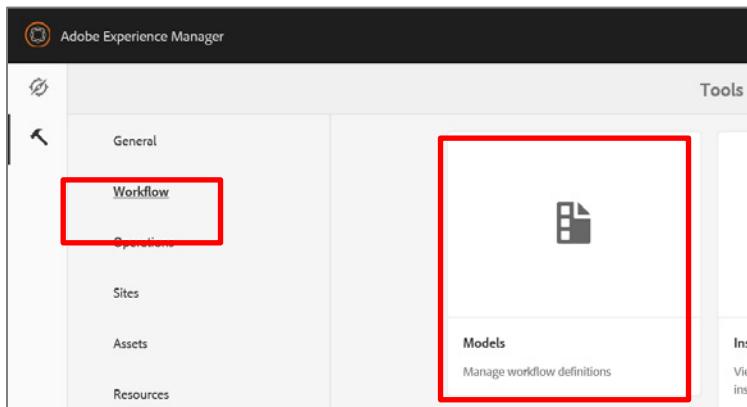
Following are the two tasks in this exercise:

1. Build a workflow
2. Test the workflow

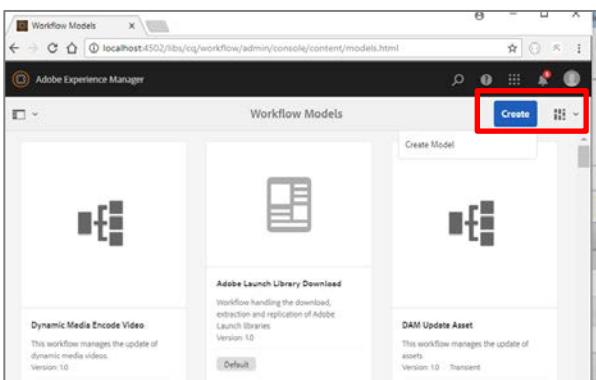
 **Note:** A content package is given in the Exercise\_Files ([training-project-workflow.zip](#)) with the completed project template. You can either install this content package or go through the exercise on your own. If you install the content package, go to step 32, the testing part of this exercise.

### Task 1: Build a workflow

1. In AEM, navigate to Tools > Workflow > Models, as shown. The **Workflow Models** window opens.

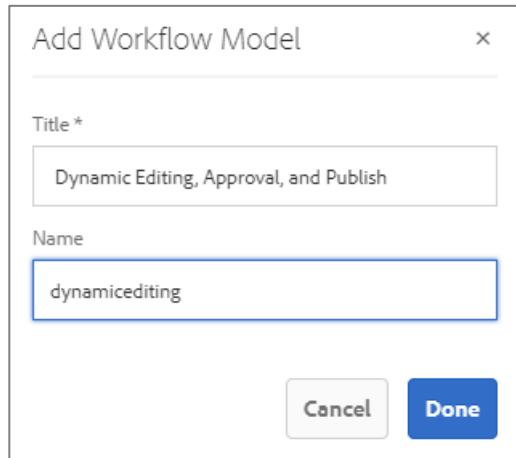


2. Select Create > Create Model, as shown:



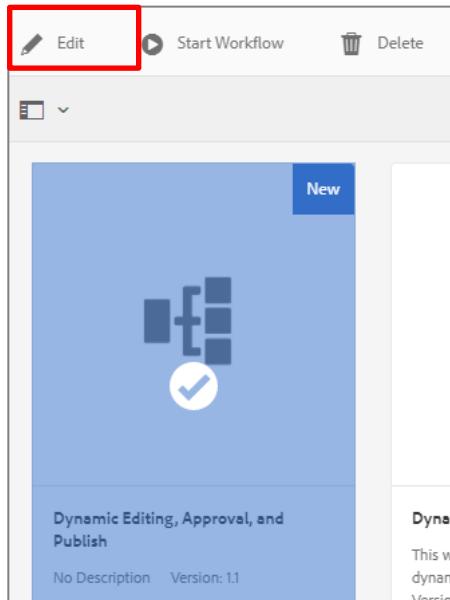
3. In the **Add Workflow Model** window, enter the values, as shown:

- Title: **Dynamic Editing, Approval, and Publish**
- Name: **dynamicediting**

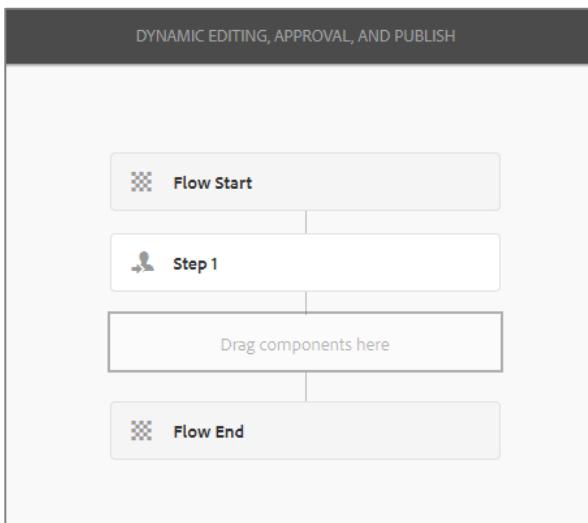


4. Click Done. The Dynamic Editing, Approval, and Publish model is created.

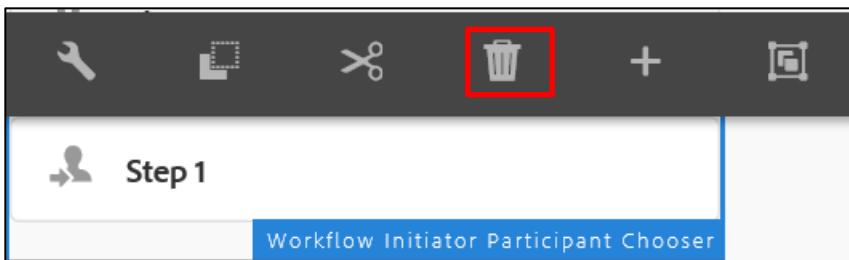
5. In the **Workflow Models** window, select the checkmark on the model **Dynamic Editing, Approval, and Publish** and click **Edit**, as shown:



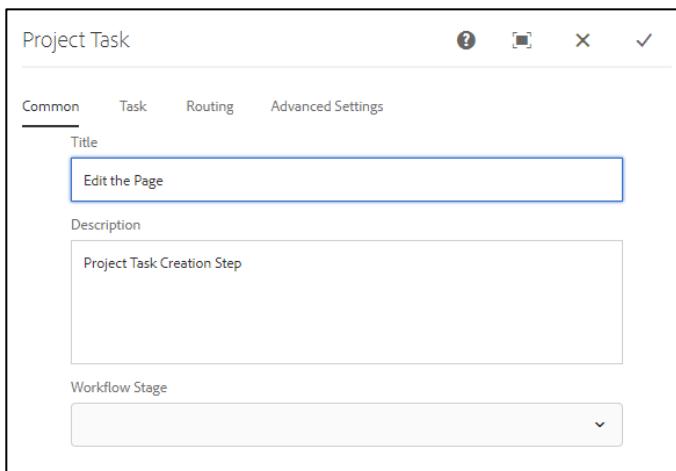
The Workflow Model **Dynamic Editing, Approval, and Publish** opens for editing, as shown:



6. Delete the **Step 1** component by double-clicking it and selecting the Delete icon from the menu bar, as shown.



7. Drag the **Create Project Task** component from the left-hand side into the **Drag components here** box.
8. Double-click Project Task Creation Step. The **Project Task** dialog box opens.
9. Select the **Common** tab, and enter the title, shown:

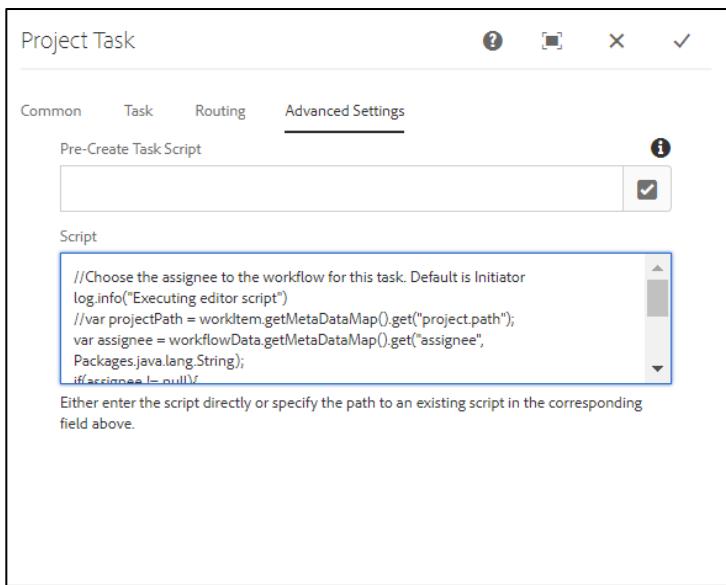


---

10. Click the **Task** tab and enter the following details:

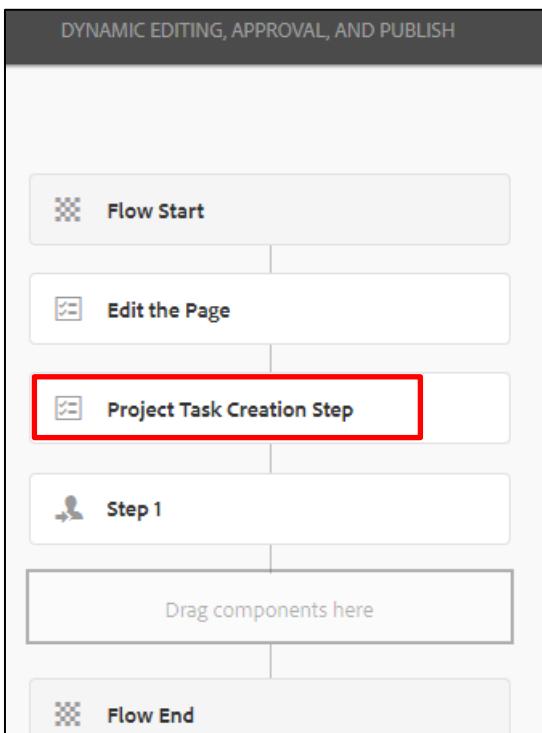
- Name: **Edit the Page**
- Select the **Email** checkbox.
- Description: **Make the appropriate edits to the page and complete this task.**

11. Select the **Advanced Settings** tab, and copy and paste the contents from the *Editor Script /Exercise\_Files/12\_Deep\_Dive\_into\_AEM\_APIs/workflow-script.txt*, as shown:



12. Click the Done icon (checkmark) on the top of the dialog box to save the changes.

13. Drag and drop another **Create Project Task** component from the left-hand side below **Edit the Page** in the model, as shown:

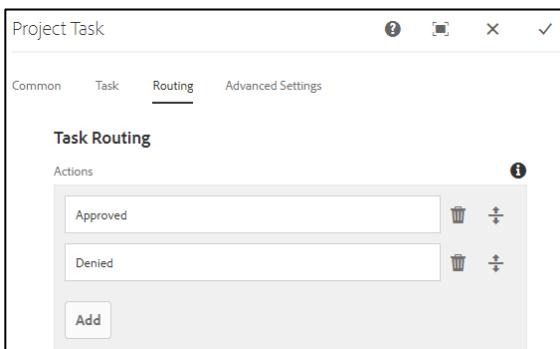


14. Double-click Project Task Creation Step. The Project Task dialog box opens.
15. Select the **Common** tab, and enter the title as **Approval**.
16. Select the **Task** tab, and enter the following details:

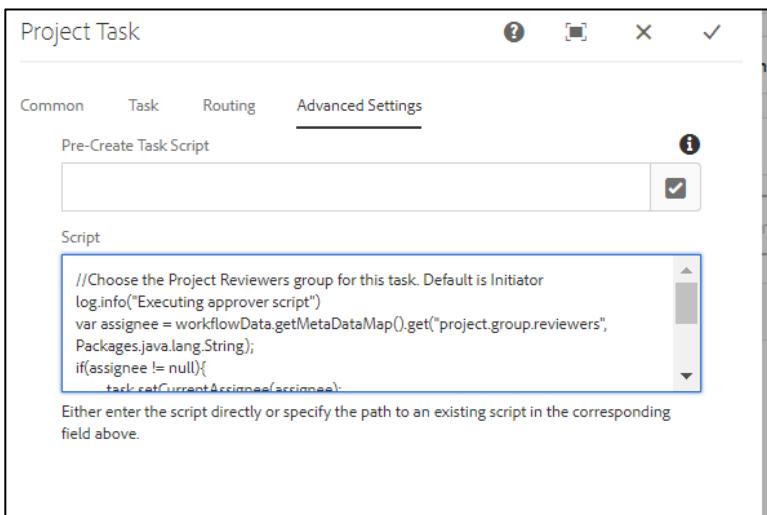
- Name: **Approval Task**
- Select the **Email** checkbox.
- Description: **Review the page for approval to publish.**

17. Select the **Routing** tab, and add the following actions, as shown:

- Approved
- Denied

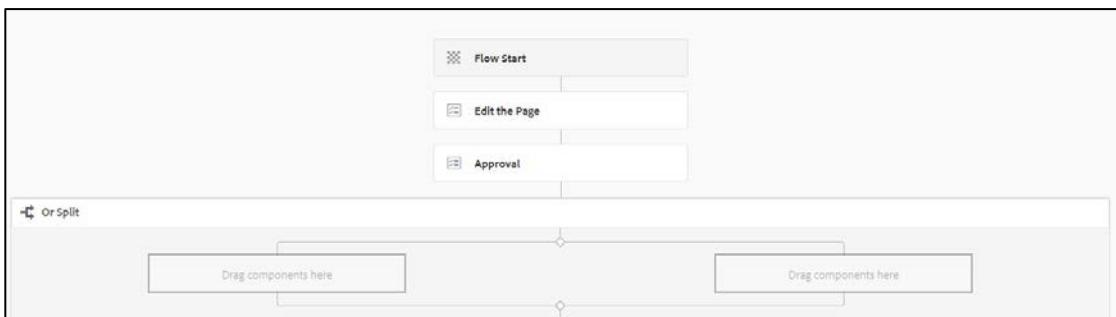


18. Select the **Advanced Settings** tab, and copy and paste the contents of *Approver Script* from */Exercise\_Files/12\_Deep\_Dive\_into\_AEM\_APIs/workflow-script.txt*, as shown:

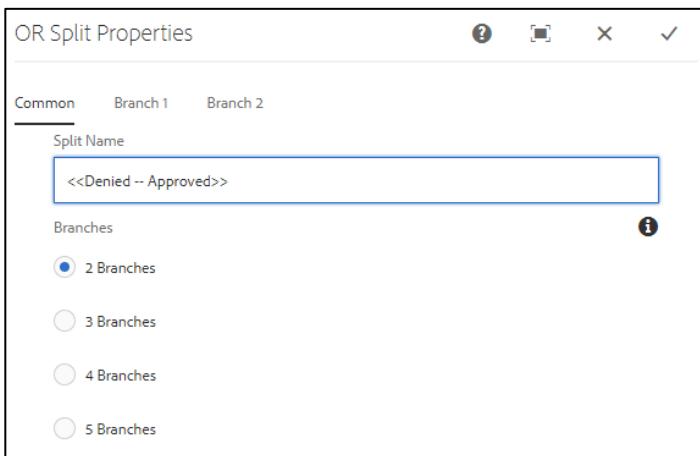


19. Click the Done icon (checkmark) on the top of the dialog box to save the changes.

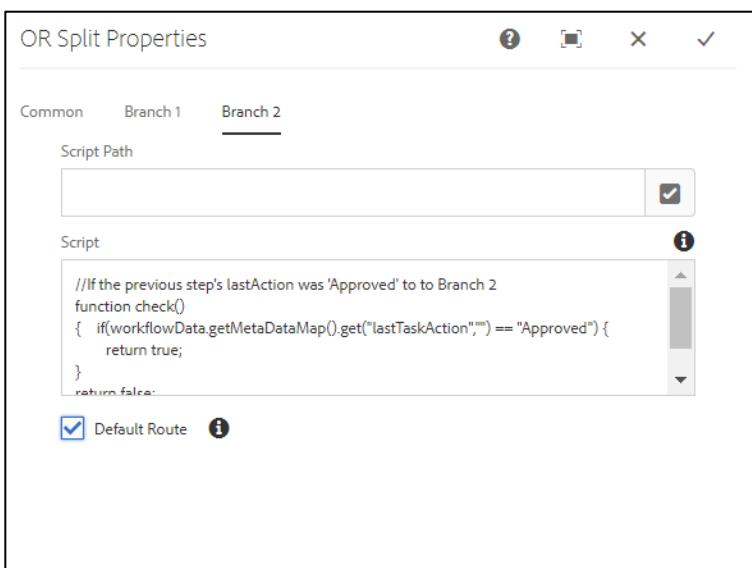
20. Drag another **OR Split** component from the left-hand side below **Approval** in the model, as shown:



21. Double-click **OR Split**. The **OR Split Properties** dialog box opens.
22. Select the **Common** tab and enter the title as **<<Denied—Approved>>** and select the **2 Branches** option button, as shown:

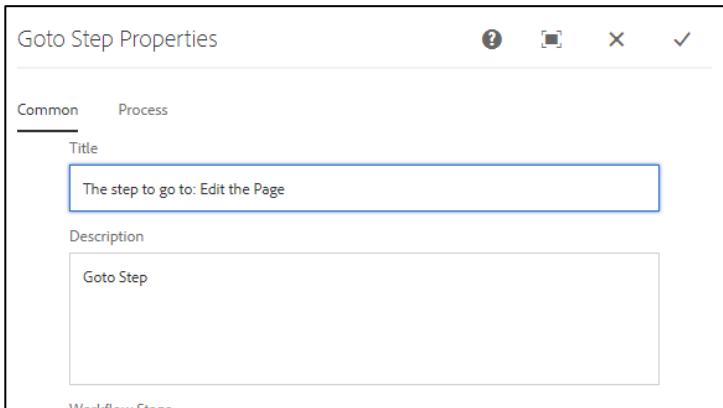


23. Select the **Branch1** tab, and copy and paste the contents of *Or Split Branch 1 Scripts* from **/Exercise\_Files/12\_Deep\_Dive\_into\_AEM\_APIs/workflow-script.txt**.
24. Select the **Branch2** tab, and copy and paste the contents of *Or Split Branch 2 Script* from **/Exercise\_Files/12\_Deep\_Dive\_into\_AEM\_APIs/workflow-script.txt**, as shown.
25. Select the **Default Route** checkbox.

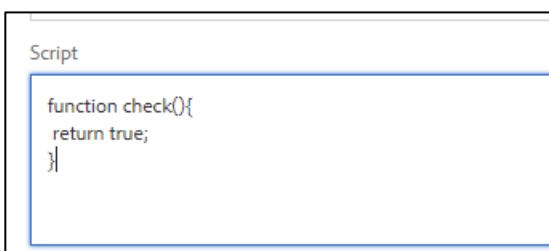



---

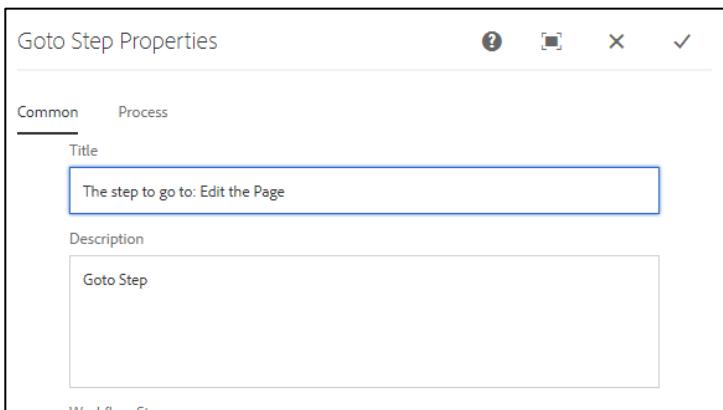
26. Click the Done icon (checkmark) on the top of the dialog box to save the changes.
27. In the left branch (Branch 1), drag the **Goto Step** component and Open the dialog box.
28. Select the **Process** tab, and enter the **Title**, as shown:



29. Script: Copy and Paste the "GOTO Script" from the workflows-script.txt, as shown:

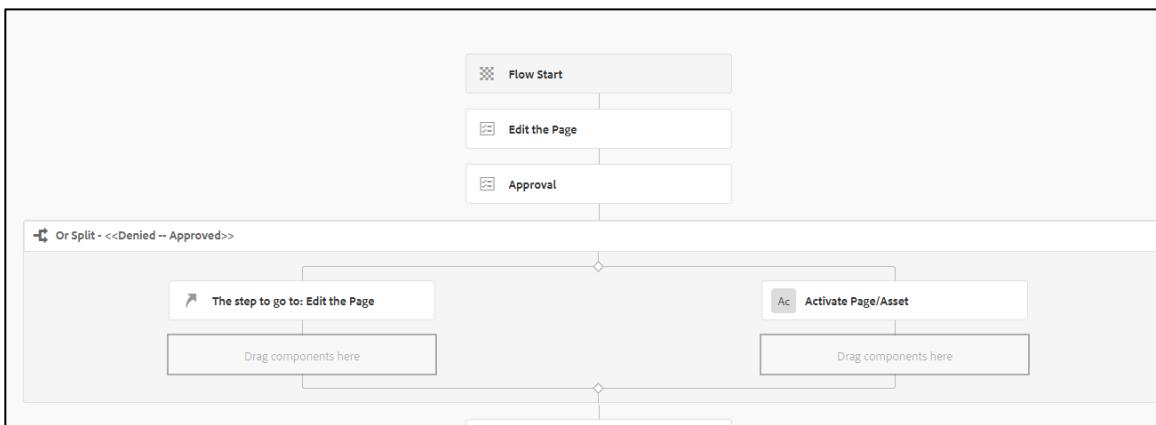


30. Select the **Common** tab, and enter the title, as shown:

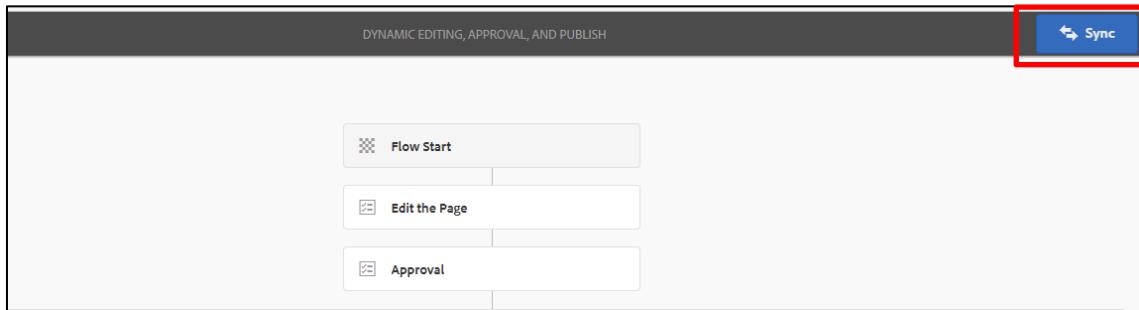


31. Click the Done icon (checkmark) on the top of the dialog box to save the changes.

32. In the right branch (Branch 2), drag the **Activate Page/Asset** component, as shown:



33. At this point the workflow is complete. In the top-right click **Sync**, as shown. This will write the workflow model.



## Task 2: Test the workflow

Recall the custom project you created earlier, you referenced the workflow you just created. To view this connection, go to /apps/trainingproject/projects/templates/training-project/workflows/models/dynamicediting. To test your workflow, you will open the newly created project and test the workflow with it.

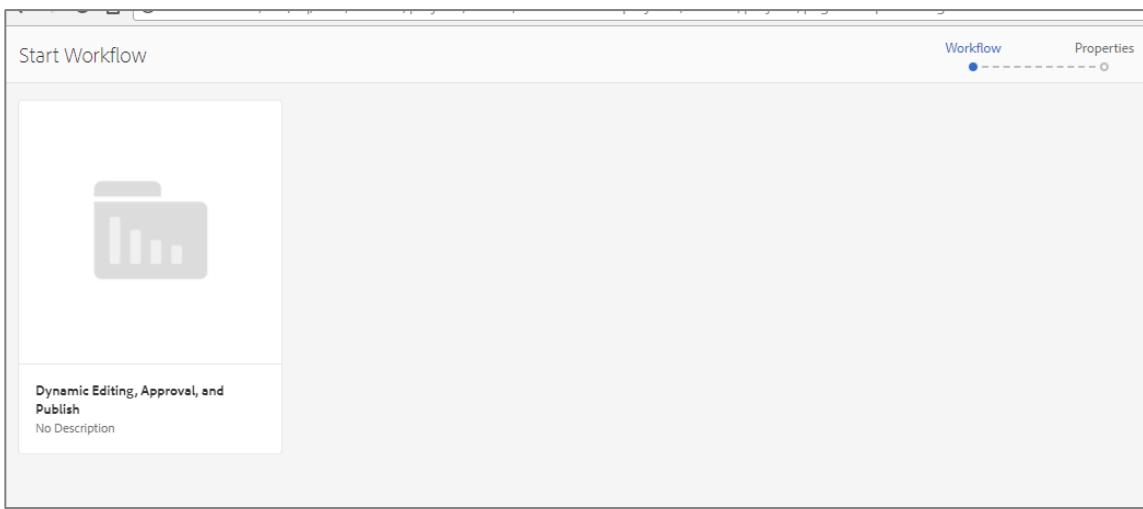
1. Navigate to **AEM > Projects**. The **Projects Console** opens, as shown:

The screenshot shows the Adobe Experience Manager Projects console. At the top, there's a header with the AEM logo, a search icon, a help icon, a grid icon, a notification bell with a red '4' badge, and a user profile icon. Below the header, the word 'Projects' is centered above a grid of cards. On the left, a card for 'MASTER PROJECTS' labeled 'Masters' has a lock icon. On the right, a card for 'PROJECT We.Retail' shows 'Reference Implementation' and 'Active' status, with a timestamp '10/27/16 9:54 PM' at the bottom.

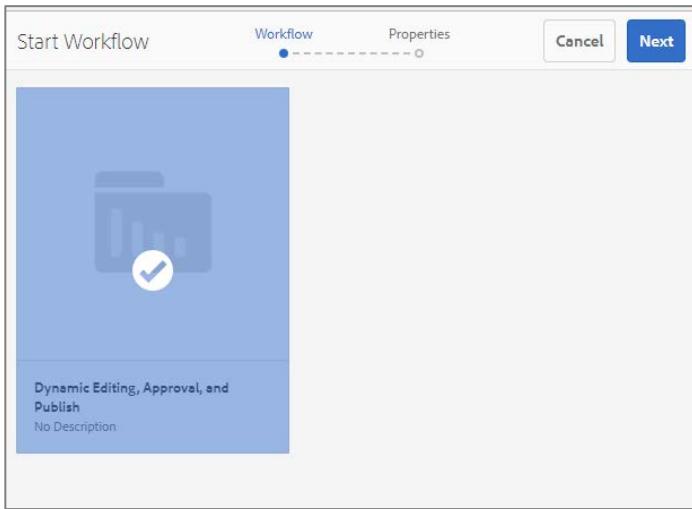
2. Click **Pages for Publishing** Project, and click **Next**. The **Pages for Publishing** window opens, as shown:

The screenshot shows the 'Pages for Publishing' window. It has three main sections: 'Team (3)' showing three user icons, 'Workflows (0)' with an 'Add Work' button, and 'Project Info' for 'Administrator'. The 'Project Info' section includes 'Pages for Publishing' (status 'Active'), a note 'Started on: a day ago', and an ellipsis button.

3. In the **Workflow** tile, click **Add Work**. The **Start Workflow** window opens, as shown:



-  **Note:** If your workflow does not appear, double check the workflow added to the project template under </apps/trainingproject/projects/templates/training-project/workflows/models/dynamicediting>.
4. Select **Dynamic Editing, Approval, and Publish** and click **Next**, as shown. The Properties window opens.



5. Enter the details, as shown:

- Title: **Edit the Equipment Page**
- Assign To: **Editors**
- Content Path: **/content/we-retail/language-masters/en/equipment**

The screenshot shows a 'Workflow Properties' dialog box. It has tabs for 'Workflow' and 'Properties'. The 'Properties' tab is active. The fields are as follows:

- Title:** Edit the Equipment
- Assign To:** Editors
- Description:** (empty)
- Content Path:** /content/we-retail/language-masters/en/equipment (highlighted with a blue border)
- Task Priority:** Medium
- Due Date:** (calendar icon)

6. Click **Submit** in the top right corner. The **Workflow** is built, as shown:

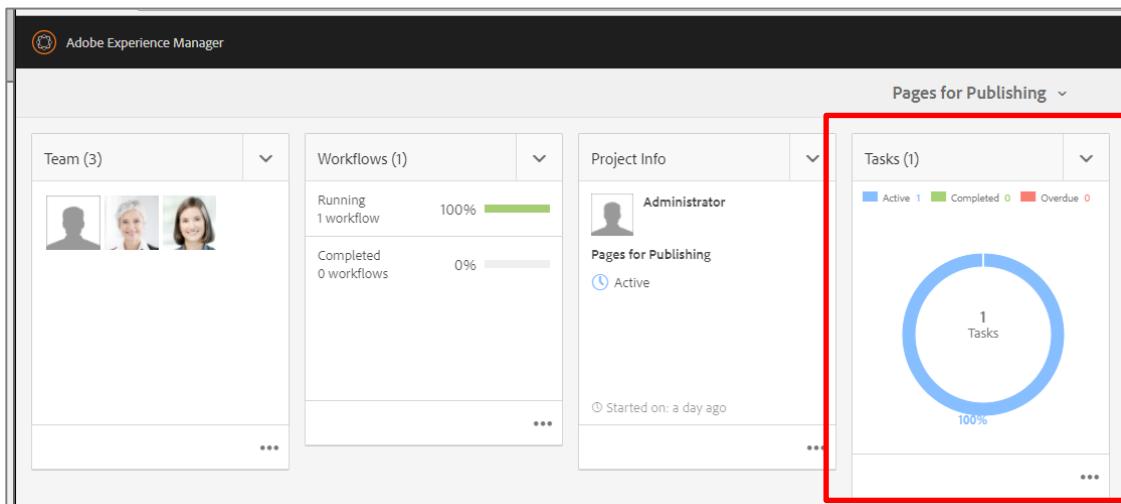
The screenshot shows the 'Pages for Publishing' dashboard with four main sections:

- Team (3):** Shows three user profiles.
- Workflows (1):** Shows 'Running 1 workflow' at 100% completion and 'Completed 0 workflows' at 0% completion.
- Project Info:** Shows 'Administrator' assigned to 'Pages for Publishing' with the status 'Active'. It also notes 'Started on: a day ago'.
- Tasks (0):** Shows an 'Add Task' button.



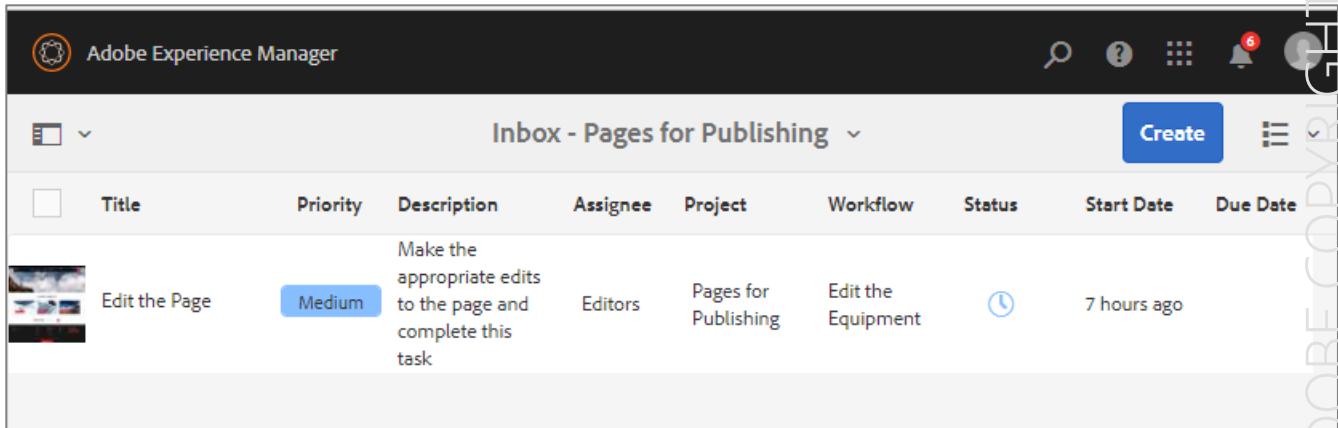
**Note:** At this point, you should be able to walk through the edit/approval steps. Even though the tasks are assigned to different project groups, the admin can see and run through all the tasks.

7. Refresh the browser and notice the new **Task**, as shown:



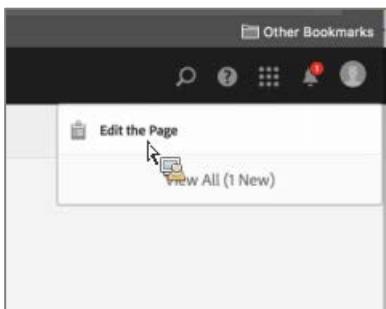
The screenshot shows the Adobe Experience Manager dashboard with a red box highlighting the 'Tasks (1)' section. The tasks are represented by a blue circle with the number '1' and '100%' completion. Below the circle, there is a legend: Active (1), Completed (0), and Overdue (0).

8. Click the ellipsis on the Task Tile. The Inbox opens, as shown:



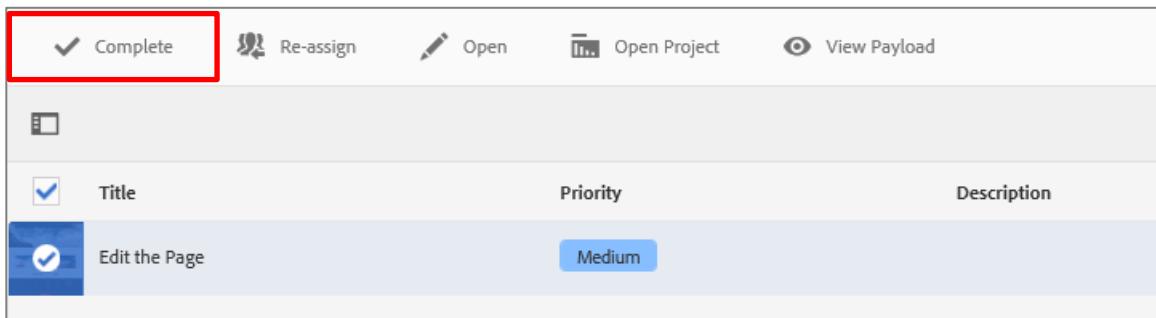
The screenshot shows the 'Inbox - Pages for Publishing' screen. It displays a single task titled 'Edit the Page' with a medium priority. The task description is: 'Make the appropriate edits to the page and complete this task'. The assignee is 'Editors', the project is 'Pages for Publishing', and the workflow is 'Edit the Equipment'. The status is 'Active', and it was last updated 7 hours ago.

9. Find the new message in the inbox, as shown:

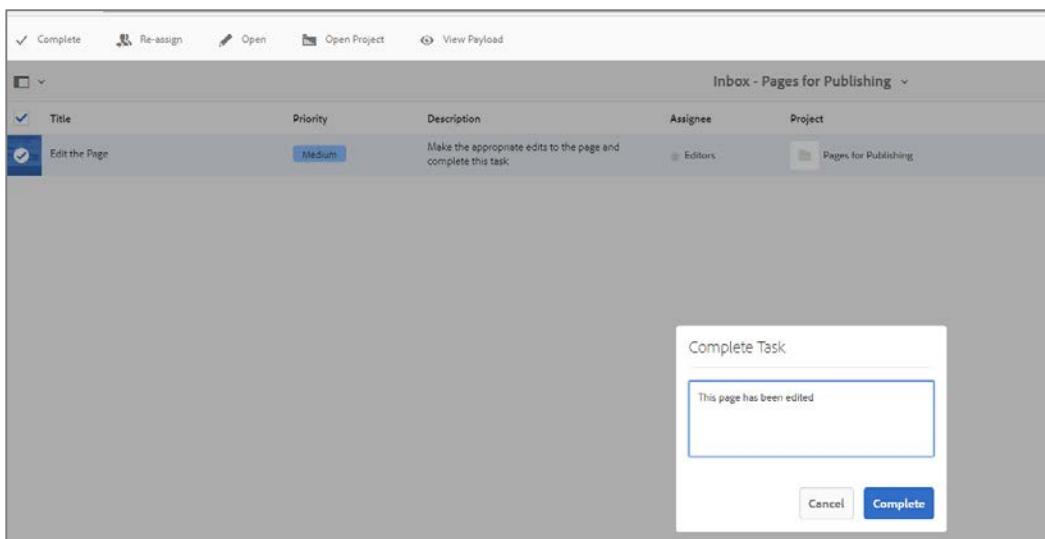


The screenshot shows a web browser window with a bookmark bar labeled 'Other Bookmarks'. Below the bookmarks, there is a list of items. The first item is 'Edit the Page' with a small icon next to it. A cursor arrow is pointing at this item. Below the list, there is a link 'View All (1 New)'.

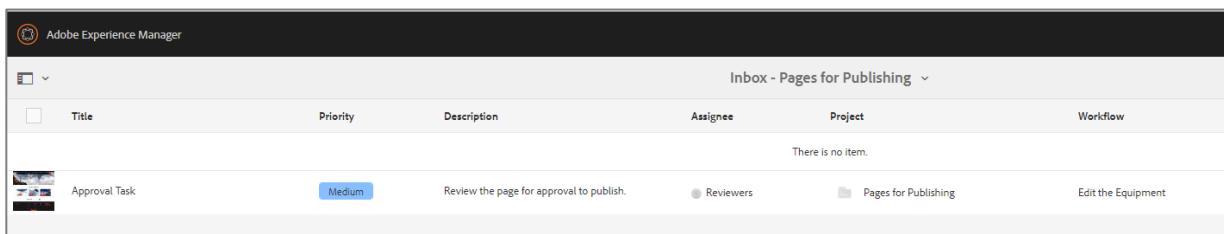
10. Select the **Edit the Page** and click **Complete**, as shown:



11. In the **Complete Task** pop-up window, enter a comment as shown and click **Complete**, as shown. The task is completed.



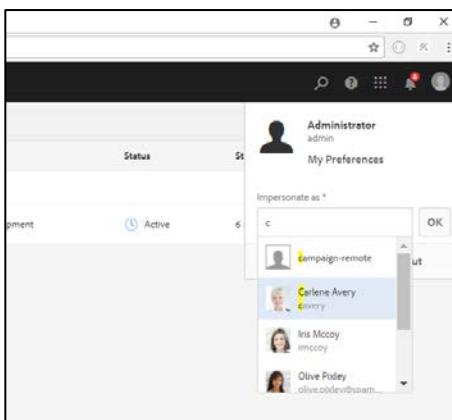
12. Notice there is an **Approval Task**, as shown:



13. Revert to the admin user.

+

14. Impersonate as **Carlene Avery**, and click **OK**, as shown:



15. Select the **Approval Task**, and click **Complete**. The **Complete Task** pop-up window appears. Notice there is a drop-down with **Approved** and **Denied**, as shown:

Title	Priority	Description	Assignee	Project
Approval Task	Medium	Review the page for approval to publish.	Reviewers	Pages for Publishing

Complete Task

Select Action

Approved

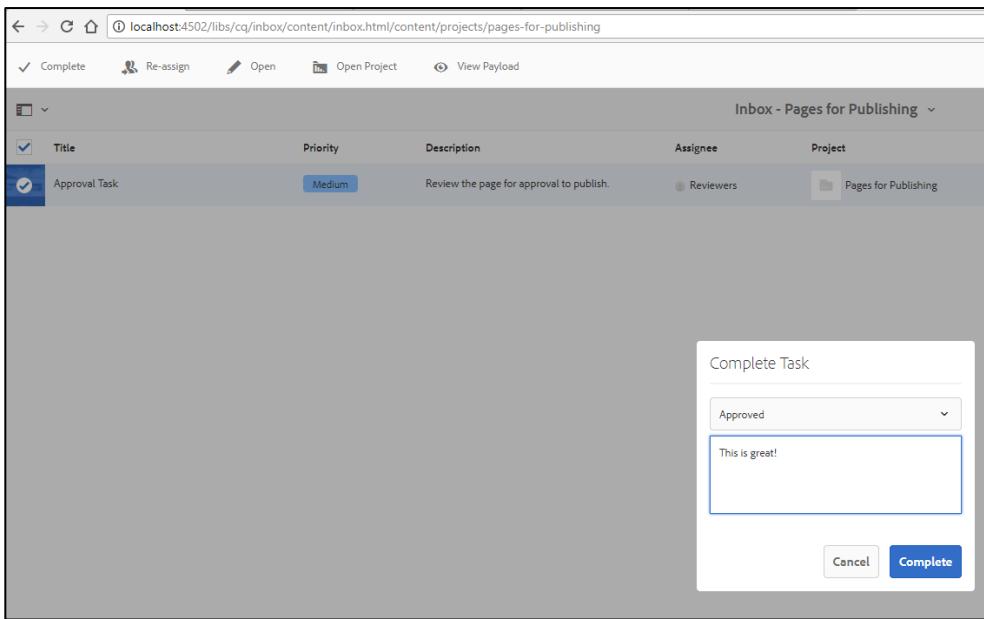
Denied

Cancel Complete

If you select **Denied**, the workflow will step back and create new task and re-edit the page with the comments of the reviewer (cavery).

If you select **Approved**, the workflow will move forward.

16. Complete the task by selecting **Approved** and add a comment, as shown:



17. Verify the page is published, as shown:



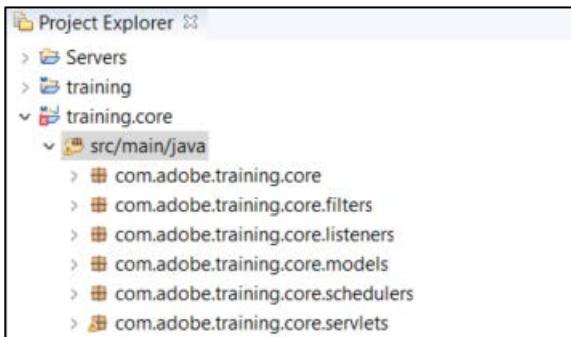
## Exercise 8: Customize the process step

In this exercise, you will customize the process step in the workflow. This exercise has four tasks:

1. Create the custom workflow process
2. Deploy the class
3. Customize the workflow
4. Test the workflow

### Task 1: Create the custom workflow process

1. Double-click the **Eclipse** shortcut on the desktop. The Eclipse Launcher opens.
2. Specify the Workspace as **C:\Workspace** and click **Launch**. The Workspace opens.
3. In Project Explorer, navigate to **training.core > src/main/java**, as shown:

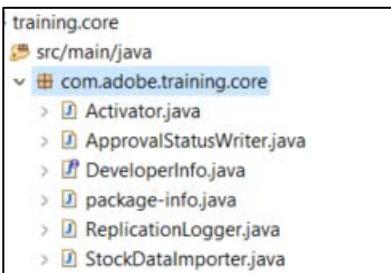


4. Copy the file **ApprovalStatusWriter.java** from **/Exercise\_Files/12\_Deep\_Dive\_into\_AEM\_APIs/**.



**Note:** The code is provided as part of the Exercise\_Files under /Exercise\_Files/12\_Deep\_Dive\_into\_AEM\_APIs/. Copy and paste the file from the exercise files referenced to ApprovalStatusWriter.java to Eclipse. Please do not copy the code from this student guide. The code in the student guide is for illustrative purposes only.

5. In Eclipse, right-click **com.adobe.training.core**. and paste the file. The **ApprovalStatusWriter.java** class is created, as shown:



6. Double-click **ApprovalStatusWriter.java**. The file opens in the editor.

```
1. package com.adobe.training.core;
2.
3. import com.adobe.granite.workflow.WorkflowSession;
4. import com.adobe.granite.workflow.WorkflowException;
5. import com.adobe.granite.workflow.exec.WorkItem;
6. import com.adobe.granite.workflow.exec.WorkflowProcess;
7. import com.adobe.granite.workflow.exec.WorkflowData;
8. import com.adobe.granite.workflow.metadata.MetaDataMap;
9.
10. import org.apache.sling.api.resource.ModifiableValueMap;
11. import org.apache.sling.api.resource.PersistenceException;
12. import org.apache.sling.api.resource.Resource;
13. import org.apache.sling.api.resource.ResourceResolver;
14. import org.osgi.service.component.annotations.Component;
15. import org.osgi.framework.Constants;
16.
17. @Component(service = WorkflowProcess.class,
18.             property = {Constants.SERVICE_DESCRIPTION + "=Workflow process to set status to ap-
19.               proved",
20.                         Constants.SERVICE_VENDOR + "=Adobe",
21.                         "process.label=Approval Status Writer"
22.                   })
23.
24. public class ApprovalStatusWriter implements WorkflowProcess {
25.
26.     private static final String TYPE_JCR_PATH = "JCR_PATH";
27.
28.     @Override
29.     public void execute(WorkItem item, WorkflowSession workflowSession, MetaDataMap args) throws
30. WorkflowException {
31.         WorkflowData workflowData = item.getWorkflowData();
32.         if (workflowData.getPayloadType().equals(TYPE_JCR_PATH)) {
33.             String path = workflowData.getPayload().toString() + "/jcr:content";
34.             try (ResourceResolver rr = workflowSession.adaptTo(ResourceResolver.class)){
35.                 Resource resource = rr.getResource(path);
36.                 resource.adaptTo(ModifiableValueMap.class).put("approved", readArgument(args));
37.             }
38.             rr.commit();
39.         }
40.     }
}
```

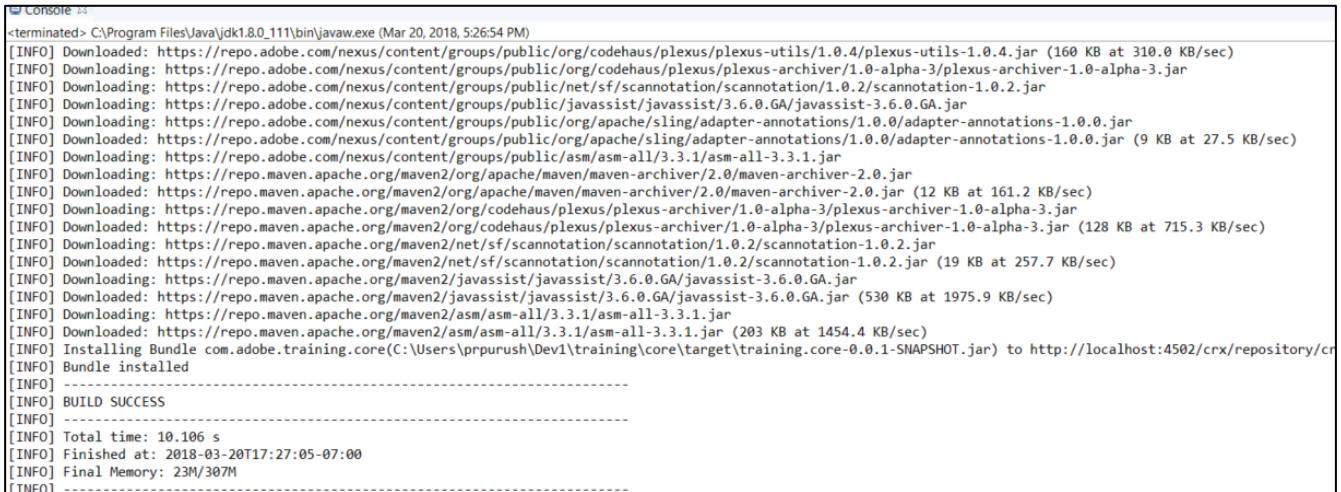
```

41.    }
42.
43.    private static boolean readArgument(MetaDataMap args) {
44.        String argument = args.get("PROCESS_ARGS", "false");
45.        return argument.equalsIgnoreCase("true");
46.    }
47. }
48.

```

## Task 2: Deploy the class

1. In Project Explorer, right-click **training.core** and choose **Run As > Maven install**. The build starts.
2. Verify the bundle installed successfully, as shown:



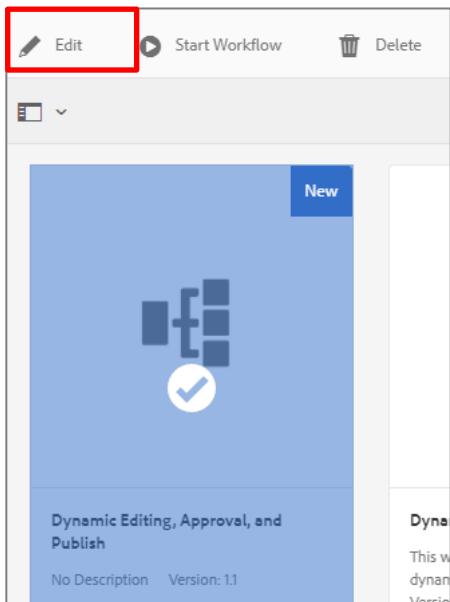
```

[terminated> C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Mar 20, 2018, 5:26:54 PM)
[INFO] Downloaded: https://repo.adobe.com/nexus/content/groups/public/org/codehaus/plexus/plexus-utils/1.0.4/plexus-utils-1.0.4.jar (160 KB at 310.0 KB/sec)
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar
[INFO] Downloaded: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar (9 KB at 27.5 KB/sec)
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/asm/asm-all/3.3.1/asm-all-3.3.1.jar
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.0/maven-archiver-2.0.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.0/maven-archiver-2.0.jar (12 KB at 161.2 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar (128 KB at 715.3 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar (19 KB at 257.7 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar (530 KB at 1975.9 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/asm/asm-all/3.3.1/asm-all-3.3.1.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/asm/asm-all/3.3.1/asm-all-3.3.1.jar (203 KB at 1454.4 KB/sec)
[INFO] Installing Bundle com.adobe.training.core(C:\Users\pprurush\Dev\training\core\target\training.core-0.0.1-SNAPSHOT.jar) to http://localhost:4502/crx/repository/cr
[INFO] Bundle installed
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 10.106 s
[INFO] Finished at: 2018-03-20T17:27:05-07:00
[INFO] Final Memory: 23M/307M
[INFO] -----

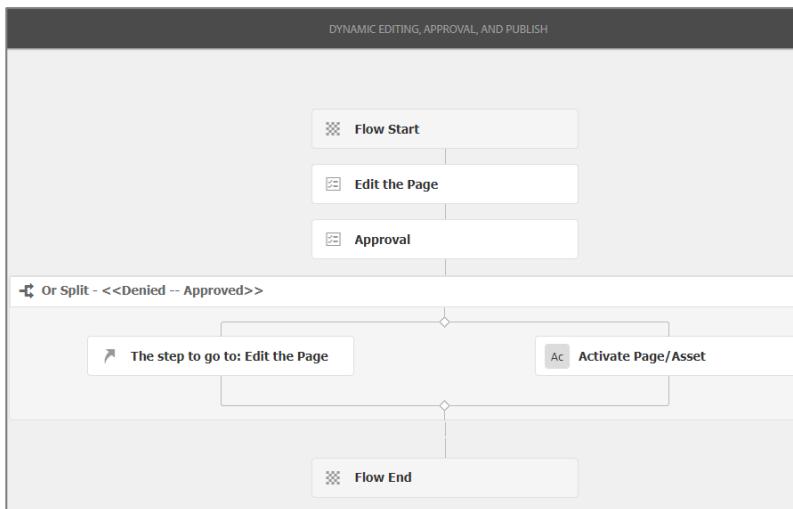
```

## Task 3: Customize the workflow

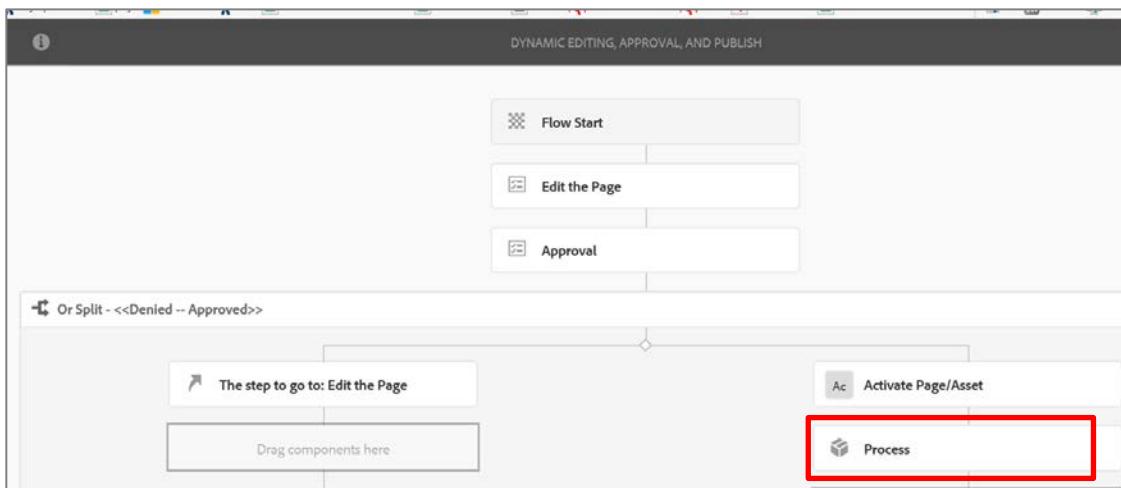
1. In the Workflow Models window, select Dynamic Editing, Approval, and Publish and click **Edit**, as shown:



The Workflow Model **Dynamic Editing, Approval, and Publish** opens for editing, as shown:



2. Drag the **Process Step** component from the left-hand side into the model, as shown:



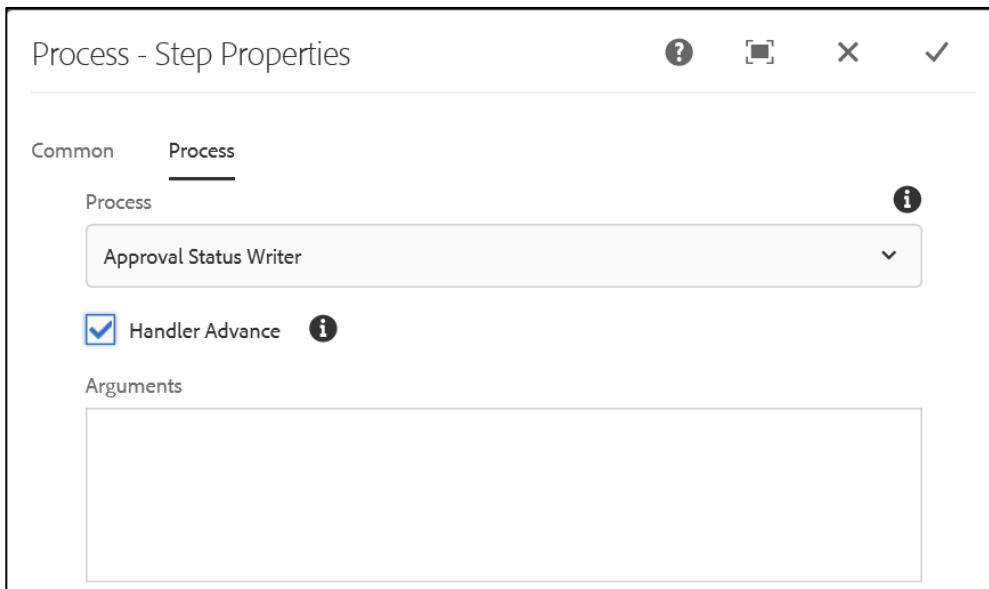
3. Double-click Process Step. The Process – Step Properties dialog box opens.

4. Select the **Common** tab and enter the title:

- Title: **Approval Status Writer**

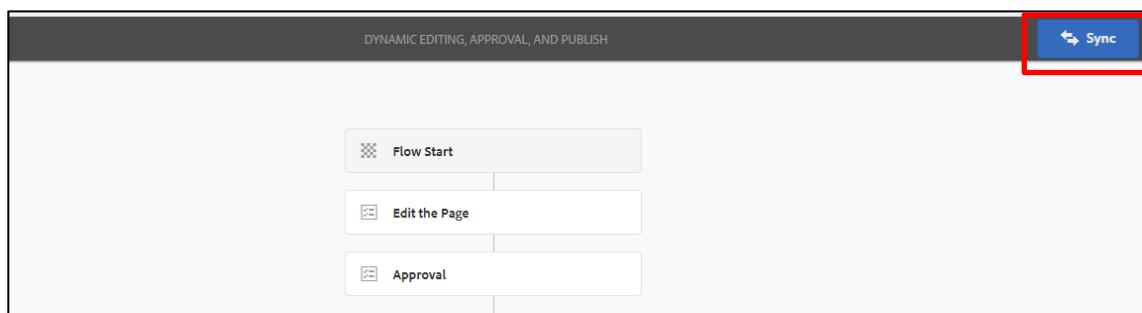
5. Select the **Process** tab and enter the following details, as shown:

- Name: **Approval Status Writer**
- Select the **Handler Advance** checkbox.



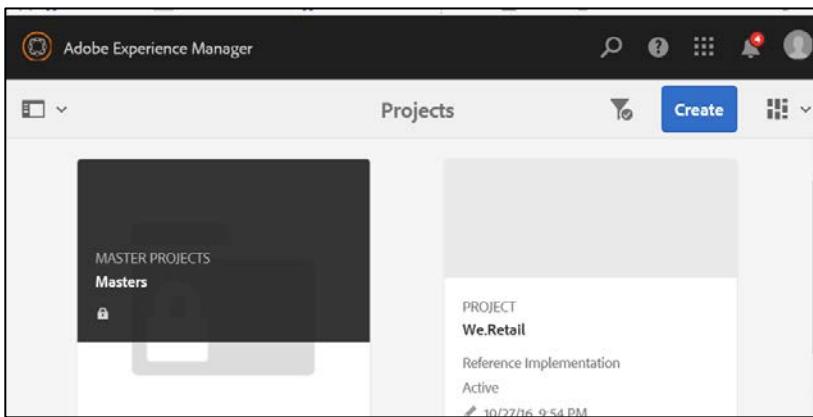
6. Click the Done icon (checkmark) on the top of the dialog box to save the changes.

7. At this point the workflow is complete. In the top-right click **Sync**, as shown. This will write the workflow model.

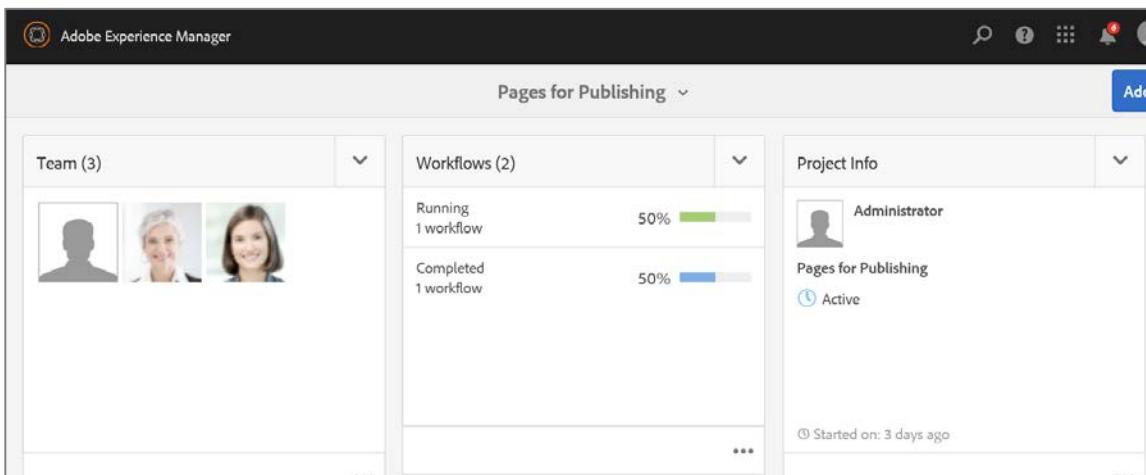


## Task 4: Test the workflow

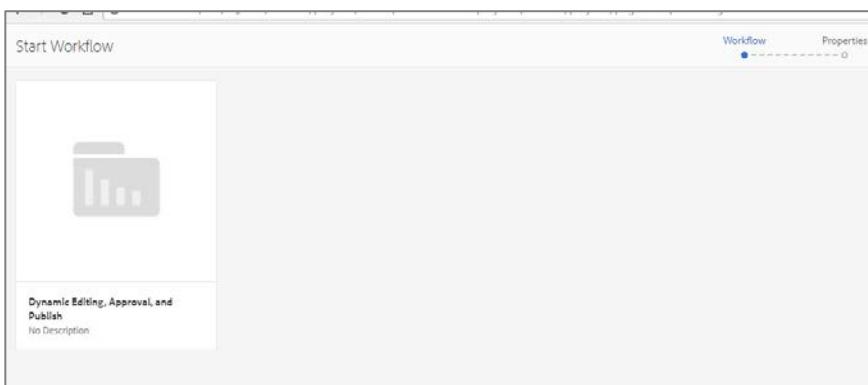
1. To test the new project, navigate to the **AEM > Projects**. The **Projects console** opens, as shown:



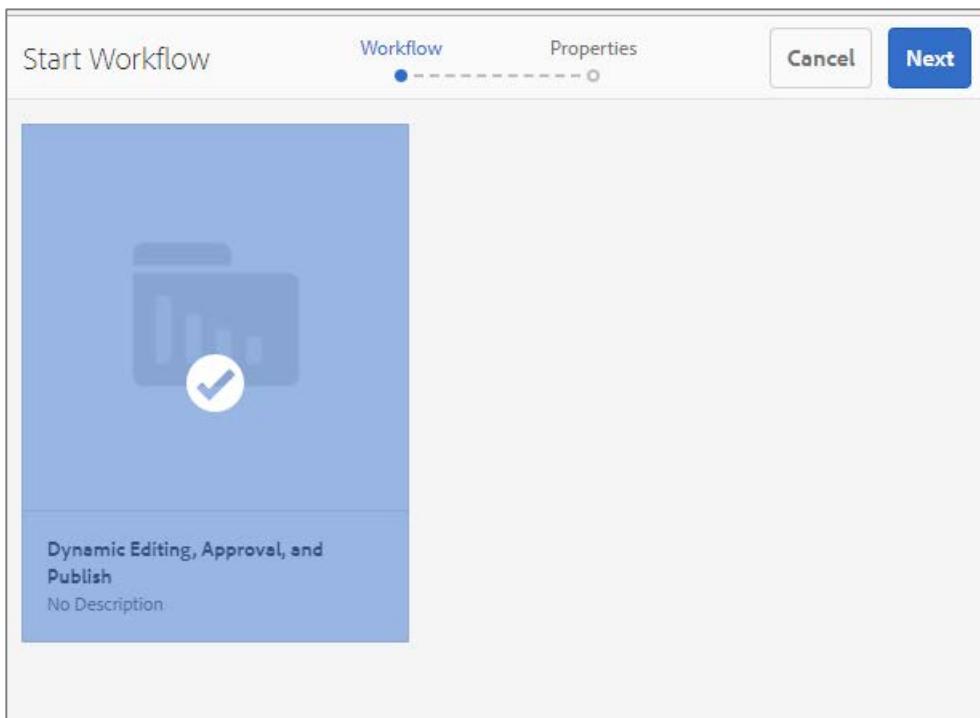
2. Click **Pages for Publishing** Project, and click **Next**. The **Pages for Publishing** window opens, as shown:



3. Click the dropdown on the **Workflow** tile and select **Start Workflow**. The **Start Workflow** page opens, as shown:



4. Select Dynamic Editing, Approval and Publish and click Next, as shown: The Properties window opens.



5. Enter the following details, as shown:

- Title: **Publish the Experience Page**
- Assign To: **Carlene Avery**
- Content Path: **/content/we-retail/language-masters/en/experience**

A screenshot of a "Properties" dialog box. At the top, it says "Properties" and has "Back" and "Submit" buttons. The form contains the following fields:

- Title: "Publish the Experience Page"
- Assign To: "Carlene Avery"
- Description: (empty text area)
- Content Path: "/content/we-retail/language-masters/en/experience"
- Task Priority: "Medium"

6. Click **Submit** at the top right corner. The workflow is built.



**Note:** At this point, you should be able to walk through the edit/approval steps. Even though the tasks are assigned to different project groups, the admin can see and run through all the tasks.

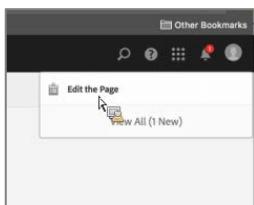
7. Refresh the browser and notice the new **Task**, as shown:

The screenshot shows the Adobe Experience Manager dashboard with the 'Pages for Publishing' project selected. The 'Tasks (1)' section is highlighted with a red box. It contains a circular progress bar with the number '1 Tasks' and '100%' completion. Below the progress bar, there is a status message: 'Started on: a day ago'.

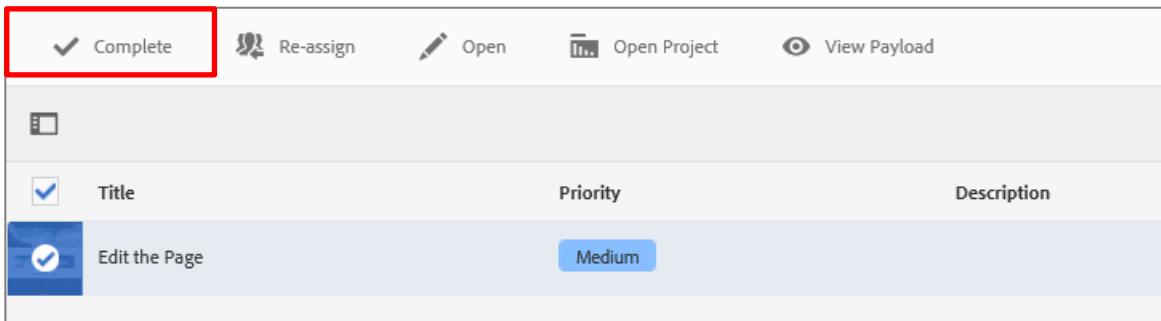
8. Click the ellipsis on the **Task** Tile. The **Inbox** opens, as shown:

Title	Priority	Description	Assignee	Project	Workflow	Status	Start Date	Due Date
Edit the Page	Medium	Make the appropriate edits to the page and complete this task	Carlene Avery	Pages for Publishing	Publish the Experience page	Active	2 minutes ago	

9. Find the new message in the **Inbox**, as shown:

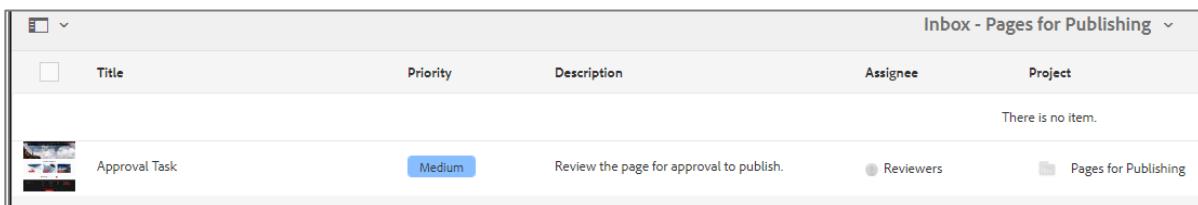


10. Select **Edit the Page**, and click **Complete**, as shown. The **Complete Task** dialog box appears.



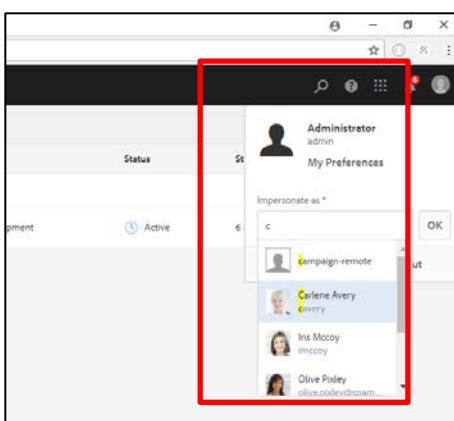
11. Enter a comment and click **Complete**.

12. Notice there is an **Approval Task**, as shown:



13. Revert to the admin user.

14. Impersonate **Carlene Avery**, and click **OK**, as shown:



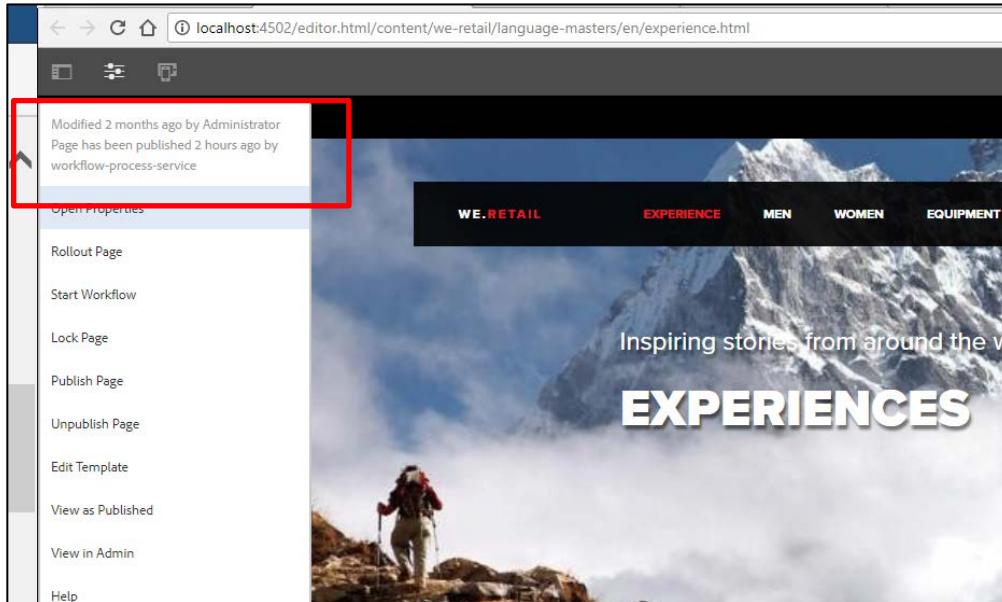
15. Select the **Approval Task**, and click **Complete**. The **Complete Task** pop-up window appears. Notice there is a drop-down with **Approved** and **Denied**.

If you select **Denied**, the workflow will step back and create new task and re-edit the page with the comments of the reviewer (cavery).

If you select **Approved**, the workflow will move forward.

16. Complete the task by selecting **Approved** and add a comment **Approved**. Click **Complete**.

17. Verify the page is published, as shown:



18. In CRXDE Lite, navigate to /content/we-retail/language-masters/en/experience/jcr:content and observe the jcr-content properties, as shown:

Name	Type	Value
approved	Boolean	false
3: cq:lastModifiedBy	String	admin
4: cq:lastReplicated	Date	2018-04-16T12:58:38.686-07:00
5: cq:lastReplicatedBy	String	workflow-process-service
6: cq:lastReplicationAction	String	Activate
7: cq:template	String	/conf/we-retail/settings/wcm/templates/hero-page
8: hideSubItemsInNav	Boolean	true
9: jcr:baseVersion	Reference	<a href="#">Tufe5853-9e76-459d-a733-f40ade43Qe2</a>
10: jcr:created	Date	2018-04-07T19:55:34.145-07:00
11: jcr:createdBy	String	admin
12: jcr:description	String	.
13: jcr:isCheckedOut	Boolean	true
14: jcr:mixinTypes	Name[]	mix:versionable
15: jcr:predecessors	Reference[]	<a href="#">Tufe5853-9e76-459d-a733-f40ade43Qe2</a>
16: jcr:primaryType	Name	cq:PageContent
17: jcr:title	String	Experience



**Note:** Notice a new property named **approved** is added to the list of properties.



## References and Helpful Links:

For more information on AEM Projects, go to:

<https://helpx.adobe.com/experience-manager/kt/platform-repository/using/projects-part1-tutorial-develop.html>

TaskManagement API- <https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/reference-materials/javadoc/com/adobe/granite/taskmanagement/package-summary.html>

Task Object - <https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/reference-materials/javadoc/com/adobe/granite/taskmanagement/Task.html>

Main Granite Workflow API - <https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/reference-materials/javadoc/com/adobe/granite/workflow/package-summary.html>

Granite Workflow Objects - <https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/reference-materials/javadoc/com/adobe/granite/workflow/exec/package-summary.html>

# Managing Users, Groups, and Access Rights in Adobe Experience Manager



## Introduction

Organizations can use Adobe Experience Manager (AEM) to manage projects, workflows, assets, forms, communities and integrations, and build websites and mobile apps. However, they need to ensure that their data, content, documents, and projects are secured from unauthorized internal and external users. In AEM, this is achieved by creating users, groups, and setting-specific access rights to the user account. Setting permissions on users and groups allow you to control how the users and groups can access resources in AEM.

## Objectives

**After completing this course, you will be able to:**

- Explain users, groups, and access control lists
- Create users and groups
- Add users to a group
- Explain permissions and access rights
- Create a custom AEM project to modify permissions

# Working with Users, Groups, and Access Control Lists

You can configure and manage user authentication and authorization within AEM.

## Users and Groups

A user refers to either a human user or an external system connected to the system. The user account stores the user details needed for accessing AEM. Each user account is unique, and stores the basic account details and privileges assigned.

Groups are collections of users and/or other groups. 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 the members of the group.

Enabling access to a CRX repository involves several topics:

- Access Rights
- User Administration
- Group Administration
- Access Right Management

Access Right Management is important, especially with different users accessing and performing different tasks within AEM. With the access control tab in CRXDE Lite, you can configure access control policies and assign their corresponding privileges.

## Permissions and ACLs

AEM uses Access Control Lists (ACLs) to determine what actions a user or a group can take, and where it can perform those actions.

Permissions define who can perform which actions on a resource. The permissions are applied after evaluating ACLs. You can change the permissions granted or denied to a given user by selecting or clearing the checkboxes for the individual AEM actions. A checkmark indicates the user can perform that action. Whereas, no checkmark indicates the user is denied permission to perform that action.

The screenshot depicts the permissions for users and groups:

Properties	Groups	Members	Permissions	Impersonators	Preferences				
		Enter search query <input type="text"/> <input type="button"/>							
Path		Read	Modify	Create	Delete	Read A...	Edit ACL	Replicate	
<input 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"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<a href="#">Details</a>
<input 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"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<a href="#">Details</a>

The location of the checkmark in the grid also indicates what permissions the users have in which locations within AEM.

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, the specific symbols are appended to the checkbox. A local entry is created in the content repository interface.

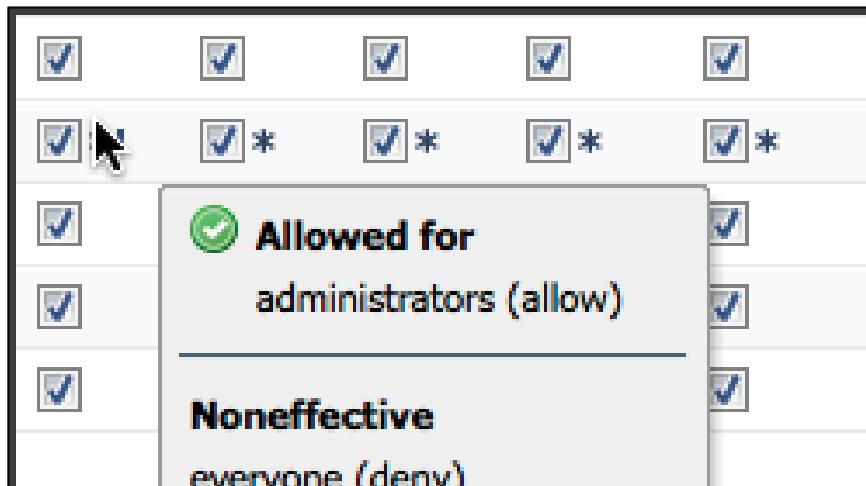
The following table describes the symbols appended to the checkbox for an action at a given path:

* (asterisk)	Indicates that there is at least one local entry (either effective or ineffective). These wildcard ACLs are defined in the content repository.
! (exclamation mark)	Indicates that the permission is inherited, that is, 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 following table describes the two parts of the tool tip:

Upper part	Lists the effective entries.
Lower part	Lists the non-effective entries that may have an effect somewhere else in the tree.

The screenshot shows the allow and disallow permissions for contents:



## Actions

You can perform actions on a page (resource). For each page in the hierarchy, you can specify the action the user can take on that page. Permissions enable you to allow or deny an action. The following table describes the actions:

Action	Description
Read	The user can read the page and all the child pages.
Modify	The user can modify the existing content on the page and on all the child pages. The user can create new paragraphs on the page or 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, such as cq:Page, nt:file, and cq:Asset.
Create	The user can create a new page or child page. If the Modify action 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 the existing paragraphs from the page or any child page. If the Modify action is denied, any subtrees below jcr:content are specifically excluded as removing jcr:content and its child nodes is considered a page modification. This only applies to nodes defining a jcr:content child node.
Read ACL	The user can read the 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 user can replicate content from an author instance to a public instance. This privilege is also applied to any child pages.

## Evaluating ACLs

AEM WCM uses ACLs to organize the permissions being applied to 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 permission at the bottom is applied to the resource.

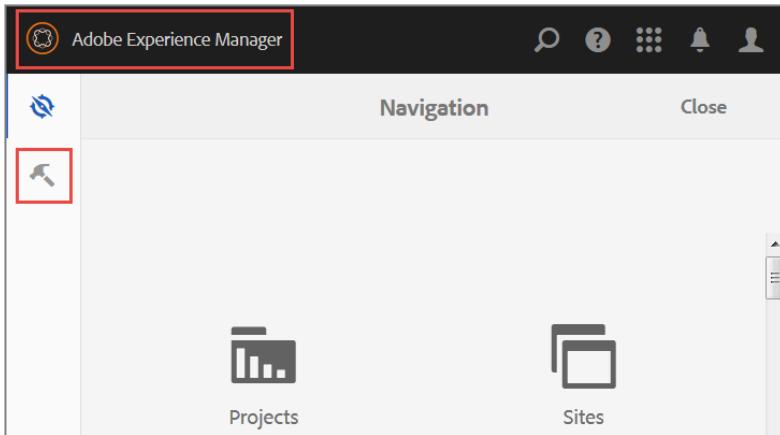
If a user is part of the two groups *allowed-it* and *restricted-it*, you can see how access to the page products is denied because the ACL deny in read access is the rule at the bottom.

If the order of the ACL is the opposite and a user is part of two groups, allowed-it and restricted-it, the user will have access to the products page because the ACL allow in read access rule is at the bottom.

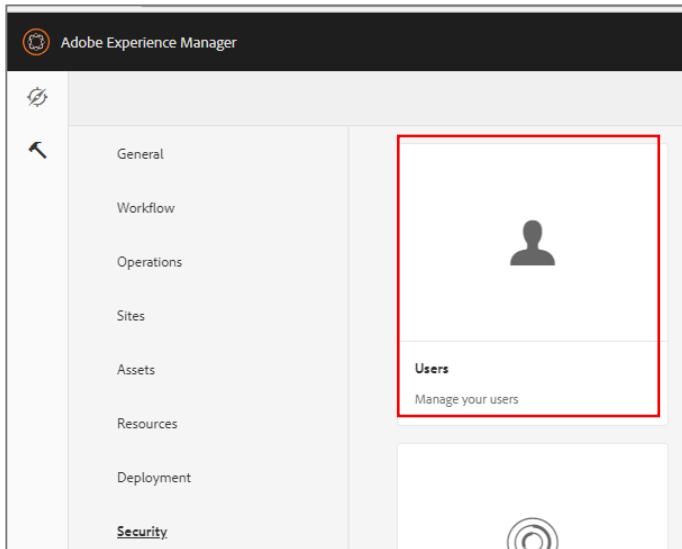
## Exercise 1: Create a new user

In this exercise, you will add yourself as a user in AEM.

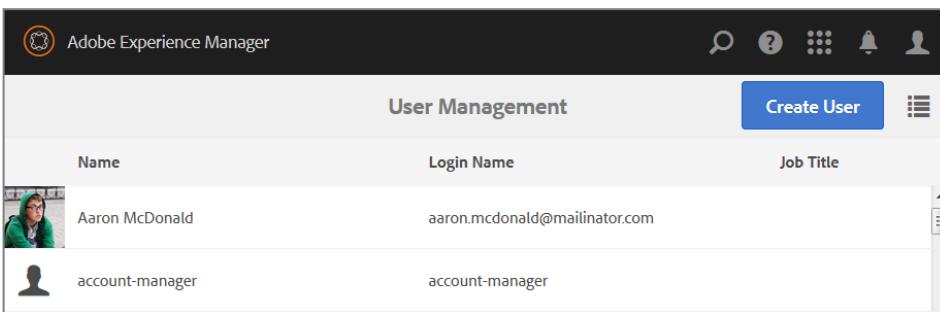
1. Open the **Navigation** console, or click the link: <http://localhost:4502>
2. Click **Adobe Experience Manager** in the upper left of the screen, and then click the Tools icon, as shown:



3. Click **Security > Users**. The **User Management** page is displayed.



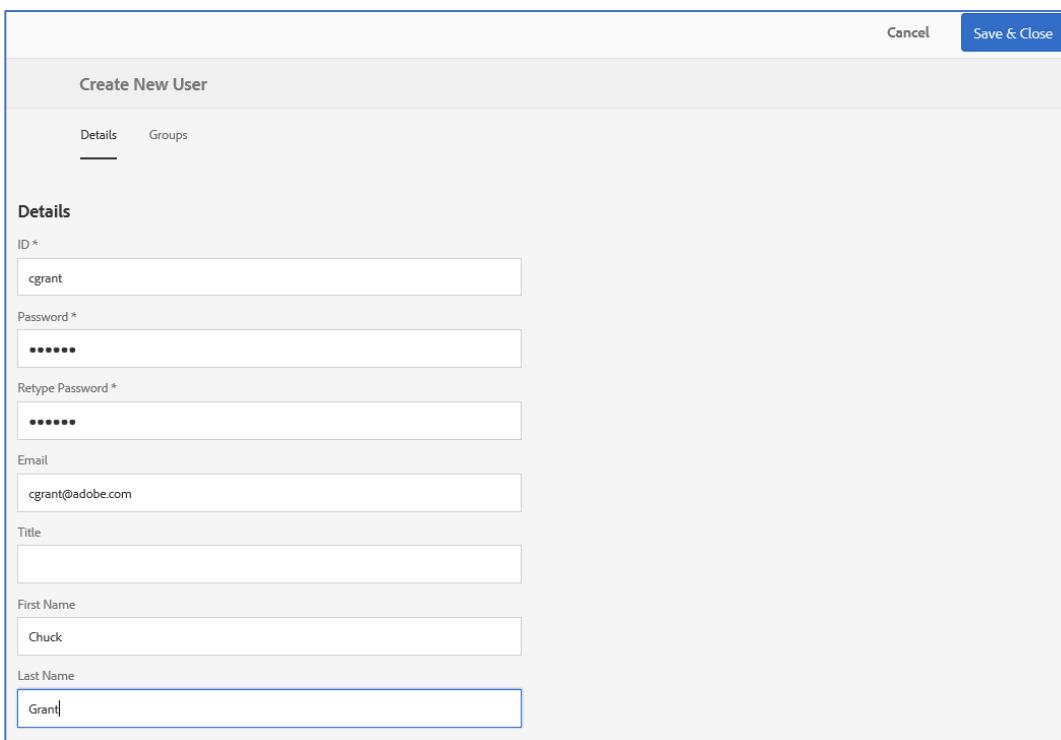
4. Click **Create User** from the actions bar. The **Create New User** wizard is displayed.



The screenshot shows the Adobe Experience Manager User Management interface. At the top, there is a search bar, a help icon, a grid icon, a bell icon, and a user profile icon. Below the header, the title "User Management" is centered, and to its right is a blue "Create User" button. The main area is a table with three columns: "Name", "Login Name", and "Job Title". There are two rows of data:

Name	Login Name	Job Title
Aaron McDonald	aaron.mcdonald@mailinator.com	
account-manager	account-manager	

5. Enter your details such as your ID, email address, password, first name, and last name, as shown:

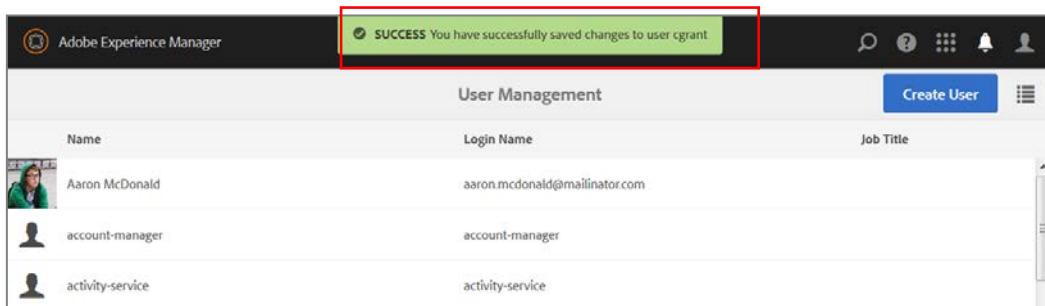


The screenshot shows the "Create New User" wizard. At the top right are "Cancel" and "Save & Close" buttons. The main section is titled "Create New User" and has tabs for "Details" and "Groups", with "Details" selected. The "Details" form contains the following fields:

- ID \*: cgrant
- Password \*: \*\*\*\*\*
- Retype Password \*: \*\*\*\*\*
- Email: cgrant@adobe.com
- Title: (empty)
- First Name: Chuck
- Last Name: Grant

6. Click Save & Close.

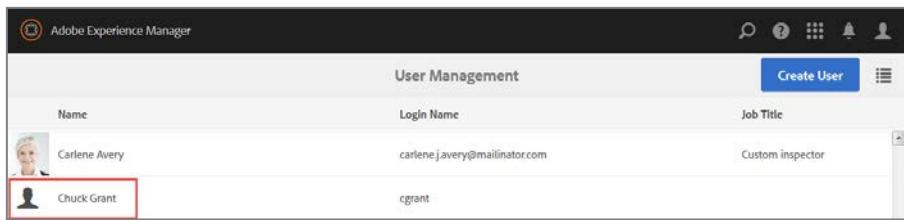
The success message displays on the **User Management** page, as shown:



The screenshot shows the User Management page again. A green success message box is displayed at the top center, containing the text "SUCCESS You have successfully saved changes to user cgrant". The rest of the page is identical to the previous screenshot, showing the list of users.

+

The user you created is added to the User Management list, as shown:



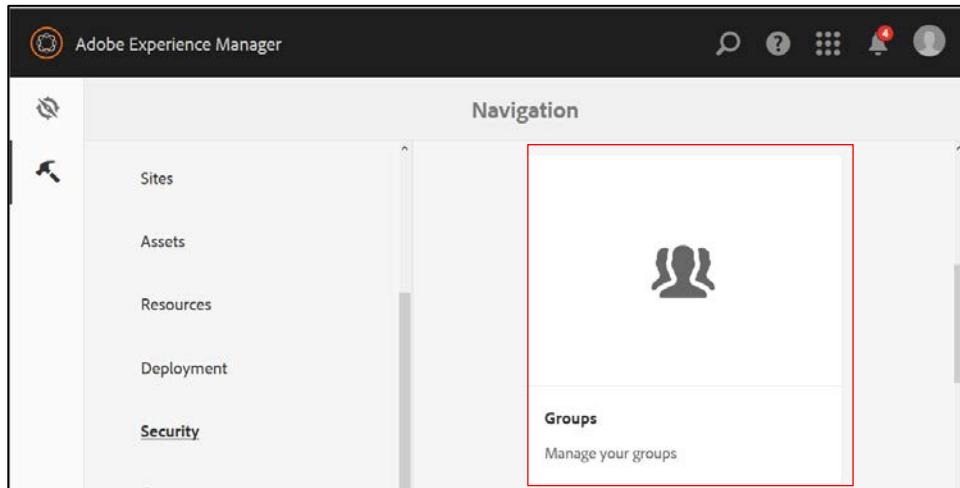
A screenshot of the Adobe Experience Manager User Management interface. The title bar says "User Management". There are three columns: "Name", "Login Name", and "Job Title". Two users are listed: "Carlene Avery" (Login Name: carlene.javery@mailinator.com, Job Title: Custom inspector) and "Chuck Grant" (Login Name: cgrant). The row for "Chuck Grant" has a red border around the entire row.

Name	Login Name	Job Title
Carlene Avery	carlene.javery@mailinator.com	Custom inspector
Chuck Grant	cgrant	

## Exercise 2: Create a new group

In this exercise, you will create a new group named Legal.

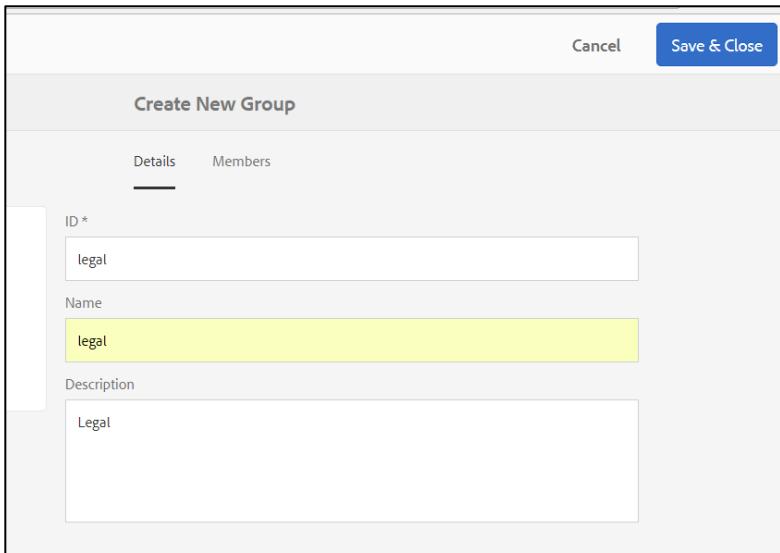
1. Navigate to the **Projects console**, or click the link: <http://localhost:4502>
2. Click **Adobe Experience Manager** at the top left corner of the screen, and then click the Tools icon.
3. Click **Security > Groups**. The **Group Management** page is displayed.



4. Click **Create Group** from the actions bar. The **Create New Group** wizard is displayed.

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

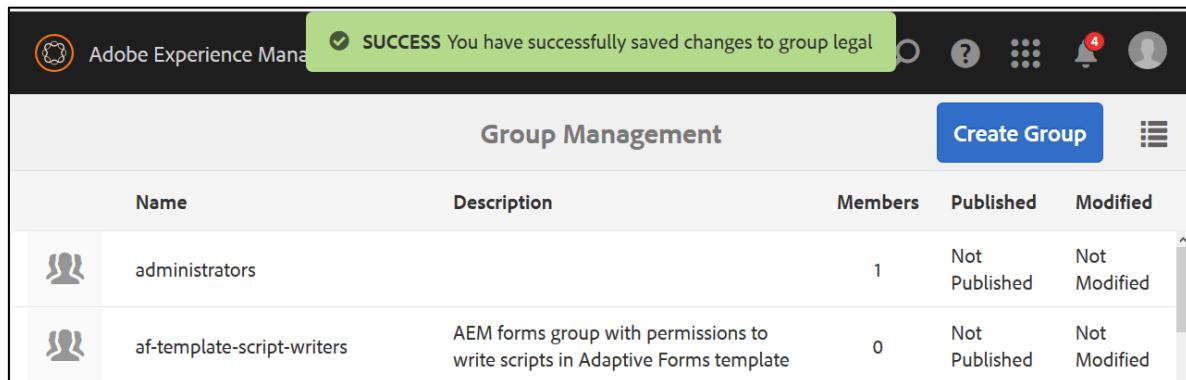
5. Enter the group details such as id, name, and description, as shown:



The screenshot shows the 'Create New Group' dialog box. At the top right are 'Cancel' and 'Save & Close' buttons. Below the title 'Create New Group' are two tabs: 'Details' (which is selected) and 'Members'. The 'Details' section contains three fields: 'ID \*' with the value 'legal', 'Name' with the value 'legal' (which is highlighted with a yellow background), and 'Description' with the value 'Legal'.

6. Click **Save & Close**. The group is created.

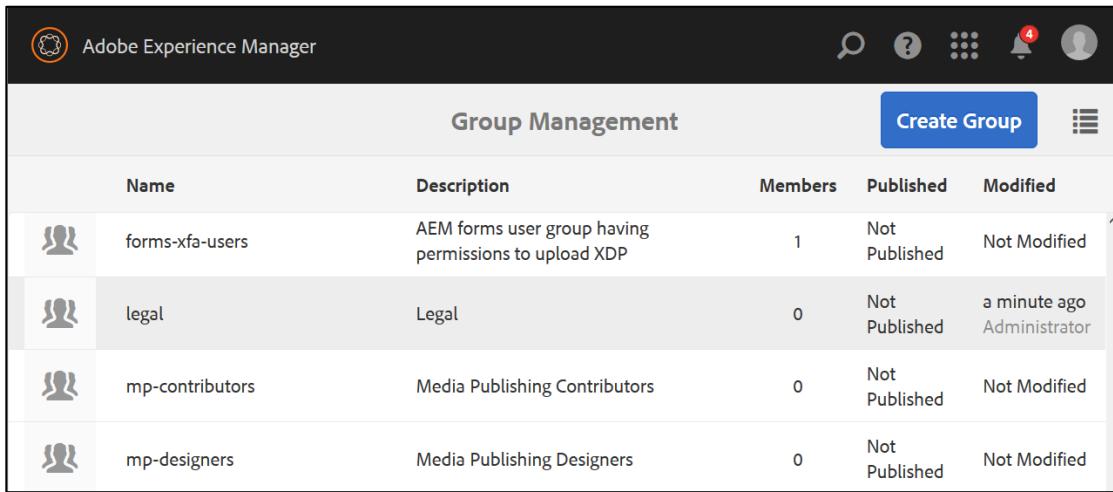
The success message is displayed on the **Group Management** page, as displayed below:



The screenshot shows the 'Group Management' page in Adobe Experience Manager. At the top, there is a success message: 'SUCCESS You have successfully saved changes to group legal'. The main area is a table titled 'Group Management' with a 'Create Group' button at the top right. The table has columns: Name, Description, Members, Published, and Modified. It lists two groups: 'administrators' (1 member, Not Published, Not Modified) and 'af-template-script-writers' (0 members, Not Published, Not Modified).

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, as shown:



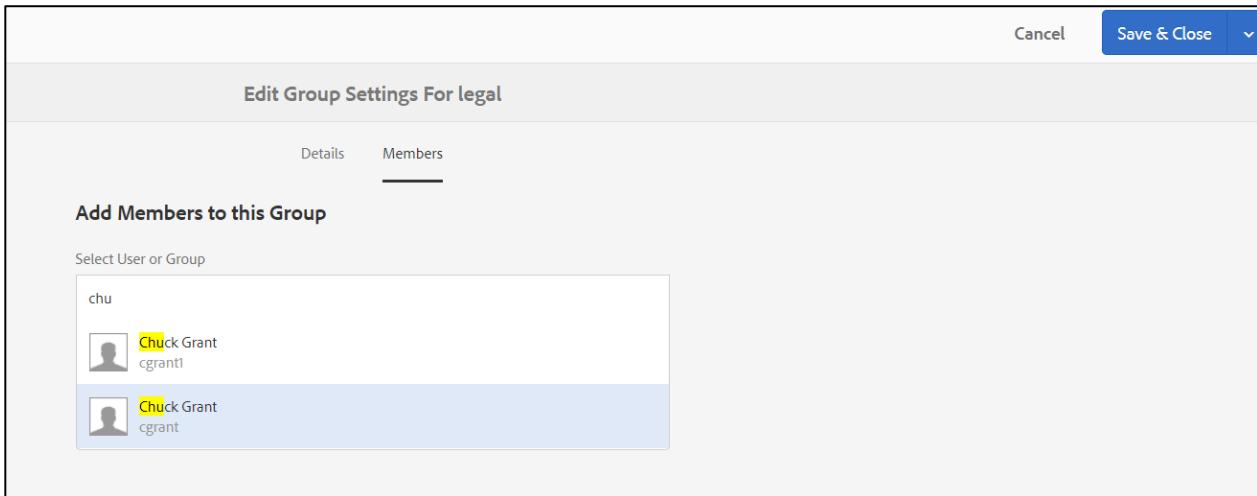
The screenshot shows the 'Group Management' page in Adobe Experience Manager. At the top, there is a header with the AEM logo and navigation icons. Below the header is a table with the following columns: Name, Description, Members, Published, and Modified. The table contains four rows of data:

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 the user to the group

In this exercise, you will add yourself to the **Legal** group.

1. Navigate to the **Projects** 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**. The **Edit Group Settings** wizard appears.
5. Select the **Members** tab. The **Add Members** to this Group appears.
6. Select **cgrant** from the drop-down list, as shown:



7. Click **Save & Close**. The success message is displayed on the **Group Management** page
8. Look for the **Contributors** group from the list, and then click the **Contributors**. **Edit Group Settings** wizard appears.
9. In the **Add Members to Group** section, select the **Legal** group and click **Save & Close**.
10. Click **Adobe Experience Manager** on the upper left corner of the screen, and then click the Tools icon.
11. Click **Security > Permissions**. The Security window opens.
12. 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.

13. From the right pane, click the **Groups** tab. You will notice the user is given access to the **Legal** and **Contributors** groups.
14. 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 checked boxes for most items, indicating the 'read' permission is selected. Other columns include 'Modify', 'Create', 'Delete', 'Read A...', 'Edit ACL', and 'Replicate'.

Path	Read	Modify	Create	Delete	Read A...	Edit ACL	Replicate
/libs	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/assetusagetrackeruser	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/audit-service	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/authentication-service	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/campaign-cloudservice	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/campaign-reader	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/campaign-remote	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/canvapage-activate-service	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/canvapage-delete-service	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/cavery	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/cdn-service	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/cgrant	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/clientlibs-service	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/commerce-backend-service	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/commerce-frontend-service	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/commerce-orders-service	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/communities-adm-manager	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/communities-analytics-admin	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/we-retail	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/we-retail/bin	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/we-retail/we-retail-client-app	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/we-retail/we-retail-commu...	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/we-retail/we-retail-screens	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/we-unlimited-app	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
/libs/assets/weretail	<input checked="" type="checkbox"/>	<input type="checkbox"/>					

## Exercise 4: Create a custom AEM project with ACL automation

In this exercise, you will automate permission customization for a custom AEM project and workflow. The JCR API is a low-level API that allows you to customize the repository and create nodes and properties. Most operations in AEM can be coded by Sling or AEM APIs more effectively rather than using the JCR API. You will use the JCR API to programmatically change the permissions for a user within a workflow in this exercise. This exercise has the following four tasks:

1. Create a custom workflow process
2. Add the workflow process to a project workflow
3. Increase permissions on the workflow process user
4. Test the permissions automation with an AEM project

**Use Case:** 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 is 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 their website and write permissions are given only on request. Authors should only have access to the pages they are currently assigned.

**Solution:** Create a custom project for each PM that allows them to add team members for editing and reviewing. When new work needs to be assigned:

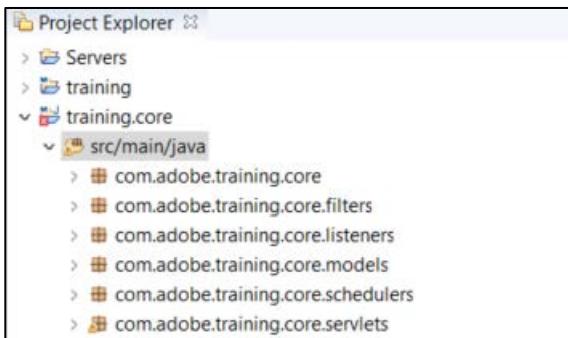
- i. The PM will start a project workflow instance with the work (or payload) and complete details of the work to be done. The PM will also specify the editor that is assigned to the work.
- ii. The workflow will then assign edit permissions to the editor and notify the editor with a task that contains the details of the work to be done.
- iii. After 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.

### Task 1: Create a custom workflow process

1. Launch **Eclipse** by double-clicking the **Eclipse** shortcut on the desktop. The Eclipse Launcher opens.
2. Specify the Workspace as **C:\Workspace** and click **Launch**. The Workspace opens.



3. In Project Explorer, navigate to **training.core > src/main/java**, as shown:

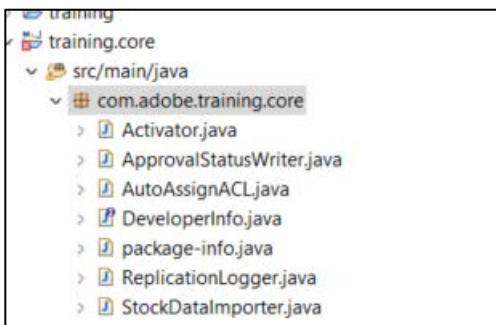


4. Copy the file **AutoAssignACL.java** from **/Exercise\_Files/13\_Automate\_ACL\_with\_JCR\_API/**.



**Note:** The code is provided as part of the Exercise\_Files under **/Exercise\_Files/13\_Automate\_ACL\_with\_JCR\_API/**. Copy and paste the file from the exercise files referenced to **AutoAssignACL.java** to Eclipse. Please do not copy the code from this exercise book. The code in the student guide is for illustrative purposes only.

5. In Eclipse, right-click **com.adobe.training.core** and paste the file. The **AutoAssignACL.java** class is created, as shown:



6. Double-click **AutoAssignACL.java**. The file opens in editor.

```

1. package com.adobe.training.core;
2.
3. import javax.jcr.security.AccessControlList;
4. import javax.jcr.security.AccessControlManager;
5. import javax.jcr.security.Privilege;
6. import javax.jcr.Session;
7.
8. import org.osgi.service.component.annotations.Component;
9.
10. import org.apache.jackrabbit.api.security.user.Authorizable;
11. import org.apache.jackrabbit.api.security.user.UserManager;
12. import org.apache.jackrabbit.commons.jackrabbit.authorization.AccessControlUtils;

```

```

13. import org.apache.sling.jcr.base.util.AccessControlUtil;
14. import org.osgi.framework.Constants;
15. import org.slf4j.Logger;
16. import org.slf4j.LoggerFactory;
17.
18. import com.adobe.granite.workflow.WorkflowException;
19. import com.adobe.granite.workflow.WorkflowSession;
20. import com.adobe.granite.workflow.exec.WorkItem;
21. import com.adobe.granite.workflow.exec.WorkflowProcess;
22. import com.adobe.granite.workflow.metadata.MetaDataMap;
23. import com.adobe.granite.workflow.exec.WorkflowData;
24.
25. /**
26. * This is a project workflow process step. When starting a workflow from a project, you can
27. specify the:
28. *
29. * Payload: The page that will have 'modify' permissions added/removed from its ACL
30. * User: The user attached to the ACL
31. *
32. * This process will add/remove jcr privileges that correlate to 'Modify, Create, Delete' in
33. * the /useradmin
34. * for the given user on the payload. An argument is used to specify to add or remove 'modify'
35. * permissions.
36. * The argument must follow permission=<add || remove>
37. */
38. @Component(service = WorkflowProcess.class,
39.             property = {Constants.SERVICE_DESCRIPTION + "=A sample workflow process implementa-
40.                         tion.",
41.                         Constants.SERVICE_VENDOR + "=Adobe Digital Learning Services",
42.                         "process.label=Auto Assign Edit Permissions"
43.                     })
44.
45. public class AutoAssignACL implements WorkflowProcess{
46.
47.     private final Logger logger = LoggerFactory.getLogger(getClass());
48.
49.     /**
50.      * @param item - Holds the payload and user
51.      * @param workflowSession - Session with service user (workflow-process-
52.      * service) that will be modifying the permissions
53.      * @param workflowArgs - permission=add will add 'modify' permissions. permission=remove
54.      * will remove 'modify' permissions
55.      */
56.     @Override
57.     public void execute(WorkItem item, WorkflowSession workflowSession, MetaDataMap workflowA-
58.     rgs) throws WorkflowException {
59.
60.         WorkflowData workflowData = item.getWorkflowData(); //get the workflow properties
61.         String userID = workflowData.getMetaDataMap().get("assignee", String.class);
62.         logger.info("User to add permissions: " + userID);
63.         String payload = workflowData.getPayload().toString();
64.         logger.info("Content path to add permissions: " + payload);
65.
66.         UserManager uM;

```

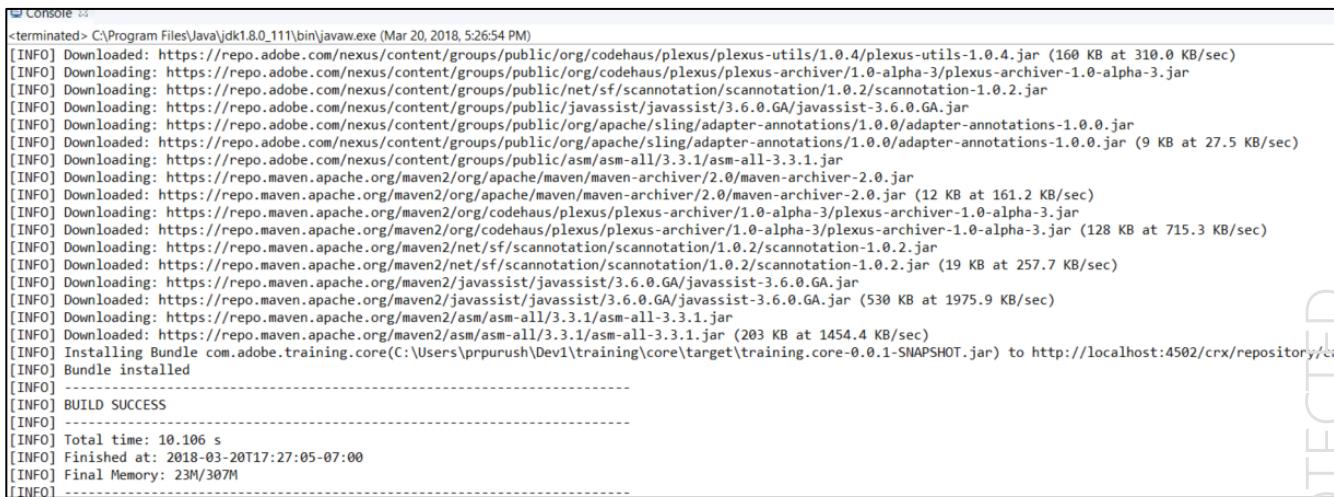
```

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.getAccessControlManager();
71.
72.          //JCR privileges that encompass AEM Permissions: "Modify, Create, Delete" in the
   /useradmin
73.          Privilege[] privileges = {accessControlManager.privilegeFromName(Privilege.JCR_MO-
   DIFY_PROPERTIES),
74.                               accessControlManager.privilegeFromName(Privilege.JCR_LOCK_MANAGEMENT),
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_CHILD_NODES),
79.                               accessControlManager.privilegeFromName(Privilege.JCR_NODE_TYPE_MANAGEMENT
   )};
80.
81.          //Get the ACL for the payload
82.          AccessControlList acl = AccessControlUtils.getAccessControlList(jcrSession, payLo-
   ad);
83.          //Add the user/privileges to the payload ACL
84.          acl.addAccessControlEntry(authorizable.getPrincipal(), privileges);
85.
86.          //Based on the process step arguments, permissions are either added or removed
87.          //Assumes arguments are given as: permission=add
88.          String arguments = workflowArgs.get("PROCESS_ARGS", "");
89.          if(arguments.contains("permission")){
90.              String permission = arguments.split("=")[1];
91.              if(permission.equals("add")){
92.                  logger.info("Adding permissions");
93.                  accessControlManager.setPolicy(payload, acl);
94.                  logger.info("Added modify permissions to " + payload + " for " + userID);
95.
96.          } else if(permission.equals("remove")){
97.              logger.info("Removing permissions");
98.              accessControlManager.removePolicy(payload, acl);
99.              logger.info("Removed modify permissions to " + payload + " for " + userID
   );
100.         }
101.         } else{
102.             //do nothing, if the workitem argument is not set
103.             logger.info("No arguments given for workitem");
104.         }
105.         jcrSession.save();
106.     } catch (Exception e) {
107.         logger.error(e.getMessage(), e);
108.         e.printStackTrace();
109.     }
110. }
111. }

```

7. In Project Explorer, right-click **training.core** and choose **Run As > Maven install**. The build starts.

8. Verify the bundle installed successfully, as shown:



```
Console >
<terminated> C:\Program Files\Java\jdk1.8.0_111\bin\javaw.exe (Mar 20, 2018, 5:26:54 PM)
[INFO] Downloaded: https://repo.adobe.com/nexus/content/groups/public/org/codehaus/plexus/plexus-utils/1.0.4/plexus-utils-1.0.4.jar (160 KB at 310.0 KB/sec)
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar
[INFO] Downloaded: https://repo.adobe.com/nexus/content/groups/public/org/apache/sling/adapter-annotations/1.0.0/adapter-annotations-1.0.0.jar (9 KB at 27.5 KB/sec)
[INFO] Downloading: https://repo.adobe.com/nexus/content/groups/public/org/asm/asm-all/3.3.1/asm-all-3.3.1.jar
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.0/maven-archiver-2.0.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/2.0/maven-archiver-2.0.jar (12 KB at 161.2 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/1.0-alpha-3/plexus-archiver-1.0-alpha-3.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar (128 KB at 715.3 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/net/sf/scannotation/scannotation/1.0.2/scannotation-1.0.2.jar (19 KB at 257.7 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/javassist/javassist/3.6.0.GA/javassist-3.6.0.GA.jar (530 KB at 1975.9 KB/sec)
[INFO] Downloading: https://repo.maven.apache.org/maven2/asm/asm-all/3.3.1/asm-all-3.3.1.jar
[INFO] Downloaded: https://repo.maven.apache.org/maven2/asm/asm-all/3.3.1/asm-all-3.3.1.jar (203 KB at 1454.4 KB/sec)
[INFO] Installing Bundle com.adobe.training.core(C:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar) to http://localhost:4502/crx/repository/crxweb
[INFO] Bundle installed
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 10.106 s
[INFO] Finished at: 2018-03-20T17:27:05-07:00
[INFO] Final Memory: 23M/307M
[INFO] -----
```



## Task 2: Add the workflow process to a project workflow

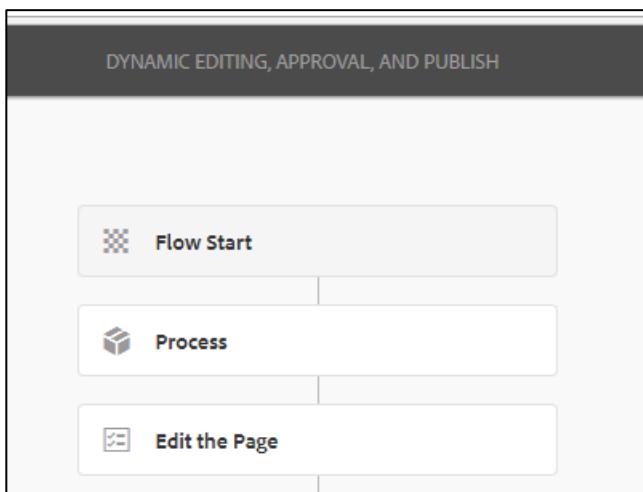
1. In a previous module (Deep\_Dive\_into\_AEM\_APIs) , you created a custom project and custom workflow. If you do not have these exercises completed, install:

- training-project-template.zip
- training-project-workflow.zip



**Note:** Both of these files can be found in the exercises\_folder.

2. Open the workflow **Dynamic Editing, Approval and Publish** by selecting the new workflow and clicking **Edit**. The workflow opens.
3. Drag and drop **Process Step** from the Workflow section to the area with the dotted line **Drop steps or participants here**, as shown:



4. Double-click to open the **Process Step**.

5. Select the **common** tab:

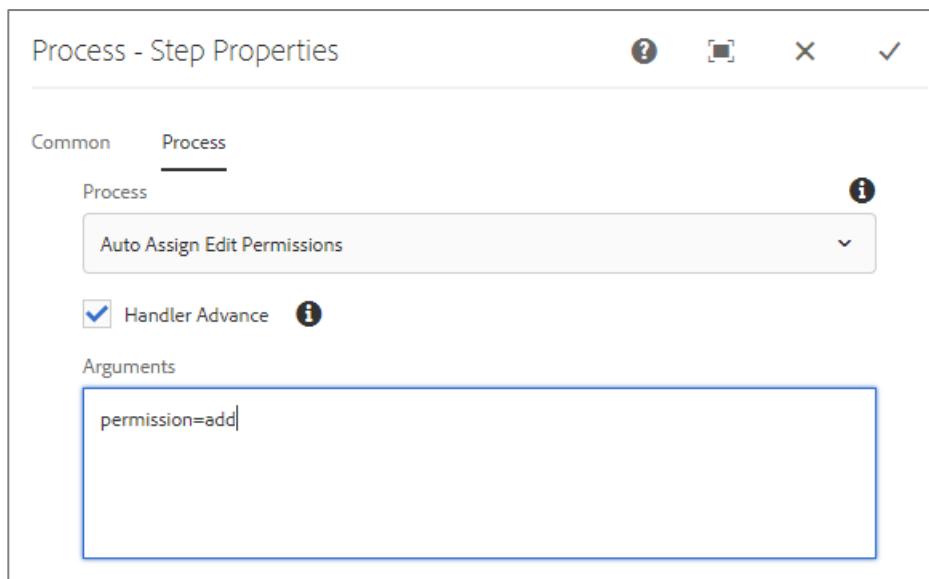
Title: **Add Edit Permissions**

6. Select the **Process** tab:

Process: **Auto Assign Edit Permissions** (created when you added the AutoAssignACL.java class)

Select the **Handler Advance** checkbox.

Arguments: **permission=add**



7. Click the Done icon (checkMark) to save the changes.

8. Add another **Process step** from the Workflow section below the **Approval Status Writer**.

- Title: **Remove Edit Permissions**

9. Select the **Process** tab:

Process: Auto Assign Edit Permissions

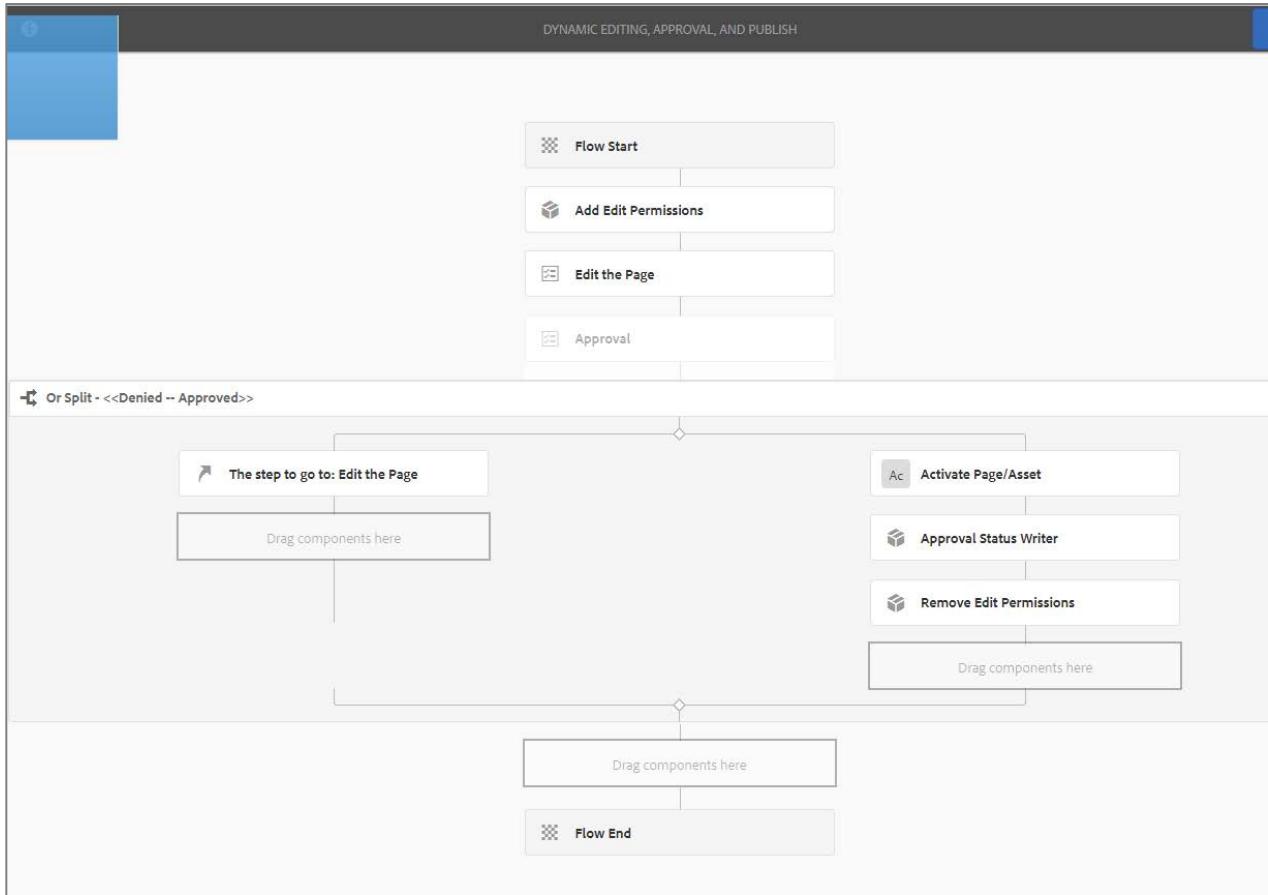
Select the **Handler Advance** checkbox.

Arguments: **permission=remove**

10. Click **CheckMark** to save the changes.

11. Click **Sync** to save the workflow.

12. Verify your completed workflow looks similar to the picture, as shown:



## Task 3: Increase permissions on the workflow process user

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, you need to give workflow-process-service the Read and Edit ACL permissions.

1. Open URL <http://localhost:4502/useradmin>. The **AEM Security** console opens.
2. In the **AEM Security** console, search for the user named **workflow-process-service**, as shown:

The screenshot shows the AEM Security interface. At the top, there's a green header bar with the AEM logo and the text 'AEM | Security'. Below it is a search bar with the word 'workflow'. Underneath is a table with columns: T... (Type), ID, Name, Pub., and Mod. The table lists several users: 'wcm-workflow-service' (ID: wcm-workflow-s...), 'workflow-administrators' (ID: workflow-admini...), 'workflow-editors' (ID: workflow-editors), 'workflow-process-service' (ID: workflow-proces...), 'workflow-service' (ID: workflow-service), and 'workflow-users' (ID: workflow-users). The row for 'workflow-process-service' has a red box around its ID column.

3. Select the **Permissions** tab for the workflow-process-service user.
4. Check **Read ACL** and **Edit ACL**, as shown:

The screenshot shows the 'Permissions' tab for the 'workflow-process-service' user. The tab has tabs for Properties, Groups, Members, Permissions (which is selected), Impersonators, and Preferences. Below is a tree view of paths: /, apps, bin, conf, content, etc, home, libs, oak:index. To the right is a grid of permissions for each path. The 'Edit ACL' column is highlighted with a red box. For the root path (/), both 'Read A...' and 'Edit ACL' are checked. For other paths like 'apps', 'content', and 'etc', 'Edit ACL' is checked but 'Read A...' is not.

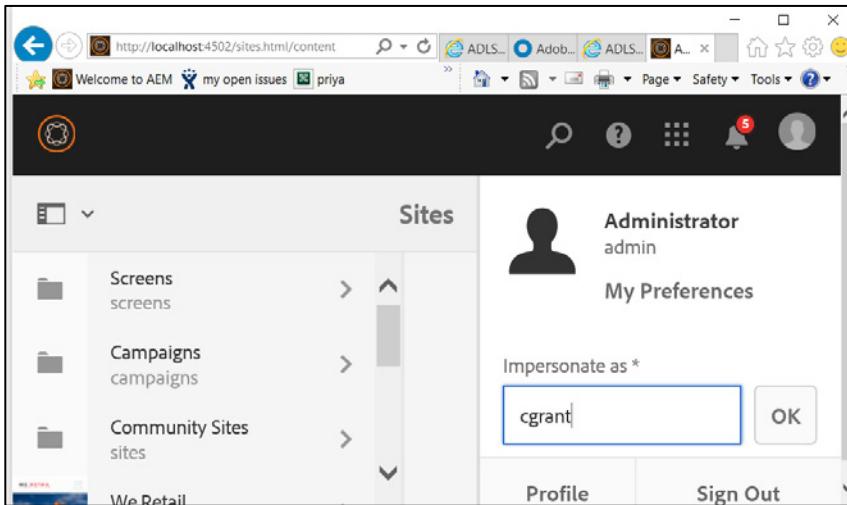
5. Save the changes.

## Task 4: Test the permissions automation with an AEM project

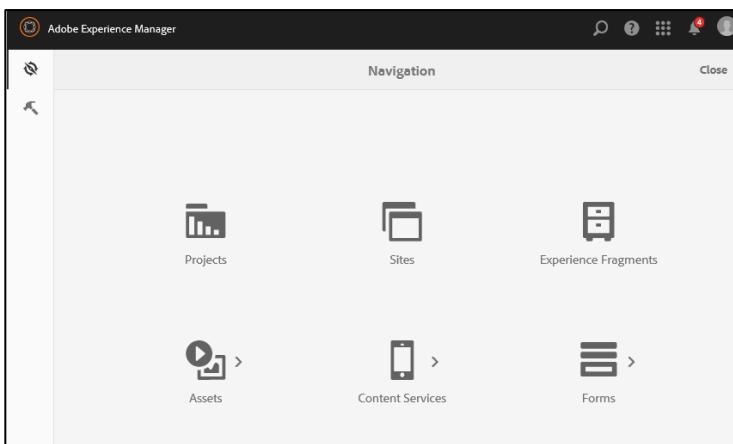
You now have your business process created that was outlined by your Project Manager. You have an AEM Project template that uses our custom workflow that implements our auto assign ACL java class. In this last step, you will run through the AEM Project as the Project Manager.

1. Impersonate Chuck Grant and open

<http://localhost:4502/editor.html/content/we-retail/language-masters/en/men.html>, as shown:



2. Observe the user Chuck Grant has only read permissions to the **We.Retail** Site.
3. Revert back to the **admin** user.
4. Navigate to Projects area - <http://localhost:4502/projects.html>, as shown:



5. Select **Create > Project**. The **Create Project** console opens.
6. Choose **Training Publishing Project** and click **Next**. The Project opens.
7. Enter the values for the project:
  - **Title: Page Editing and Approval**

- Start Date: <Specify a date>
- Due Date: <Specify a date>
- User: **Chuck Grant** (Select Editors from drop-down list)
- Add another user: (by clicking **Add**)
- User: **Carlene Avery** (Select Reviewers from drop-down list)

Template Properties

Basic Advanced

Title \*

Page Editing and Approval

Description

Start Date: 2018-04-19 18:28

Due Date: 2018-04-19 18:28

User: Administrator (Owners), Chuck Grant (Editors)

Members: Administrator, Chuck Grant, Carlene Avery (Reviewers)

8. Click **Create**. The Success pop-up window opens.
9. In the Success pop-up window, click **Open**. The Project opens, as shown:

Adobe Experience Manager

Page Editing and Approval

Team (3)

Workflows (0)

Project Info

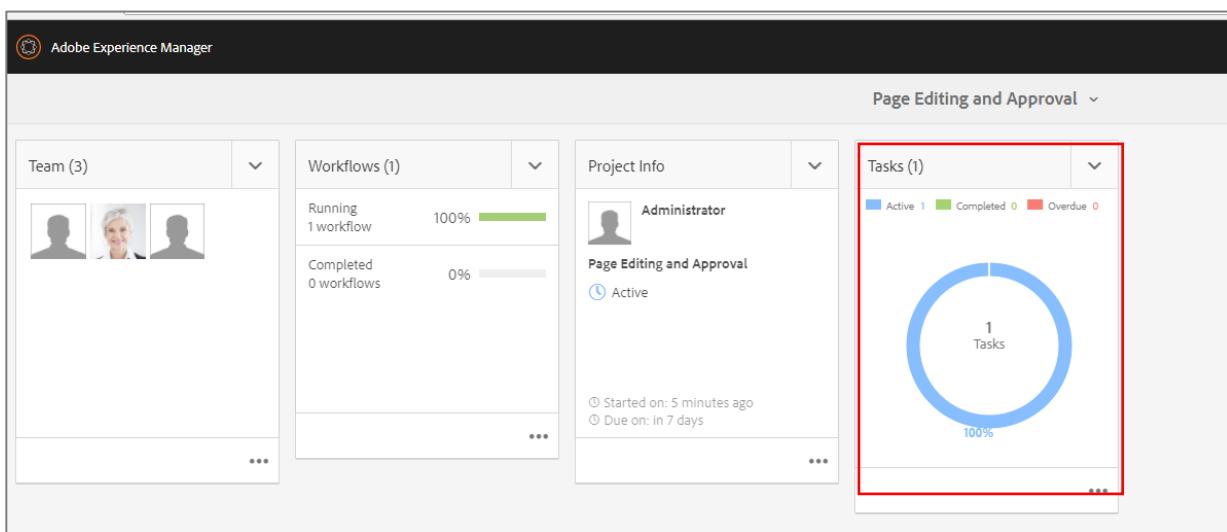
Administrator  
Page Editing and Approval  
Active  
Started on: 3 minutes ago  
Due on: in 7 days



10. In the **Workflows** Tile, Click **Add Work**. The **Start Workflow** screen opens.
11. In the **Start Workflow** screen, Select **Dynamic Editing, Approval, and Publish** and click **Next**. The **Properties** screen opens.
12. In the **Properties** screen, enter the values, as shown:
  - Title: **Edit the Mens Page**
  - Assign To: **Chuck Grant**
  - Content Path: **/content/we-retail/language-masters/en/men**

The screenshot shows the 'Properties' screen in Adobe Experience Manager. It includes fields for Title, Assign To, Description, Content Path, Task Priority, and Due Date. The 'Content Path' field is highlighted with a blue border. At the top right are 'Back' and 'Submit' buttons.

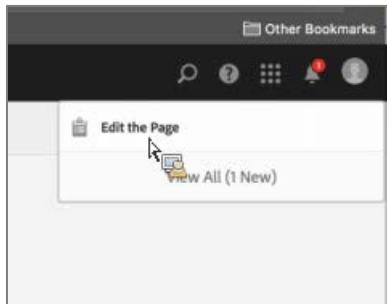
13. Click **Submit**. The Success pop-up window opens.
14. Click **open**. The project opens.
15. Refresh the browser and notice the new task, as shown:



+

16. Open another browser and log in as **Chuck user**.

17. Find the new message in the inbox, as shown:



18. Click **View All (1 New)**. The message is displayed.

19. Select **Edit the Page** and click **View Payload**, as shown:

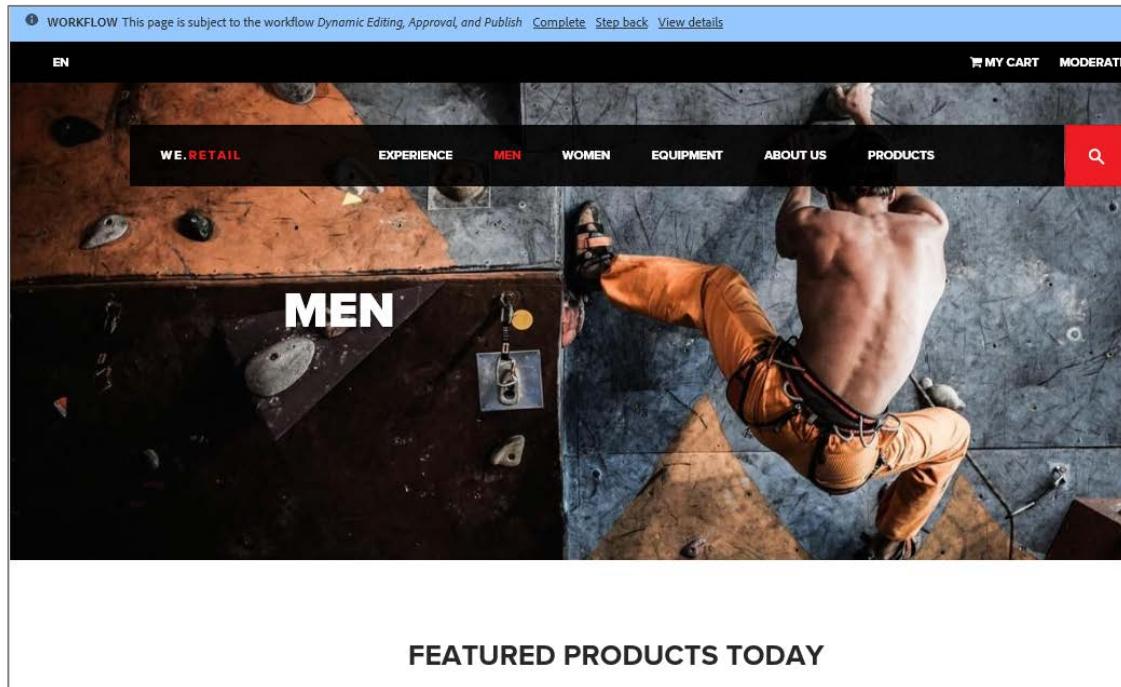
A screenshot of a task management interface. At the top, there are several buttons: "Complete", "Re-assign", "Open", "Open Project", and "View Payload" (which is highlighted with a red box). Below this, there is a table with columns for "Title", "Priority", and "Description". A single row is selected, showing "Edit the Page" in the Title column, "Medium" in the Priority column, and a blue "Edit" button in the Description column.

20. Notice the page is in the workflow and you have the option to edit the page:

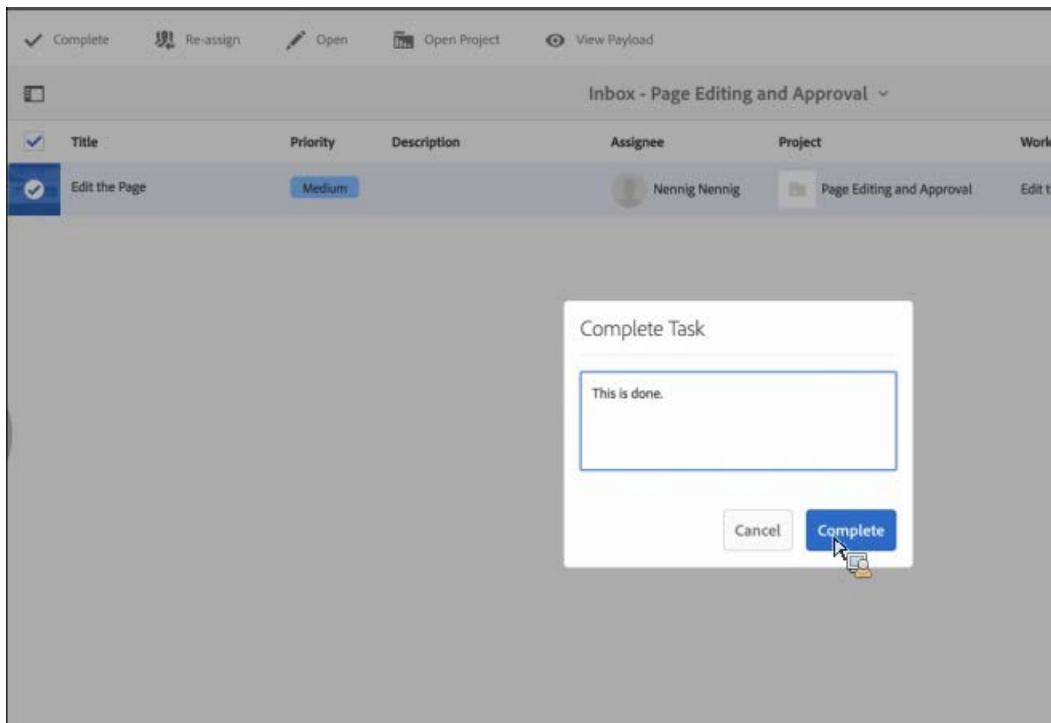
A screenshot of the Adobe Experience Manager (AEM) authoring interface. The page content includes a banner for "WE.RETAIL" and a "MEN" section featuring a shirtless man climbing. A modal window titled "Edit" is open over the page, showing options like "Layout", "Timewarp", and "Targeting". At the bottom of the page, there is a "FEATURED PRODUCTS" component.

21. Click on the Title component called **Featured Products**. Notice all Edit actions are now enabled.

22. Click on the Wrench icon.
23. Make changes to the **Title** and click **Done**. The changes are saved.
24. Verify the page has updated with the changed you made.



25. Go back to the Inbox, select **Edit the Page**, and click **Complete** to complete the task.



26. Enter a comment and **Complete**. The task is completed.

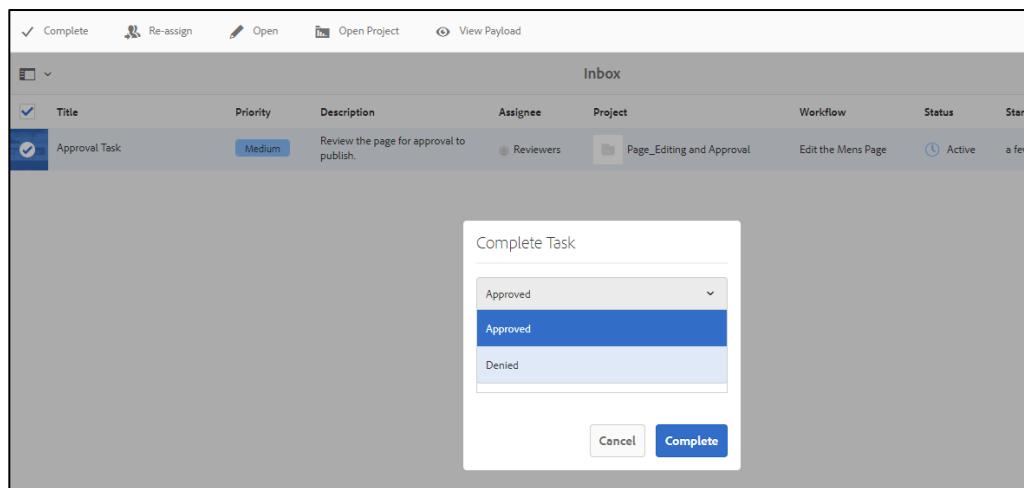
27. Revert to the admin user.

28. Impersonate Carlene Avary.

29. Notice there is a new message in her inbox.

30. Click **View All (1 New)**. The message is displayed.

31. Notice there is an Approval Task, as shown:



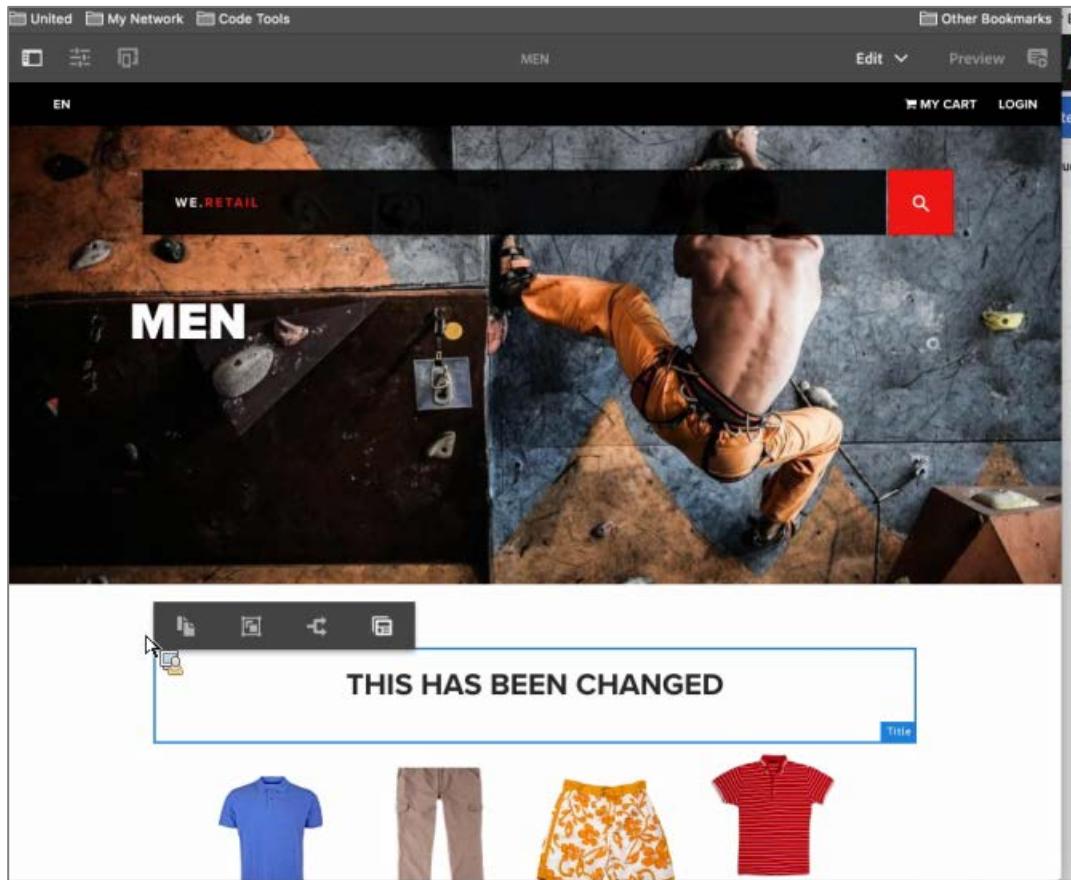
32. Click **Complete** and notice there is a dropdown with Approved and Denied.

If you select **Denied**, the workflow will step back and create new task for Chuck to re-edit the page with the comment Reviewer (Carlene). Otherwise, if you select **Approved**, the workflow will move forward.

+

33. Complete the task by selecting Approved and adding a comment. The task is approved.
34. Impersonate Chuck Grant.
35. 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 as the edit permissions have been removed, as shown:



## References and Helpful Links:

User administration and security: <https://docs.adobe.com/docs/en/aem/6-3/administer/security/security.html>

User, group, and access rights: <https://docs.adobe.com/docs/en/aem/6-3/administer/security/user-group-admin.html>

ACL automation: <https://jackrabbit.apache.org/oak/docs/security/accesscontrol/editing.html>

<https://docs.adobe.com/docs/en/spec/jsr170/javadocs/jcr-2.0/javax/jcr/security/AccessControlManager.html>

<https://www.programcreek.com/java-api-examples/?api=javax.jcr.security.AccessControlEntry>

<http://aempodcast.com/2017/apache-sling/sling-resource-api-vs-jcr-api/>

<https://www.slideshare.net/connectwebex/jcr-sling-or-aem-which-api-should-i-use-and-when>

<https://stackoverflow.com/questions/3908584/when-to-use-jcr-content-repository-over-other-options>

# Writing Tests in Adobe Experience Manager



## Introduction

In Adobe Experience Manager (AEM), the testing frameworks, such as Maven, Mockito, Sling JUnit, and Hobbes tests help automate the testing process. Unit tests in AEM can be set up and run very quickly outside any container. Integration tests can run within an AEM instance. The idea of writing tests in AEM is to detect defects as early as possible, ultimately reducing cost. This testing module is based on how the AEM Archetype approaches testing in a real project.

## Objectives

After completing this course, you will be able to:

- Describe testing framework
- Explain the different types of testing
- Run unit tests and functional tests on your project
- Create unit tests using Mockito
- Create unit tests using Sling Mocks
- Create unit tests using AEM Mocks

# 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 AEM. 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>

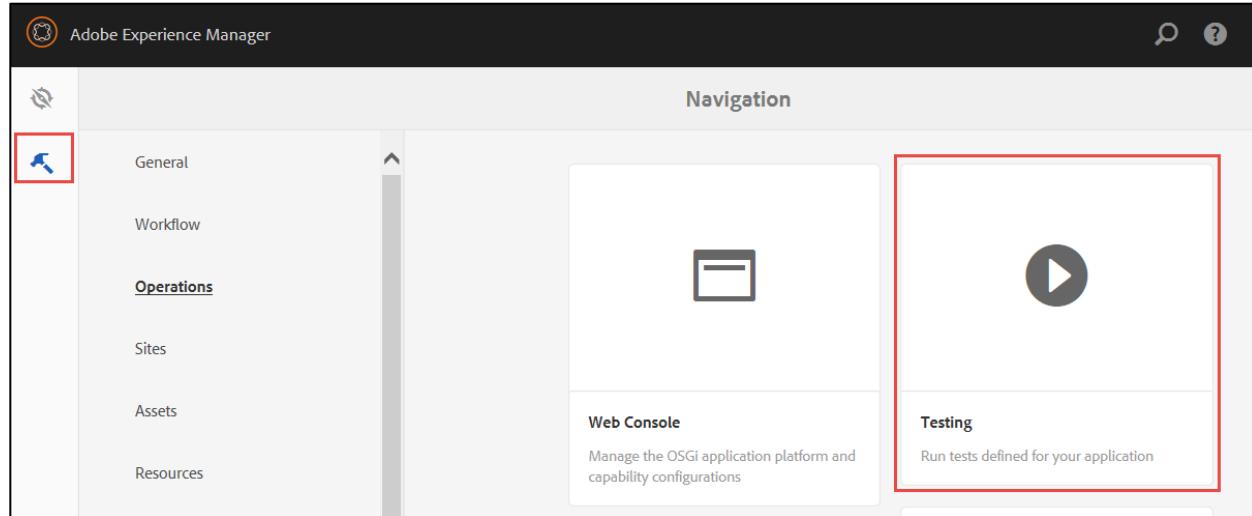
## Performing Functional Tests

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.

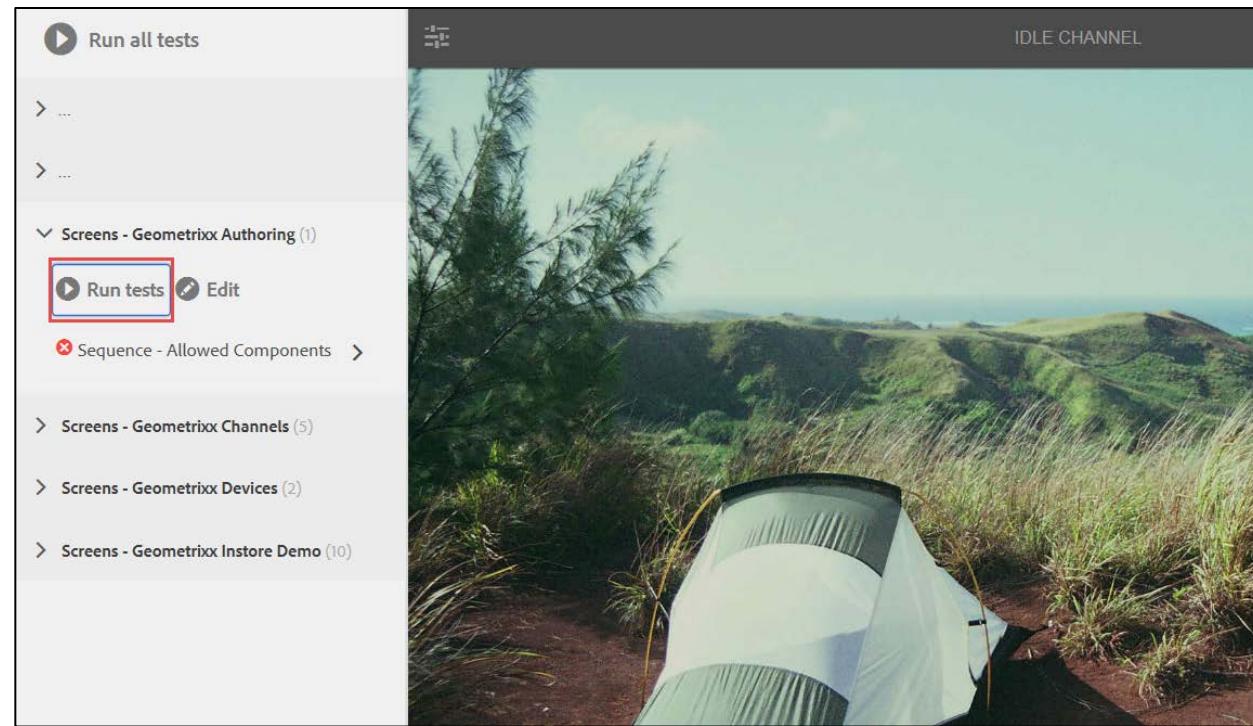
Functional testing does not involve testing of back-end code, but rather the front-end is tested to check if it reacts properly to the user input. Using Test-Driven Development (TDD), you will write a test first, then write the simplest code possible to implement the desired functionality. You will then refactor the code to meet production standards.

### Performing Hobbes Tests

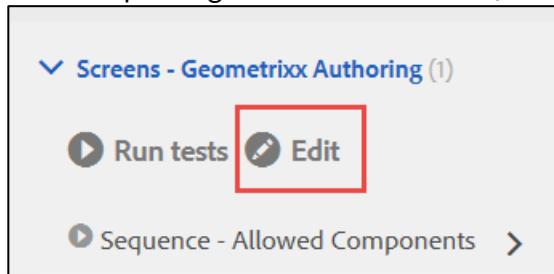
You can access this framework through Tools > Testing in AEM, as shown:



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, as shown:



## Jenkins for Continuous Integration

Continuous integration is a process where all the 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 AEM 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 AEM Integration Server.

Move the feature or bug-fix to the Quality Assurance (QA) team.

QA team will pick up the feature or 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 must be fixed before the next build process can take place.

## Exercise 1: Create unit tests using Mockito

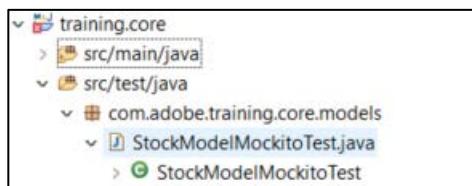
In this lab exercise, you will create a unit test and test StockModel.java class you created previously. This is a basic unit Test outside the server.

1. Launch **Eclipse** by double-clicking the **Eclipse** shortcut on the desktop. The Eclipse Launcher opens.
2. Specify the Workspace as **C:\Workspace** and click **Launch**. The Eclipse Workspace opens.
3. In **Project Explorer**, navigate to **training.core > src/test/java**.
4. Copy the file **StockModelMockitoTest.java** from **/Exercise\_Files/14\_Writing\_Tests/**.



**Note:** The code is provided as part of the Exercise\_Files under /Exercise\_Files/14\_Writing\_Tests/. Copy and paste the file from the exercise files referenced to StockModelMockitoTest.java to Eclipse. Please do not copy the code from this exercise book. The code in the student guide is for illustrative purposes only.

5. In Eclipse, right-click **com.adobe.training.core.models** and paste the file. The **StockModelMockitoTest.java** class is created, as shown:



6. Double-click **StockModelMockitoTest.java**. The file opens in the editor.

```
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:
```

```

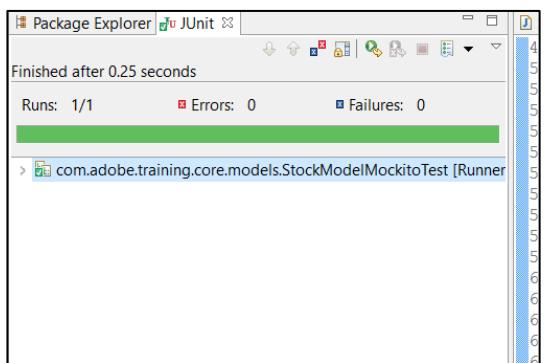
21. * com.adobe.training.core.models.StockModel.java
22. *
23. * To correctly use this testing class:
24. * -
25. *
26. */
27. public class StockModelMockitoTest {
28.
29.     private StockModel stock;
30.
31.     @Before
32.     public void setup() throws Exception {
33.
34.         //Adapt the Resource if needed
35.         Resource RESOURCE_MOCK = mock(Resource.class);
36.         StockModel STOCKMODEL_MOCK = mock(StockModel.class);
37.         when(RESOURCE_MOCK.adaptTo(StockModel.class)).thenReturn(STOCKMODEL_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 requestData Property
48.         int numberofDays = randBetween(1,365);
49.         Date date = new SimpleDateFormat("D").parse(numberofDays + " " + Calendar.getInstance
        ().get(Calendar.YEAR));
50.         String tradeDate = new SimpleDateFormat("MM/dd/yyyy").format(date);
51.         when(STOCKMODEL_MOCK.getRequestDate()).thenReturn(tradeDate);
52.
53.     }
54.
55.     private static int randBetween(int start, int end) {
56.         return start + (int)Math.round(Math.random() * (end - start));
57.     }
58.
59.     @Test
60.     public void testGetLadeTradeValue() throws Exception{
61.         assertNotNull("lastTradeModel is null", stock);
62.         assertTrue("lastTrade value is incorrect", stock.getLastTrade() > 100);
63.         assertFalse("requestData value is incorrect", stock.getRequestDate().isEmpty());
64.     }
65.
66. }
```

## 7. Examine the code.

**Note:** This class will test the Java class **StockModel.java**.

## 8. Right-click StockModelMockitoTest.java and select Run As > Junit Test. The Junit Test starts.

9. Verify the test ran successfully, as shown:



**Note:** Alternatively, you can run all the tests under `src/test/java` by right-clicking `training.core` and selecting `Run As > Maven test`.

## Exercise 2: Create unit tests using Sling Mocks

In this lab exercise, you will create unit tests using Sling mocking framework.

This exercise includes three tasks:

1. Observe the sling-mock dependency
2. Create a unit test with sample data
3. Run the test

### Task 1: Observe the sling-mock dependency

1. In Project Explorer, navigate to training > pom.xml.
2. Double-click **pom.xml** to open it in editor. The **pom.xml** opens.

```
570      </dependency>
571      <!-- Sling Testing Resource Resolver Mock dependency -->
572      <dependency>
573          <groupId>org.apache.sling</groupId>
574          <artifactId>org.apache.sling.testing.sling-mock</artifactId>
575          <version>2.2.18</version>
576          <scope>test</scope>
577      </dependency>
```



**Note:** This dependency allows you to use the Sling Mocks framework. Sling Mocks allow you to create mock resources to test our different classes. Mock resources can be autogenerated using JSON files.

3. Navigate to training > core > pom.xml.
4. Double-click **pom.xml** to open it in editor. The **pom.xml** opens.
5. Observe the **org.apache.sling.testing.sling-mock** dependency has been added to **training.core** as well (around line 123).

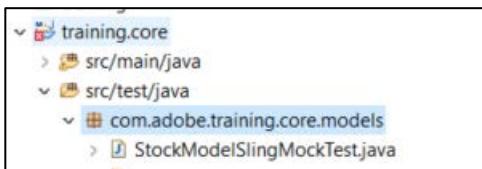
## Task 2: Create a unit test with sample data

1. In Project Explorer, navigate to training.core > src/test/java.
2. Copy the file StockModelSlingMockTest.java from [/Exercise\\_Files/14\\_Writing\\_Tests/](#).



**Note:** The code is provided as part of the Exercise\_Files under [/Exercise\\_Files/14\\_Writing\\_Tests/](#). Copy and paste the file from the exercise files referenced to StockModelSlingMockTest to Eclipse. Please do not copy the code from this exercise book. The code in the student guide is for illustrative purposes only.

3. In Eclipse, right-click **com.adobe.training.core.models** and paste the file. The **StockModelSlingMockTest.java** class is created, as shown:



4. Double-click **StockModelSlingMockTest.java**. The file opens in editor, as shown:

```
1. package com.adobe.training.core.models;
2.
3.
4. import static org.junit.Assert.assertEquals;
5. import static org.junit.Assert.assertNotNull;
6.
7. import org.apache.sling.servlethelpers.MockSlingHttpServletRequest;
8.
9. import org.apache.sling.testing.mock.sling.ResourceResolverType;
10. import org.apache.sling.testing.mock.sling.junit.SlingContext;
11.
12. import org.junit.Before;
13. import org.junit.Rule;
14. import org.junit.Test;
15.
16.
17. /**
18. * Tests the StockModelAdaptFromRequest using the Sling Mock implementation for Sling APIs
19. *
20. * A Sling context (the mock environment) needs to be created to test the classes.
21. * The implementation used in that example is the ResourceResolver mock,
22. * to allow in-memory reading and writing resource data using the Sling Resource API.
23. *
24. * It also provides support for Sling Models (Sling Models API 1.1 and Impl 1.1 or higher required)
25. *
26. * Note that the testable class is under /src/main/java:
27. * com.adobe.training.core.models.StockModelAdaptFromRequest.java
28. *
29. * To correctly use this testing class:
```

```

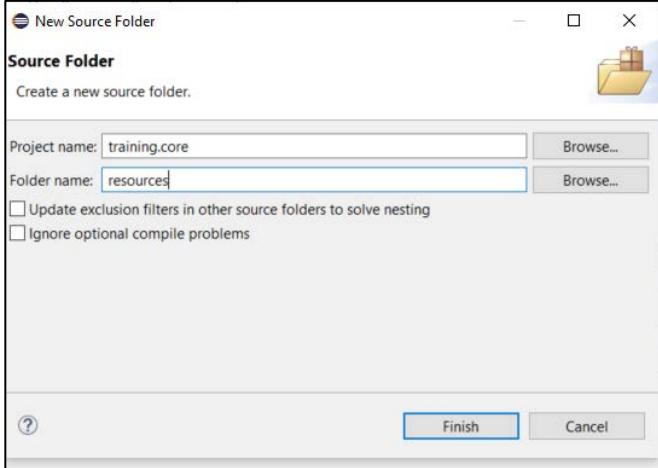
30. * -
   put this file under training.core/src/test/java in the package com.adobe.training.core.models
31. *
32. */
33.
34.
35. public class StockModelSlingMockTest {
36.
37.     private final double EPSILON = 2.0;
38.
39.
40.     @Rule
41.     public final SlingContext context = new SlingContext(ResourceResolverType.RESOURCE_RESOLVE_R_MOCK);
42.
43.     private StockModelAdaptFromRequest stockModel;
44.
45.
46.     @Before
47.     public final void setup() throws Exception{
48.         // Mock the request object with the class MockSlingHttpServletRequest
49.         MockSlingHttpServletRequest request = context.request();
50.
51.         // Load the test content from a json file
52.         context.load().json("/adbe-content.json", "/content/stocks/ADBE");
53.
54.         // Register our model to allow adaptation from the context request
55.         context.addModelsForClasses(StockModelAdaptFromRequest.class);
56.
57.         // The model accesses the resource data through request suffix
58.         context.requestPathInfo().setSuffix("/content/stocks/ADBE");
59.
60.         // Adapting the request to our model for testing
61.         stockModel = request.adaptTo(StockModelAdaptFromRequest.class);
62.
63.     }
64.
65.
66.     @Test
67.     public void testStockModel() {
68.
69.         // Validate that the request has been successfully adapted to the model
70.         assertNotNull(stockModel);
71.         // Test if expected and actual values for lastTrade are off by the value of EPSILON
72.         assertEquals(220.0, stockModel.getLastTrade(), EPSILON);
73.     }
74.
75.
76. }

```

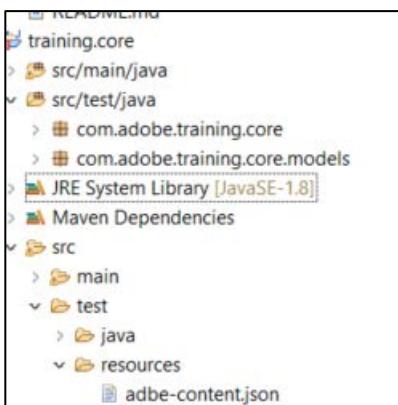
5. Examine the code. Observe how the mock test loads a json file named /adbe-content.json.
6. In Project Explorer, navigate to training.core > src/test/java.

7. Right-click **src/test** and select **New > Source Folder**. The **New Source Folder** screen opens.
8. Provide the values to create the folder, as shown:

- o Folder name: **resources**

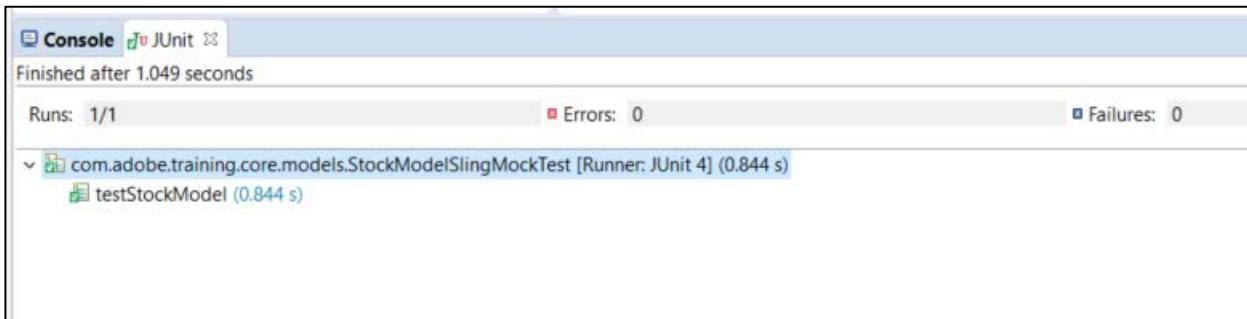


9. Click **Finish**. The folder is created.
10. Copy the file `adbe-content.json` from **/Exercise\_Files/14\_Writing\_Tests/**.
11. Right-click **resources** and select **Paste**, as shown:

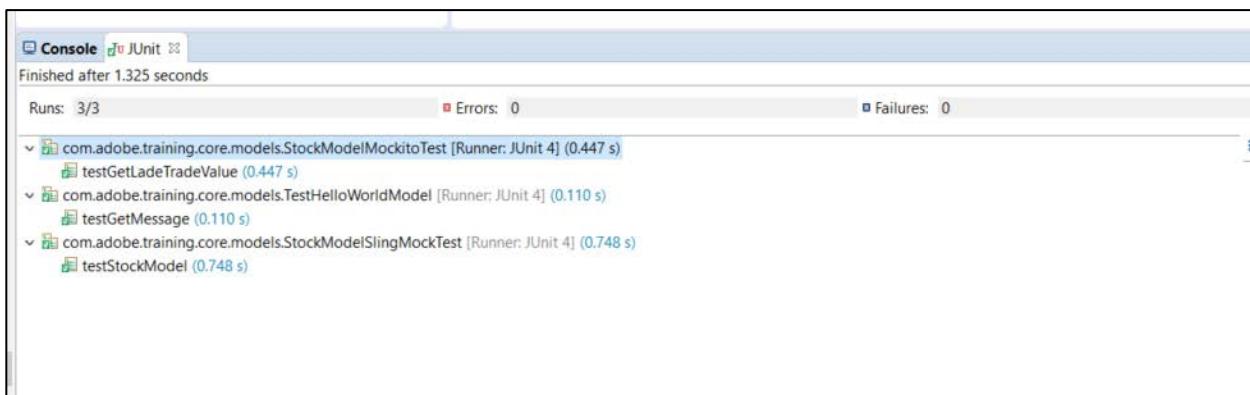


## Task 3: Run the test

1. Right-click StockModelSlingMockTest.java and select Run As > Junit Test. The test runs.
2. Verify the test ran sucessfully, as shown:



3. To run all the tests within the bundle, right-click **training.core** and select Run As > Junit Test.
4. Verify the tests ran sucessfully, as shown:



## Exercise 3: Create unit tests using AEM Mocks

In this lab exercise, you will create unit tests using AEM mocks.

This exercise includes three tasks:

1. Observe the AEM-mock dependency
2. Create a unit test with sample data
3. Run the test

### Task 1: Observe the AEM-mock dependency

1. In Project Explorer, navigate to **training > pom.xml**.
2. Double-click **pom.xml** to open it in editor. The **pom.xml** opens.
3. Look for the dependency **io.wcm.testing.aem-mock** (around line 585), as shown:

```
585@    <!-- AEM Mock dependency from AEM 8.0 library -->
586      <dependency>
587          <groupId>io.wcm</groupId>
588          <artifactId>io.wcm.testing.aem-mock</artifactId>
589          <version>2.2.0</version>
590          <scope>test</scope>
591      </dependency>
```

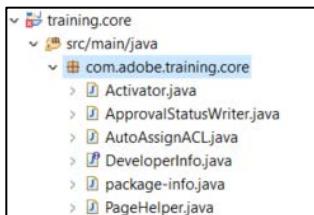


**Note:** This dependency allows you to use the AEM Mocks framework. AEM mocks allow you to mock objects from the AEM APIs through json files.

4. Navigate to **training > core > pom.xml**.
5. Double-click **pom.xml** to open it in editor. The **pom.xml** opens.
6. Observe the **io.wcm.testing.aem-mock** dependency has been added to **training.core** as well (around line 128).

## Task 2: Create a unit test with sample data

1. Copy the file PageHelper.java from **/Exercise\_Files/14\_Writing\_Tests/**
2. In Project Explorer, navigate to **training.core > src/main/java**.
3. Right-click **com.adobe.training.core** and paste the file, as shown:



4. Double-click **PageHelper.java**. The file opens in editor, as shown:

```

5. package com.adobe.training.core;
6.
7. import org.apache.commons.lang3.StringUtils;
8. import org.slf4j.Logger;
9. import org.slf4j.LoggerFactory;
10.
11. import com.day.cq.wcm.api.Page;
12.
13. public class PageHelper {
14.
15.     private static final Logger LOGGER = LoggerFactory.getLogger(PageHelper.class);
16.
17.
18.     /**
19.      * Returns the page title of the given page. If the page title is empty it will fallback to the title and t
20.      * o the
21.      * name of the page.
22.      * @param page The page.

```

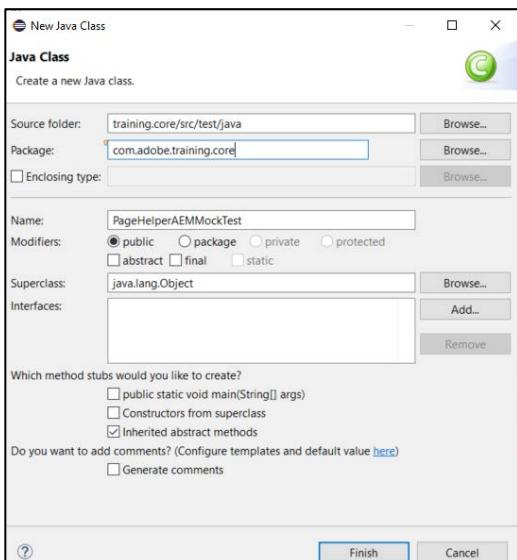
```
22. * @return The best suited title found (or <code>null</code> if page is <code>null</code>).
23. */
24. public static String getPageTitle(final Page page) {
25.     if (page != null) {
26.         final String title = page.getPageTitle();
27.         if (StringUtils.isBlank(title)) {
28.             return getTitle(page);
29.         }
30.         return title;
31.     } else {
32.         LOGGER.debug("Provided page argument is null");
33.         return null;
34.     }
35. }
36.
37. /**
38. * Returns the title of the given page. If the title is empty it will fallback to the name of the page.
39. * @param page The page.
40. * @return The best suited title found (or <code>null</code> if page is <code>null</code>)
41. */
42. public static String getTitle(final Page page) {
43.     if (page != null) {
44.         final String title = page.getTitle();
45.         if (StringUtils.isBlank(title)) {
46.             return page.getName();
47.         }
48.         return title;
49.     } else {
50.         LOGGER.debug("Provided page argument is null");
```

```
51.     return null;  
52. }  
53. }  
54.  
55. }
```

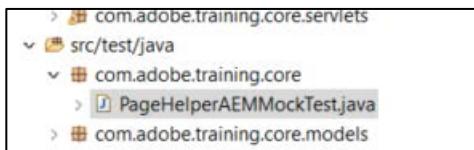


**Note:** Now that you have your PageHelper class, you can test it. You will create a AEM Mock class and the json with our test data.

4. In Project Explorer, navigate to training.core > src/test/java.
5. In Eclipse, right-click **src/test/java** and select **New > class**. The **New Java class** window opens.
6. Provide the values:
  - o Package: **com.adobe.training.core**
  - o Name: **PageHelperAEMMockTest**



7. Click **Finish**. The class gets created, as shown:



8. Copy the content of the file `PageHelperAEMMockTest.java` from [/Exercise\\_Files/14\\_Writing\\_Tests/](#)
9. In Eclipse, double-click **PageHelperAEMMockTest.java** in the editor, and replace the content with the one copied from the exercise file.
10. Save the changes.

11. Double-click **PageHelperAEMMockTest.java**. The file opens in the editor, as shown:

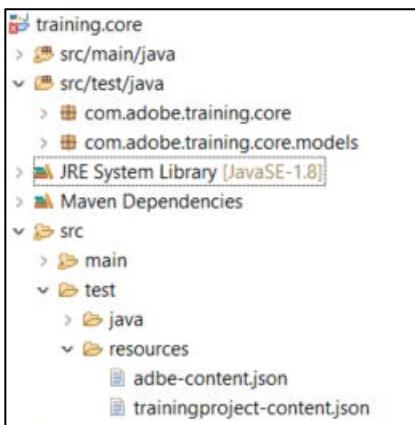
```

1. package com.adobe.training.core;
2.
3. import static org.junit.Assert.assertEquals;
4.
5. import org.junit.Before;
6. import org.junit.Rule;
7. import org.junit.Test;
8.
9. import com.day.cq.wcm.api.Page;
10.
11. import io.wcm.testing.mock.aem.junit.AemContext;
12.
13. public class PageHelperAEMMockTest {
14.
15.
16.     /**
17.      * Tests the helper methods of the class PageHelper
18.      *
19.      * An AEM context object (the mock environment) needs to be created to test the methods,
20.      * as they require an AEM Page object to be injected.
21.      *
22.      * AEM Mocks is a mock implementation that extends the Sling Mocks implementation
23.      * http://wcm.io/testing/aem-mock/
24.      *
25.      * Note that the testable class is under /src/main/java:
26.      * com.adobe.training.core.PageHelper.java
27.      *
28.      * To correctly use this testing class:
29.      * -
30.      * put this file under training.core/src/test/java in the package com.adobe.training.core
31.      */
32.
33.
34.     @Rule
35.     public AemContext context = new AemContext();
36.
37.     private Page page;

```

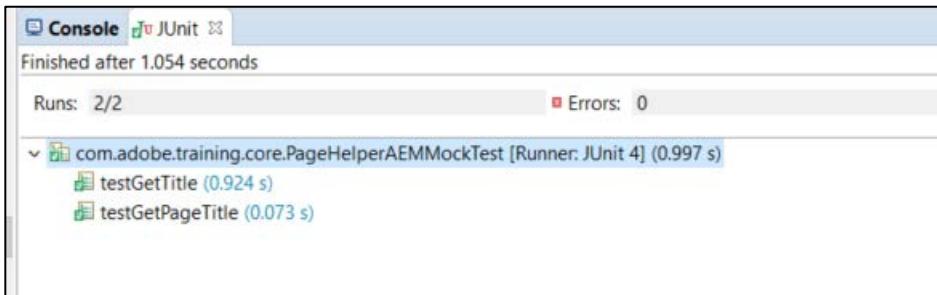
```
38.
39.     @Before
40.     public void setup() throws Exception{
41.         // Load the test content from a json file
42.         context.load().json("/trainingproject-content.json", "/content/trainingproject");
43.
44.         // Adapt the resource defined by the path to an AEM Page
45.         page = context.currentPage("/content/trainingproject/en");
46.     }
47.
48.
49.     @Test
50.     public void testGetPageTitle() throws Exception{
51.
52.         assertEquals("New Project", PageHelper.getPageTitle(page));
53.
54.     }
55.
56.     @Test
57.     public void testgetTitle() throws Exception{
58.
59.         assertEquals("English", PageHelper.getTitle(page));
60.
61.     }
62.
63. }
```

12. Examine the code. Observe how the mock test loads a json file named **/trainingproject-content.json**.
13. Copy the file **trainingproject-content.json** from **/Exercise\_Files/14\_Writing\_Tests/**.
14. Right-click **resources** and select **Paste**, as shown:

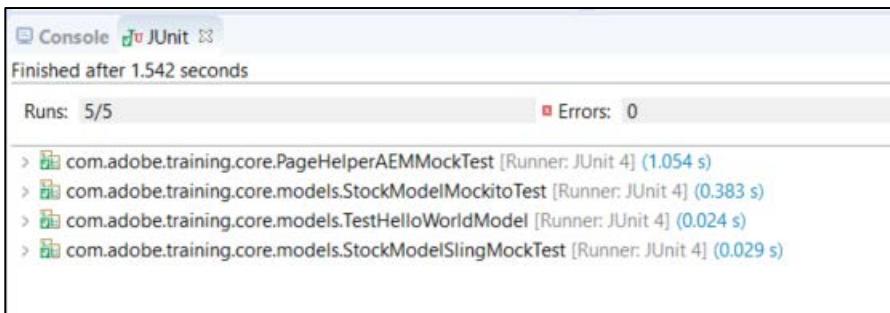


## Task 3: Run the test

1. Right-click PageHelperAEMMockTest.java and select Run As > Junit Test. The test runs.
2. Verify the test ran sucessfully, as shown:



3. To run all tests within the bundle, right-click **training.core** and select Run As > Junit Test.
4. Verify the tests ran sucessfully, as shown:



## References and Helpful Links:

Sling Mocks: <https://sling.apache.org/documentation/development/sling-mock.html>

AEM Mocks: <http://wcm.io/testing/aem-mock/usage.html>

Maven dependency: <https://mvnrepository.com/artifact/io.wcm/io.wcm.testing.aem-mock>

Maven dependency: <http://wcm.io/testing/aem-mock/usage-content-loader-builder.html>