Extend and Customize Adobe Experience Manager

STUDENT WORKBOOK

ADOBE DIGITAL LEARNING SERVICES

Extend and Customize Adobe Experience Manager

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

04/23/2020

Extend and Customize Adobe Experience Manager

The Adobe instructor-led course, Extend and Customize Adobe Experience Manager, is a combination of modules from two different courses. You will have received two Student Guides and two exercise folders, one for each course:

- AEM Technical Basics
- Extend and Customize Adobe Experience Manager

This table shows the order in which those courses, and the modules within them, flow.

Module #	**Course Name**	**Module Name**
1	AEM Technical Basics	AEM Architecture
2	AEM Technical Basics	AEM Installation
3	AEM Technical Basics	AEM Developer Tools
4	AEM Technical Basics	AEM Sites Authoring Basics
6	AEM Technical Basics	Creating a Project Using Maven
7	AEM Technical Basics	Eclipse with an AEM project
1 thru 9	Extend and Customize AEM	All modules
11	AEM Technical Basics	Cloud Manager Basics

Contents

OSGi configurations and runmodes

Introduction

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

Objectives

After completing this course, you will be able to:

- Explain OSGi configurations
- Explain the supported runmodes
- Explain how OSGi configurations are organized to match runmodes
- Create an OSGi configuration
- Start AEM with an environment run mode

OSGi configurations

OSGi is a fundamental element in the technology stack of AEM. OSGi also supports the 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.

OSGI configuration should be committed to source control rather than through the Web Console. Techniques include:

- Making the necessary changes on the developer's local AEM environment with the AEM Web Console's configuration manager and then exporting the results to the AEM project on the local file system
- Creating the OSGI configuration manually in the AEM project on the local file system, referencing the AEM console's configuration manager for the property names.

Creating OSGi configurations 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

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.

Supported Runmodes

In existing AEM solutions, customers have the option of running services with arbitrary run modes and apply OSGI configuration or install OSGI bundles to those specific services. Run modes that are defined typically include the "service" (author and publish) and the environment (dev, stage, and prod), but there could be more (for example, local-dev, QA, and so on).

AEM as a Cloud Service on the other hand is more opinionated about which run modes are available and how OSGI bundles and OSGI configuration can be mapped to them:

- OSGI configuration run modes must reference dev, stage, and prod for the environment or author, publish for the service. A combination of <service>.<environment_type> is being supported whereas these must be used in this particular order (for example, author.dev or publish.prod). The OSGI tokens should be referenced directly from code rather than using the getRunModes method, which will no longer include the environment_type at runtime.
- OSGI bundles run modes are limited to the service (author, publish). You should install per-run mode OSGI bundles in the content package under install/author OR install/publish.

Like the existing AEM solutions, there is no way to use run modes to install just content (for example, /content) for specific environments or services. So for example, if it was desired to seed a dev environment with data or HTML that is not on stage or production, Package Manager could be used.

The supported runmode configurations are:

- config (The default, applies to all AEM services)
- config.author (Applies to all AEM Author services)
- config.author.dev (Applies to AEM Dev Author services)
- config.author.stage (Applies to AEM Staging Author services)
- config.author.prod (Applies to AEM Production Author services)
- config.publish (Applies to AEM Publish services)
- config.publish.dev (Applies to AEM Dev Publish services)
- config.publish.stage (Applies to AEM Staging Publish services)
- config.publish.prod(Applies to AEM Production Publish services)
- config.dev (Applies to AEM Dev services)
- config.stage (Applies to AEM Staging services)
- config.prod(Applies to AEM Production services)

The OSGI configuration that has the most matching runmodes is used.

When developing locally, a runmode startup parameter can be passed in to specify which runmode OSGI configuration will be used. For example:

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

Exercise 1: Create an OSGi configuration

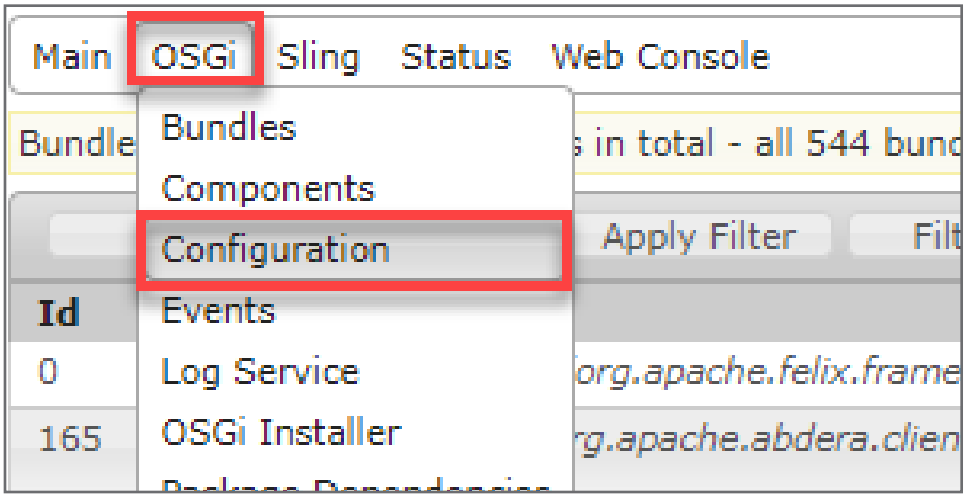
Scenario: As a developer, you need to create OSGi configurations in the JCR using /apps.

In a typical development process, there are multiple servers designated for specific tasks in your infrastructure. Each server could have different configurations depending on the service and deployment type.

Task 1: Create an OSGi configuration

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 Service 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.**

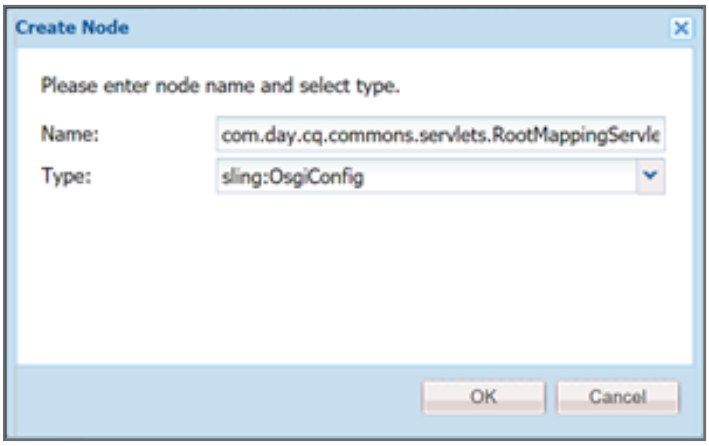


4. To locate the **Root Mapping** OSGi configuration, press Ctrl+F to open a "find" window. In the window, type these words to search: **Day CQ Root Mapping** in your browser. The search string's corresponding words will become orange, indicating a match for your search string, as shown.

5. Click the **Day CQ Root Mapping** configuration to open it in a dialog window:

6. 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.
7. In a new browser tab, open **CRXDE Lite** by navigating to http://localhost:4502/crx/de (or within AEM, use **Tools** > **General** > **CRXDE Lite**).
8. Navigate to **apps** > **training**.
9. Right-click the **config.author** folder node and create a new node by selecting **Create** > **Create Node**.

10. Specify the following values for **Name** and **Type**:



› 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**

Note: 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.

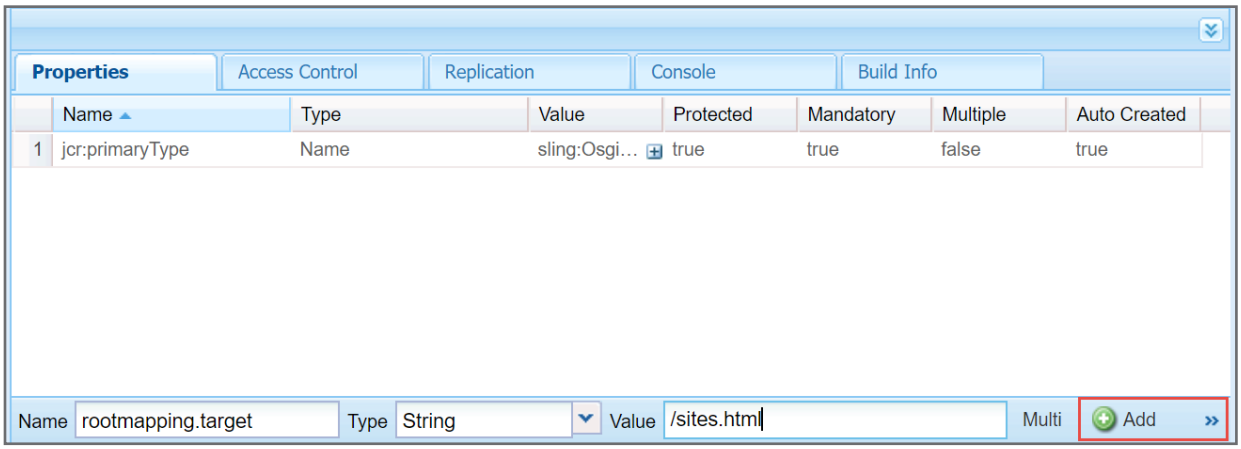
11. Click **OK** and then click **Save All** in CRXDE Lite to ensure your new node is saved.

12. With the **com.day.cq.commons.servlets.RootMappingServlet** node selected, navigate to the JCR Properties tab of CRXDE Lite and add the following:

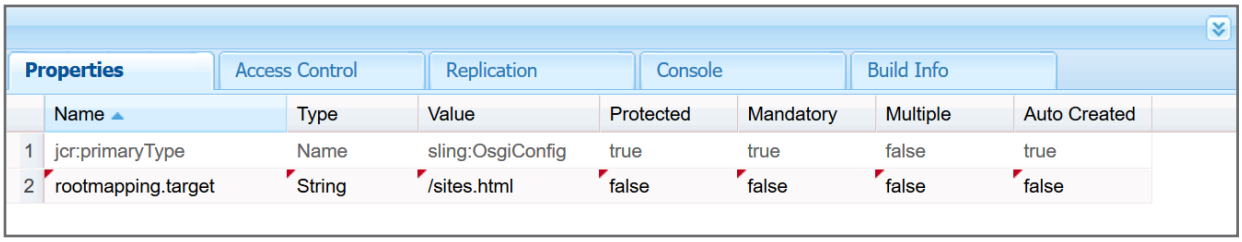
- Name: **rootmapping.target**
- Type: **String**
- Value: **/sites.html**

Note: You can obtain the property value of **rootmapping.target** 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 screenshot in Step #4 to see how this matches.

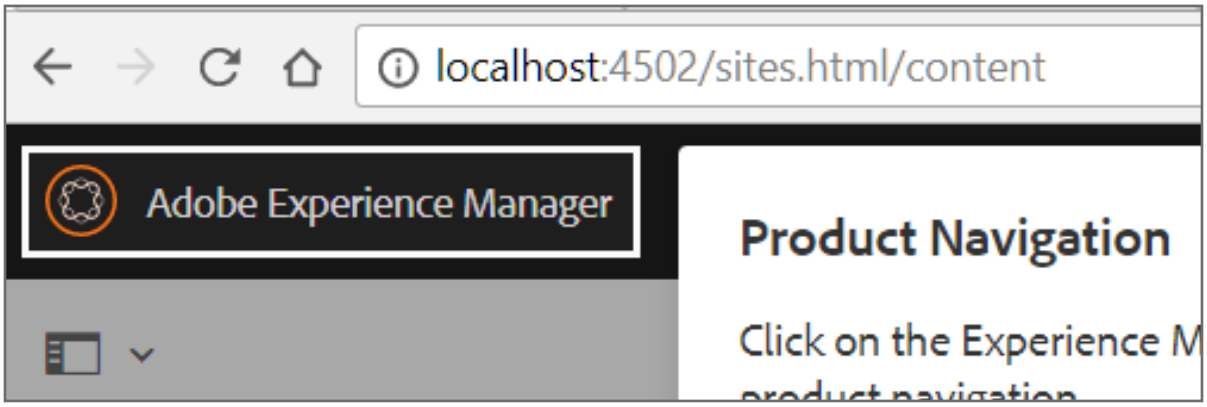
13. Click **Add** in the bottom-right corner in the **JCR Properties** tab to add your property, as shown:



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



14. Click **Save All** in CRXDE Lite.
15. Open a new browser tab and navigate to the AEM author Service (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 Service is using the author run mode currently. In the next exercise, you will use a tool to check the current run modes of a Service.

Note: Because we created this node in the **CRXDE Lite** , it needs to be synchronized back to your local Maven project. Refer to the **Project with Eclipse** module on how to import content to local project.

Exercise 2: Create a configuration for an environment

Scenario: As a developer you need to:

- Be able to start AEM with an environment run mode using the command line
- Create multiple sets of custom OSGi configurations to match supported run modes in order to specify different groups of settings for each Service or environment you are deploying

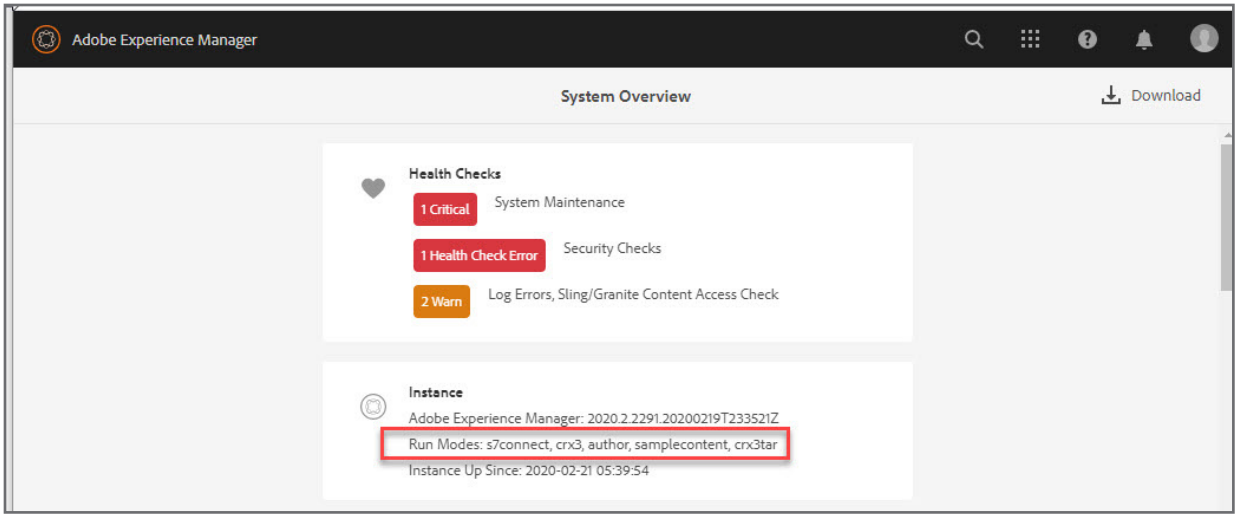
This exercise includes the following tasks:

1. Start AEM with an environment run mode
2. Create a custom OSGi configuration node for an environment run mode

Task 1: Start AEM with an environment run mode

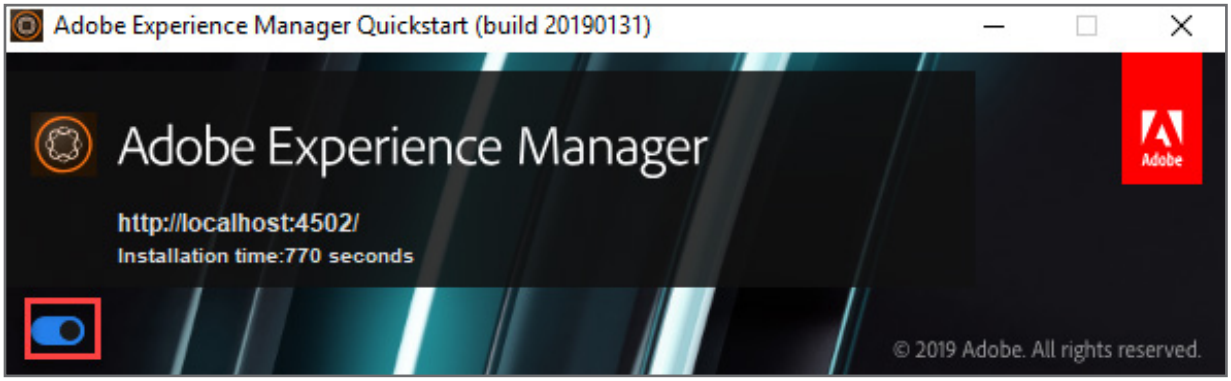
In this task, you will start your AEM author Service using an environment run mode called **dev** via the command-line.

1. Within AEM, click **Tools** > **Operations** > **System Overview**.
2. In the Instance area, observe the current run modes your author Service is using (**s7connect, crx3, author, samplecontent,** and **crx3tar**):



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

3. Shut down your running AEM author Service by clicking the **ON / OFF** toggle button in the GUI window, as shown:



If you are running AEM using the command line, use **CTRL+C** (in Windows) in your command window to shut down AEM.

4. Start AEM using the command line in order to use an environment run mode. This is because the environment run modes can be generated using command line parameters. Navigate to the directory on your machine where your author Service quickstart file resides.

Note: If you are using a ReadyTech Service, this directory should be **C:/adobe/aem-sdk/**.

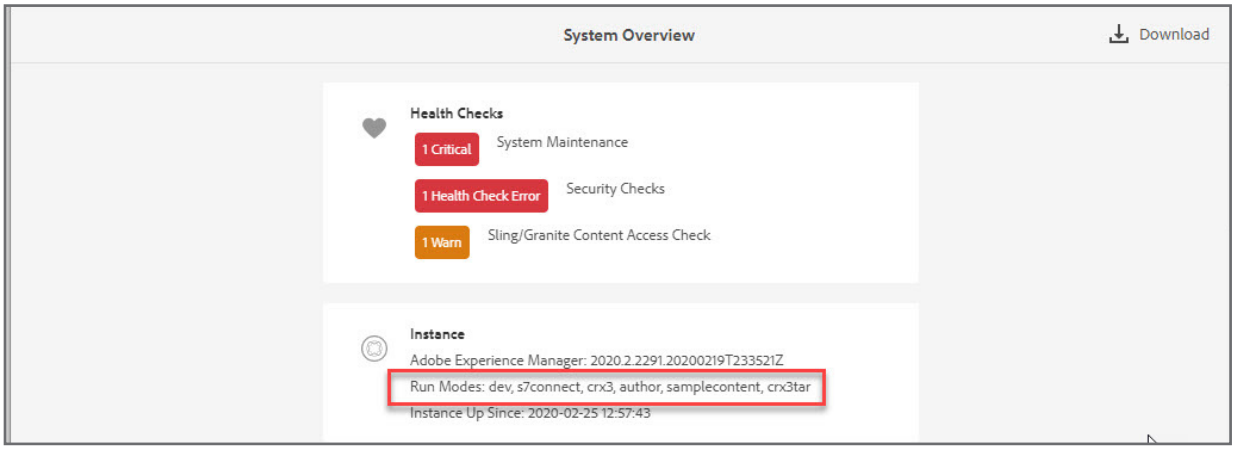
5. Start AEM again using the following command that specifies the author and dev run modes:

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

6. Verify your AEM author service started, this time with a command window available to view details of the startup. In addition, the GUI window will be available.

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

7. Sign in to AEM again.
8. In AEM, click **Tools** > **Operations** > **System Overview**. Your environment run mode **dev** should now appear:

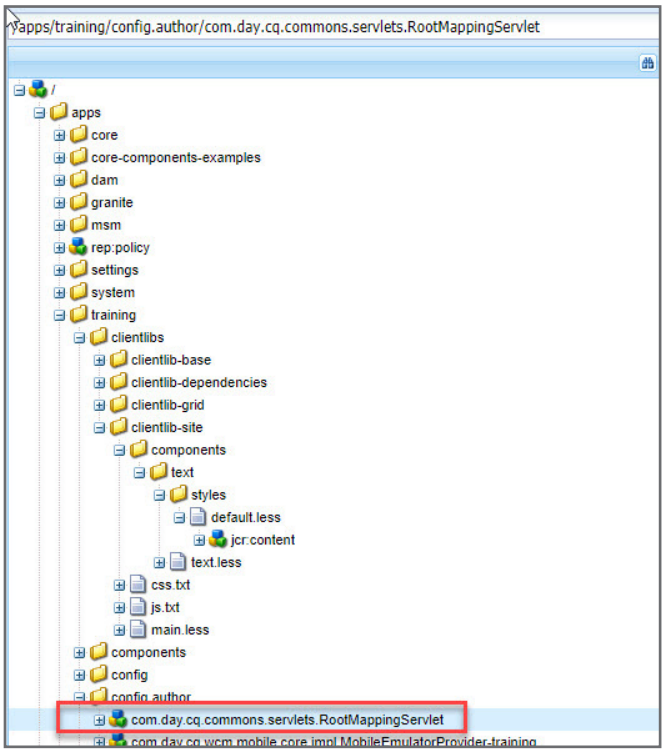


Task 2: Create a custom OSGi configuration node for an environment run mode

In this task, you will change the default "root" page to load CRXDE Lite for the "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** > **training.**
3. Right-click the **training** folder and select **Create** > **Create Folder** to create a new folder.
4. Type **config.author.dev** in the **Name** field.
5. Click **OK** and click **Save All** in CRXDE Lite to ensure your new folder is saved.

   Now, instead of manually creating another root mapping node that is very similar to the one you created in Exercise 1, you can copy the node you created and paste it into your new folder and make the necessary changes.
6. Right-click the **com.day.cq.commons.servlets.RootMappingServlet** node under **/apps/training/config.author** folder and select **Copy**.



7. Right-click the **config.author.dev** folder and select **Paste**.
8. Click **Save All**.

9. Verify your node structure looks like this:

10. Now, to change the properties of your copied node (config.author.dev), double-click the **Value** field for the **rootmapping.target** property (for com.day.cq.commons.servlets.RootMappingServlet) in the **Properties** tab. This will make the field editable.
11. Enter **/crx/de/index.jsp** in the **Value** field.
12. Click **Save All** in CRXDE Lite.
13. To verify if the change was made to the dev run mode, log out of your author service. Open a browser tab with http://localhost:4502 and sign in. The browser should automatically navigate you to the CRXDE Lite 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 environment 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, refer:

- Configuring Run Modes: https://docs.adobe.com/help/en/experience-manager-cloud-service/implementing/deploying/overview.html#runmodes
- OSGi configurations: https://docs.adobe.com/help/en/experience-manager-cloud-service/implementing/deploying/overview.html#osgi-configuration

Logging with 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 your own log files for local development. Your custom logs can speed up local development to quickly find messages during application development.

Objectives

After completing this course, you will be able to:

- Explain the Logging system in AEM
- Explain the types of log files in AEM
- Observe the available logs
- Explain logging for AEM as a Cloud Service

AEM Logging system

The logging framework 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.slng.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

Types of Log Files in AEM

Audit.log: Registers all modern actions

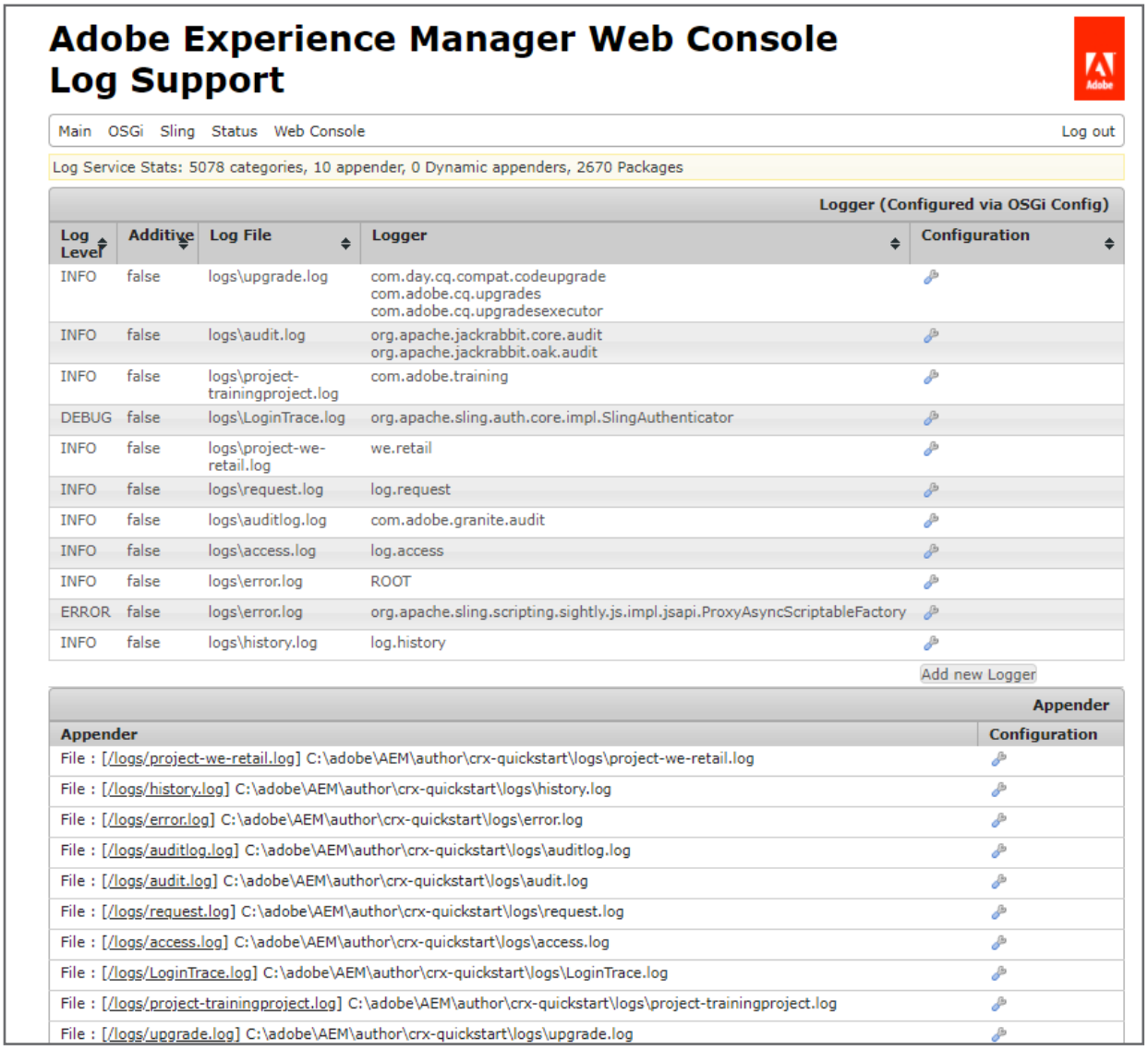
- Error.log: Registers all error messages
- Request.log: Registers all access requests along with their responses
- Stderr.log: Holds error messages generated during startup
- Upgrade.log: Provides a log of all upgrade operations
- Access.log: Registers all access requests sent to AEM and the repository

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

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.

If you are using AEM locally, you can also view the log files through the Web Console at:

http://localhost:4502/system/console/slinglog



Adobe Experience Manager Web Console
Log Support

Main OSGi Sling Status Web Console Log out

Log Service Stats: 5078 categories, 10 appender, 0 Dynamic appenders, 2670 Packages

Logger (Configured via OSGi Config)

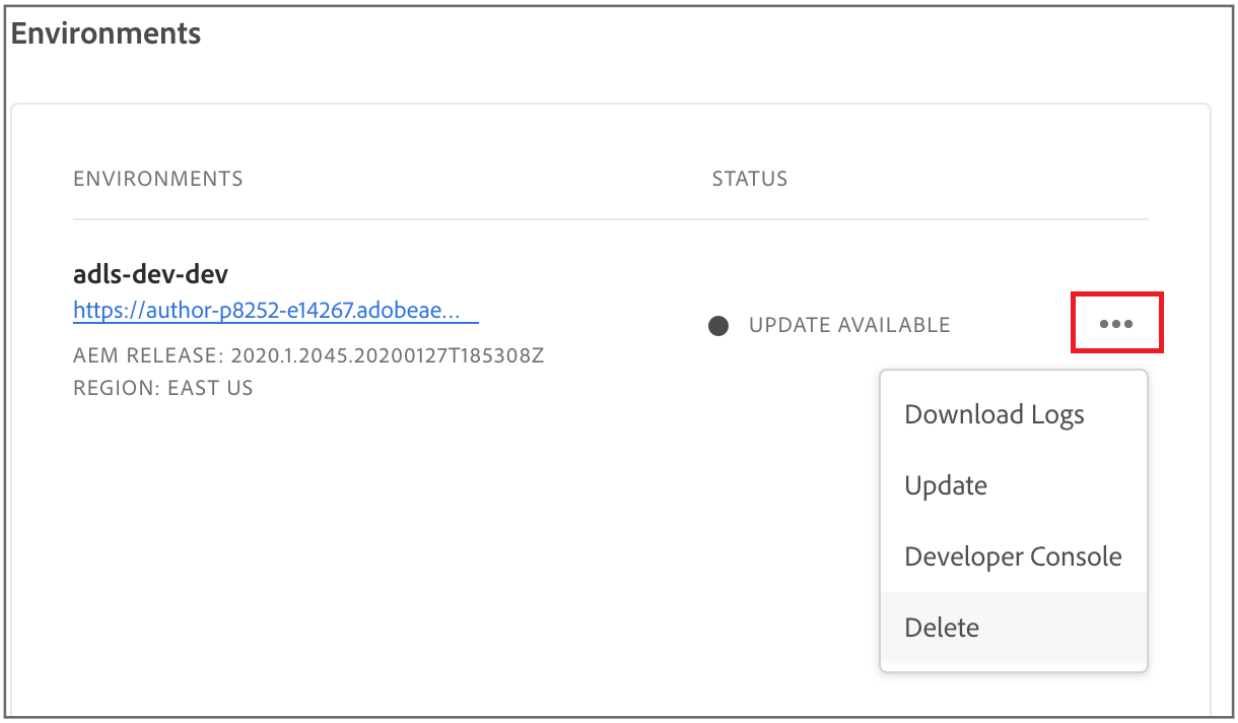
Log Level	Additive	Log File	Logger	Configuration
INFO	false	logs\upgrade.log	com.day.cq.compat.codeupgrade com.adobe.cq.upgrades com.adobe.cq.upgradesexecutor	
INFO	false	logs\audit.log	org.apache.jackrabbit.core.audit org.apache.jackrabbit.oak.audit	
INFO	false	logs\project-trainingproject.log	com.adobe.training	
DEBUG	false	logs\LoginTrace.log	org.apache.sling.auth.core.impl.SlingAuthenticator	
INFO	false	logs\project-we-retail.log	we.retail	
INFO	false	logs\request.log	log.request	
INFO	false	logs\auditlog.log	com.adobe.granite.audit	
INFO	false	logs\access.log	log.access	
INFO	false	logs\error.log	ROOT	
ERROR	false	logs\error.log	org.apache.sling.scripting.sightly.js.impl.jsapi.ProxyAsyncScriptableFactory	
INFO	false	logs\history.log	log.history	

Add new Logger

Appender

Appender	Configuration
File : [/logs/project-we-retail.log] C:\adobe\AEM\author\crx-quickstart\logs\project-we-retail.log	
File : [/logs/history.log] C:\adobe\AEM\author\crx-quickstart\logs\history.log	
File : [/logs/error.log] C:\adobe\AEM\author\crx-quickstart\logs\error.log	
File : [/logs/auditlog.log] C:\adobe\AEM\author\crx-quickstart\logs\auditlog.log	
File : [/logs/audit.log] C:\adobe\AEM\author\crx-quickstart\logs\audit.log	
File : [/logs/request.log] C:\adobe\AEM\author\crx-quickstart\logs\request.log	
File : [/logs/access.log] C:\adobe\AEM\author\crx-quickstart\logs\access.log	
File : [/logs/LoginTrace.log] C:\adobe\AEM\author\crx-quickstart\logs\LoginTrace.log	
File : [/logs/project-trainingproject.log] C:\adobe\AEM\author\crx-quickstart\logs\project-trainingproject.log	
File : [/logs/upgrade.log] C:\adobe\AEM\author\crx-quickstart\logs\upgrade.log	

You can download logs in Cloud Manager by selecting the ellipsis on the selected environment, as shown:



Custom logs for local development

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. By default, all logs in AEM use the same logging writer.

Logging Loggers:

Default loggers have the following configuration set:

- 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

Creating Your Own Loggers and Writers for local development

For local development, you will typically only create a Logger since the default writer (writes once a day) is good enough for development.
To define your Logger/Writer pair:

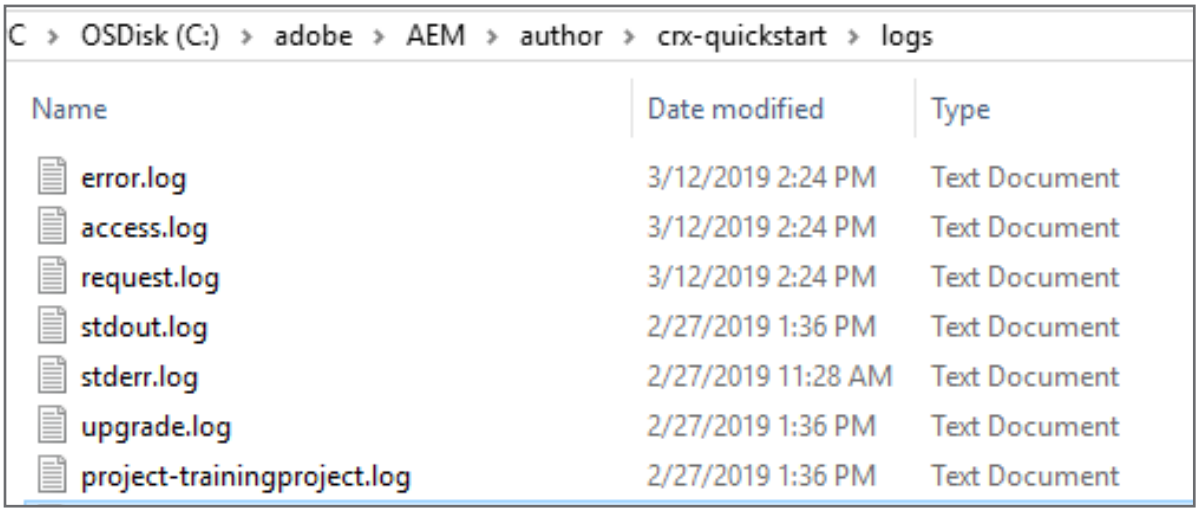
1. Create a new service of the Factory Configuration Apache Sling Logging Logger Configuration.
   a. Specify the log file.
   b. Specify the logger.
   c. Configure the other parameters as required.
2. Create a new service of the Factory Configuration Apache Sling Logging Writer Configuration.
   a. Specify the log file.
   b. Configure other parameters as required.

Exercise 1: Observe the available logs

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 service folder (**C:\adobe\AEM\author**), as shown:

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



Note: The **project-training.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.

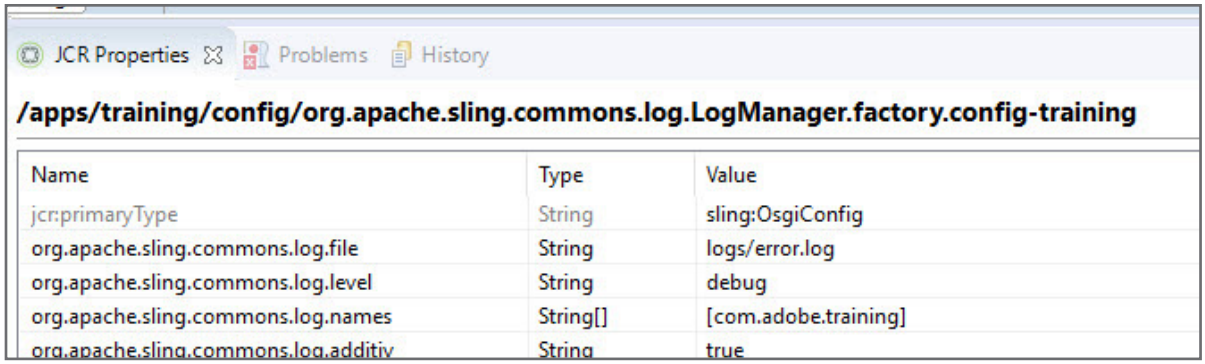
Exercise 2: Use local custom logs

In this exercise, you will observe different ways to create custom log files for local development. These custom logs can be extremely useful for debugging, testing locally, and verifying functionality of your code. These custom log files will only work for local development. In AEM as a Cloud Service, logs will be written into the error.log file.

1. Launch **Eclipse** by double-clicking the shortcut for Eclipse on your desktop and opening your workspace. The **training** project opens in the Eclipse Development Environment.
2. When the **Eclipse IDE Launcher** window opens, keep the default workspace as-is and click **Launch**.
3. In the Eclipse Project Explorer, navigate to **training.ui.apps** > **src/main/content/jcr_root** > **apps** > **training/config**. You will see the file org.**apache.sling.commons.log.LogManager.factory.config-training,** as shown:

4. Double-click the **org.apache.sling.commons.log.LogManager.factory.config-training** file. The file opens in the Eclipse text editor.
5. Click the **JCR Properties** tab below the file. The **Properties** window opens.

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



7. The logger that comes with the archetype points to the **error.log**. To create a local log that filters this project, double click on the property in the JCR Properties window to update.

8. Update the **org.apache.sling.commons.log.file** to logs/**project-training.log**.

To push this change to **AEM**:

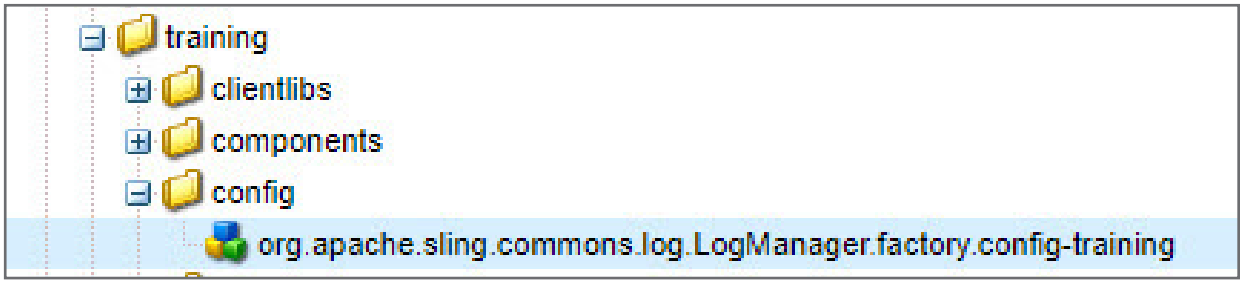
9. In Project Explorer, right-click **training.ui.apps** and select **Run As** > **Maven install**. The build starts.

10. Verify that the bundle is installed successfully,

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

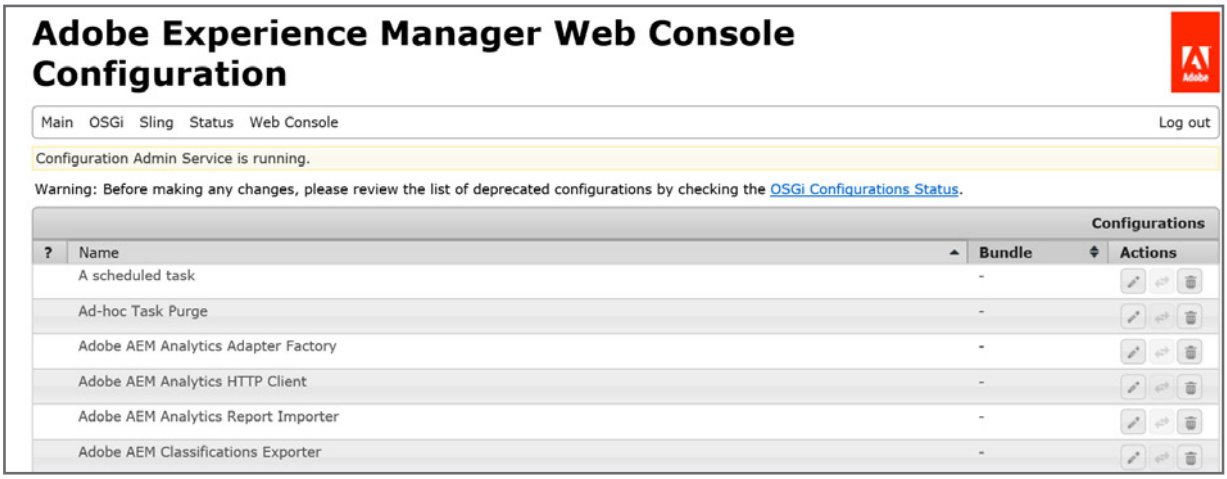
12. Click **Adobe Experience Manager** at the top left and navigate to **Tools** > **General** > **CRXDE Lite**. The **CRXDE Lite** screen opens.

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

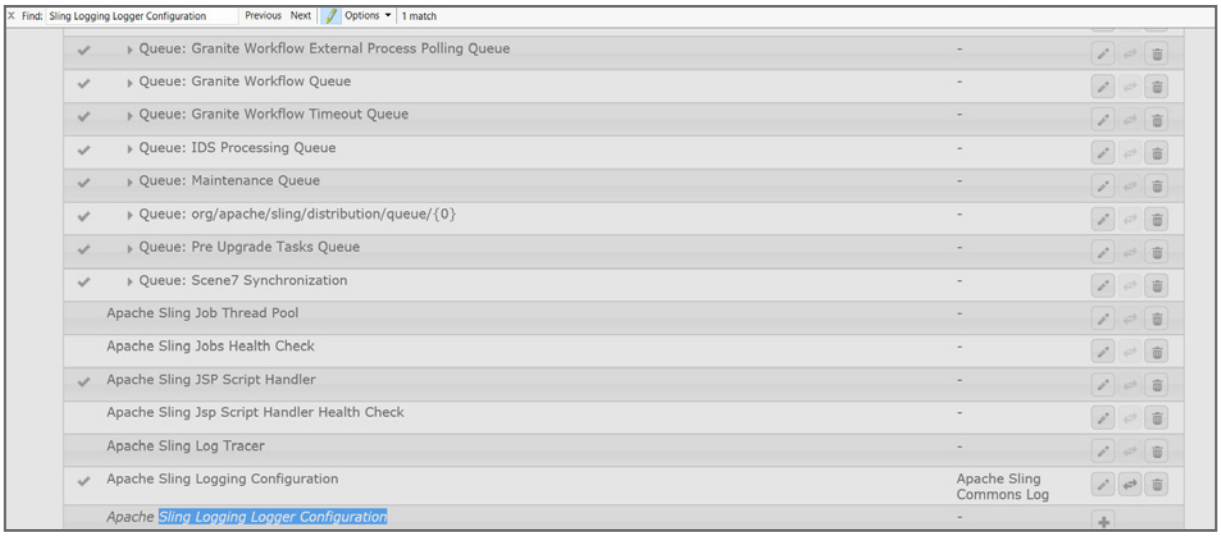


14. Double-click **org.apache.sling.commons.log.LogManager.factory.config-training**. The **Properties** window opens. Observe the values for the log properties.

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



16. Use your browser search (**Ctrl+F)** to search for **Apache Sling Logging Logger Configuration**. The Apache Sling Logging Logger Configuration is listed, as shown:



17. Under **Apache Sling Logging Logger Configuration**, find the configuration called, **logs/project-training.log: info**.
18. Click **Apache Sling Logging Logger Configuration** to open it. The logs for Apache Sling Logging Logger Configuration opens.
19. Click **X** in the upper right to close the Apache Sling Logging Logger Configuration window.
20. Under Apache Sling Logging Logger Configuration, click **logs/project-training.log: info** and verify the properties are the same as the configuration node.

Logging for AEM as a Cloud service

For local development, log entries are written to local files under **./crx-quickstart/logs.**

On Cloud environments, developers can download logs through Cloud Manager or use a command line tool to tail the logs.

Note: The Custom logs are not supported, and so all custom logs are output to the error log.

To access logs for AEM as a Cloud service, you need to perform the following steps:

1. Log into Cloud Manager
2. Open the necessary application.
3. Find the appropriate environment (dev/stage/prod) and select the ellipsis (...)
4. Select **Download Logs**
5. Specify the service and day for the logs you want to view.
6. Download the logs needed.

To learn more about accessing logs within Cloud Manager , go to:
https://docs.adobe.com/help/en/experience-manager-cloud-service/implementing/using-cloud-manager/manage-logs.html

Dive into OSGi

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 the dynamic application is comprised of reusable components.

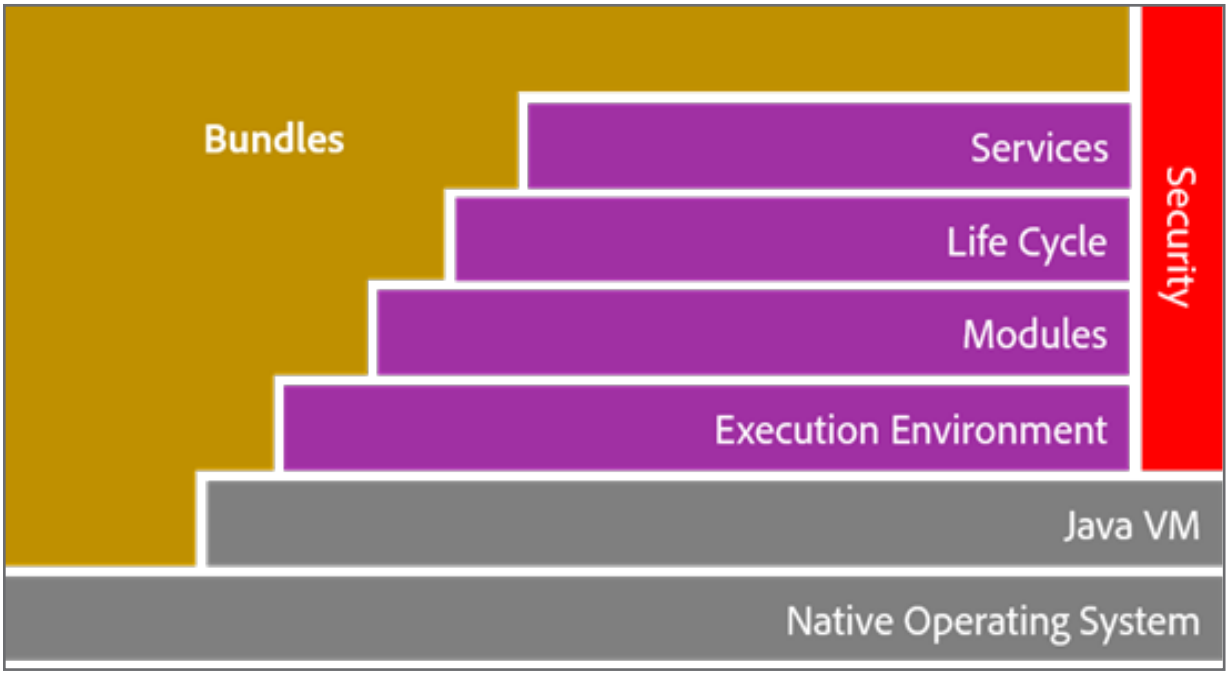
Objectives

After completing this course, you will be able to:

- Describe OSGi architecture
- Define OSGi annotations
- Implement OSGi configurations

OSGi Architecture

The OSGi has a layered model, as shown:



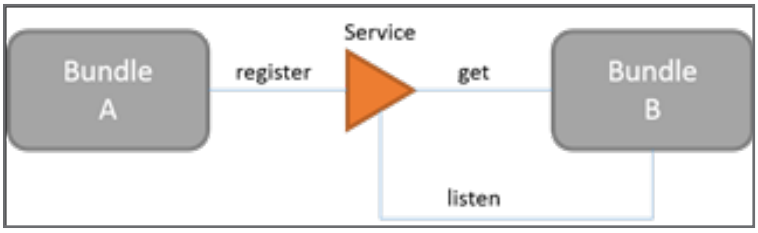
- Bundles: OSGi components created by developers.
- Services: Connect bundles in a dynamic way by offering a publish-find-bind model for plain old Java objects.
- Life Cycle: API that installs, starts, stops, updates, and uninstalls bundles.
- Modules: Define how a bundle can import and export code.
- Security: Handles the security aspects.
- Execution Environment: Defines the methods and classes that are available in a specific platform.

To gain an in-depth understanding of the OSGi architecture, visit:
https://www.osgi.org/developer/architecture/

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 interface 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 process is shown in the following figure:

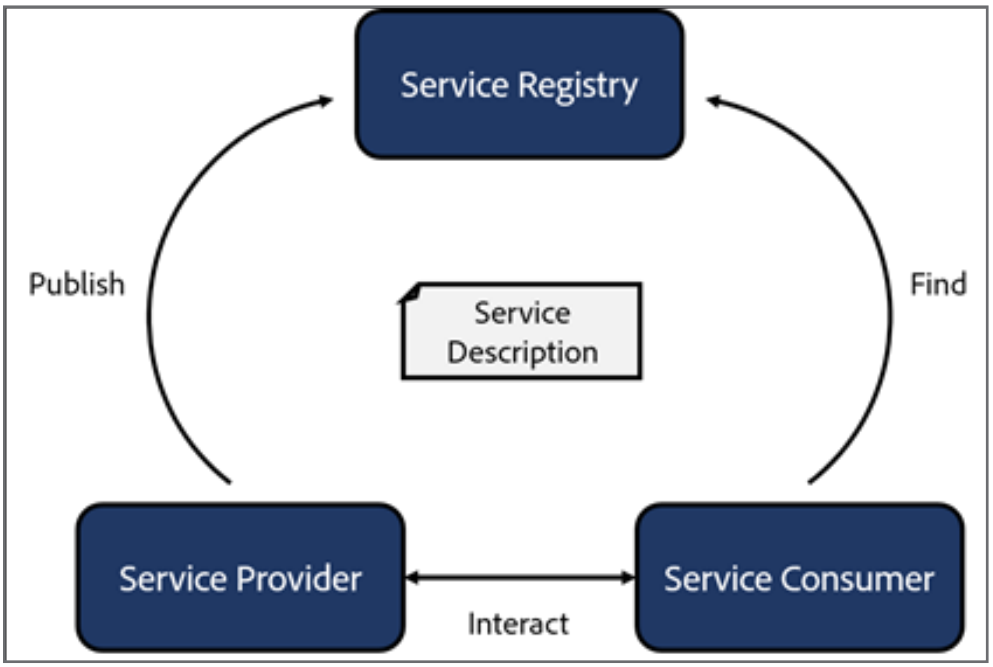


Each service registration has a set of standard and custom properties. An expressive filter language is available to select only the services in which you are interested. You can use properties to find the proper service or they can play other roles at the application level.

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

Service Registry Model

OSGi provides a service-oriented component model 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 lookup using the Whiteboard registry pattern, as shown:



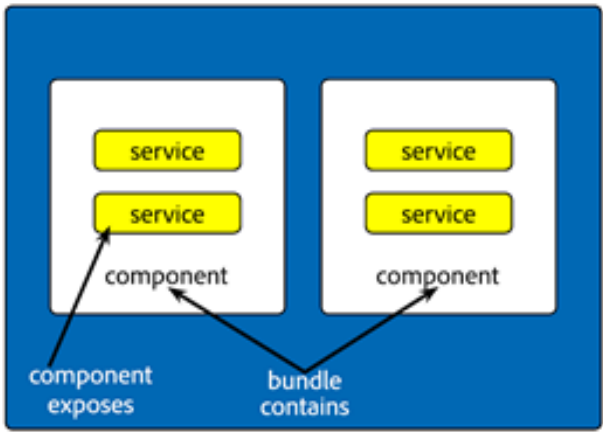
Deployment of Bundles

Bundles are deployed on an OSGi framework—the bundle runtime environment. It is a collaborative environment, where the bundles run in the same Virtual Machine (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 involve 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.

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 an 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.

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

Annotations in OSGi

With Declarative Services, OSGi components are developed using annotations to generate bundle descriptors as well as metatype description for their configuration. Since AEM 6.2, the official OSGi R6 Declarative Services Annotations are supported and Adobe recommends using those annotations as they are defined as a standard and allow for simpler and cleaner code.

Projects based on the previously recommended Felix SCR annotations (now in maintenance mode) can be easily migrated and both annotations styles can coexist within a bundle during the migration phase.

The following annotations are supported:

- Component
- Activate
- Deactivate
- Modified
- Reference

@Component

The @Component annotation enables the OSGi Declarative Services to register your component. This is the only required annotation for an OSGi 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.osgi.service.component.annotations.Component;

@Component
public class MyComponent {
```

Component Modifiers

With OSGi DS annotations, type and property definitions are not done using annotations, as they are with Felix SCR annotations, but through component modifiers (or attributes).
In fact, what we are doing when creating a component is to register the type of service (by implementing one or several interfaces) and to declare the properties made available by metatype generation, all this with the same annotation.
Some of the attributes you can use with the @Component annotation are:

- service: Types under which to register this component as a service
- properties: Property entries for this component
- name: Name of the component
- immediate: Indicates whether the component is immediately activated on bundle start

Example:

```
@Component (service=WorkflowProcess.class,
        name=”My Custom Workflow",
        property={“description=Workflow process to set approval status”,
        “process.label=Approval Status Writer”})
```

When creating a Sling servlet, the servlet properties are defined in the property component modifier like in this example:

```
@Component (service=Servlet.class,
        name=”My Custom Sling Servlet",
        property={“sling.servlet.extensions=html”,
        “sling.servlet.selectors=foo”,
        “sling.servlet.paths=/bin/foo”,
        “sling.servlet.methods=get”,
        “sling.servlet.resourceTypes=project/components/mycomponent”,
    })
```

@Activate, @Deactivate, and @Modified

Methods annotated with these annotations specify what happens when the component is activated, deactivated, or its configuration is modified.

```
import org.osgi.service.component.annotations.Activate;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Deactivate;
import org.osgi.service.component.annotations.Modified;
@Component
public class MyComponent{
@Activate
    protected void activate() {
        // do something
    }
@Deactivate
    protected void deactivate() {
        // do something
    }
@Modified
    protected void readConfig() {
        // get values
    }
}
```

Configurable Services

The OSGi Configuration Admin Service enables components to get or retrieve a configuration, and provides the entry point for management agents to retrieve and update configuration data. 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.

Unlike SCR annotations, for which properties need to be read by overriding the activate() method, OSGi DS annotations provide a mechanism for metatype declaration through an interface, avoiding reading and converting each property. This interface is annotated with @ObjectClassDefinition and defines each property with its type, using the annotations @AttributeDefinition and @AttributeType.

Here is an example:

```
import org.osgi.service.metatype.annotations.ObjectClassDefinition;
import org.osgi.service.metatype.annotations.AttributeDefinition;
import org.osgi.service.metatype.annotations.AttributeType;
@ObjectClassDefinition(name = My Configuration Service)
public @interface MyConfigInterface{
   @AttributeDefinition(
name="Enter a String",
description="Your description",
type=AttributeType.STRING
)
    String myconfig_property() default "";
}
```

This interface is then defined in the component using the annotation @Designate. A modifier named factory can be added to define the configuration as a factory.

```
import org.osgi.service.component.annontations.Component;
import org.osgi.service.metatype.annotations.Designate;
@Component
@Designate(ocd=MyConfigInterface.class, factory=true)
public class MyComponent {
   private myString
  @Activate
   protected void activate(MyConfigInterface config) {
           myString = config.myconfig_property()
   }
}
```

Exercise 1: Create and use a custom service

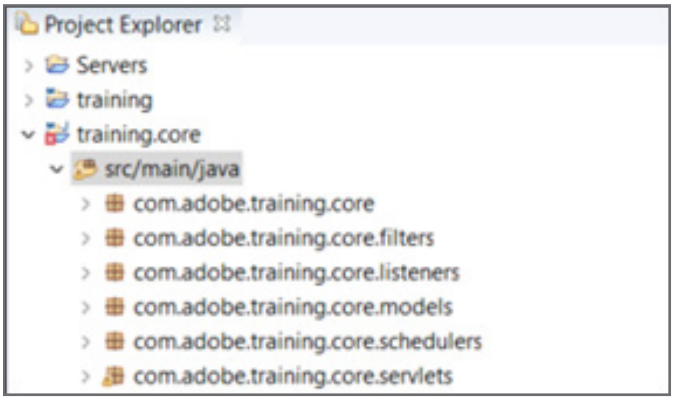
In this exercise, you will create and use a custom service.

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

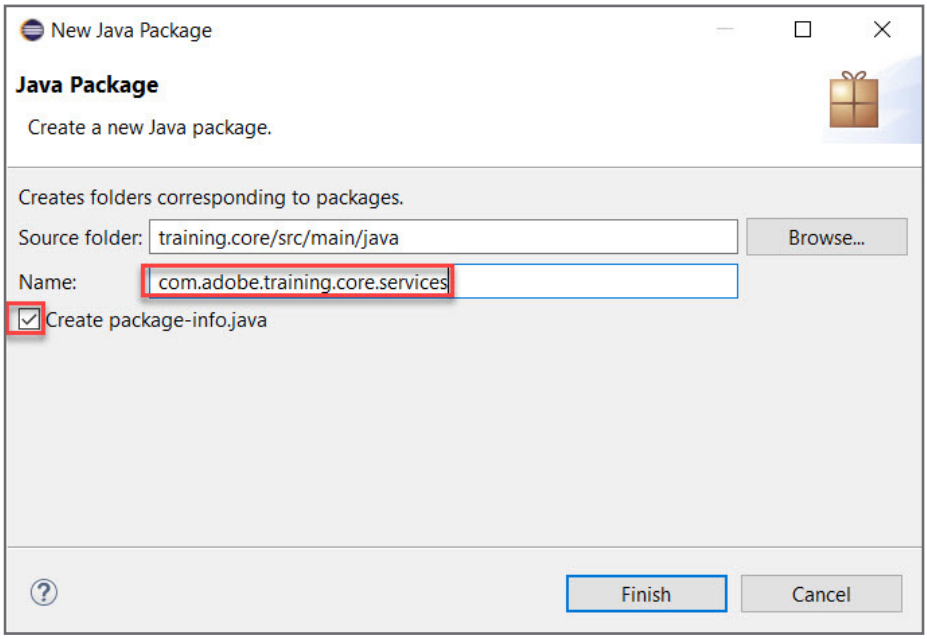
1. If not already open, open **Eclipse**.
2. In Project Explorer, navigate to **training.core** > **src/main/java**, as shown:



3. Right-click **com.adobe.training.core** and choose **New** > **Package.** The **New Java Package** window appears.

Note: If you do not see **com.adobe.training.core**, you need to click the Filter icon on the Project Explorer tab and, in the **Filters and Customizations** window, uncheck the **Empty parent packages** checkbox.

4. Keep the **Source folder** value as-is and type the value for the **Name** field, as shown:



› Name**: com.adobe.training.core.services**

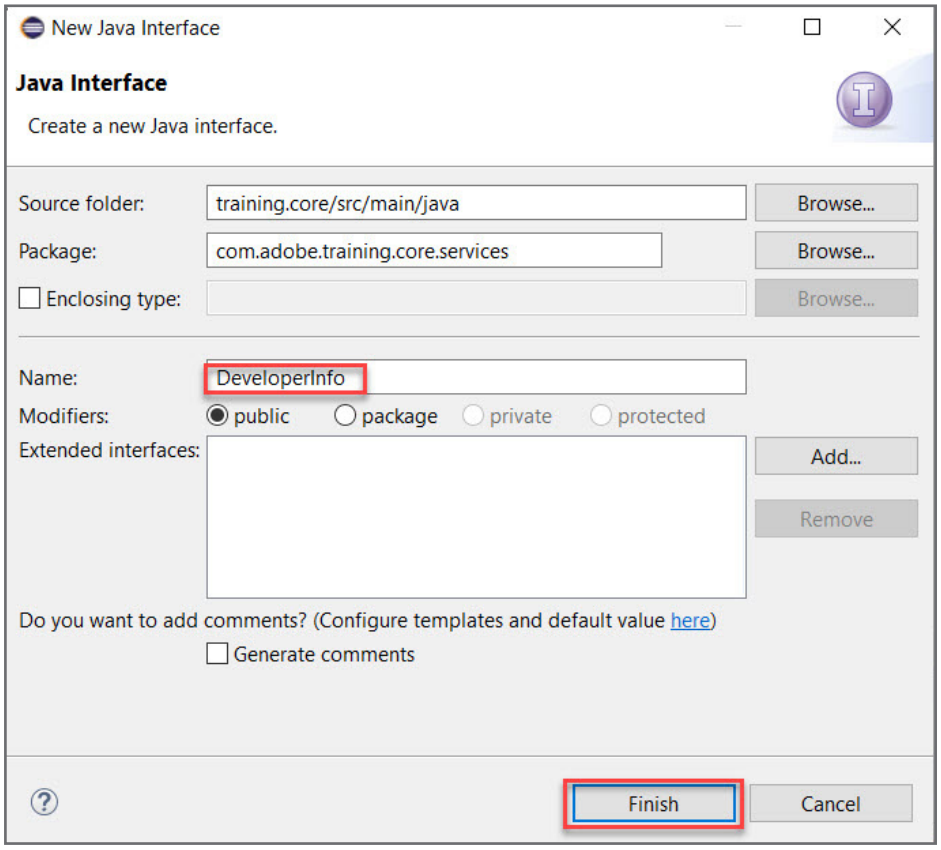
5. Select the **Create Package-info.java** checkbox and click **Finish**. The **Package-info.java opens** in the **Java Editor.**

Note: Selecting the **Create Package-info.java** allows for this package to be visible in OSGI.

6. Click **Finish**. Eclipse opens its text editor.
7. Copy the contents of the file **Package-info.java** from **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/services/package-info.java**.
8. Verify the completed file, as shown:

```
1  /**
2   *
3   */
4  /**
5   * @author prpurush
6   *
7   */
8  @Version("1.0")
9  package com.adobe.training.core.services;
10
11 import org.osgi.annotation.versioning.Version;
12
```

9. Click **Ctrl+S** or click **File** > **Save** to save your changes to **package-info.java** in the Eclipse editor.
10. Right-click **com.adobe.training.core.services** and choose **New** > **Interface**. The **New Java Interface** window appears.
11. Type the name as **DeveloperInfo** and click **Finish,** as shown:



12. Copy the contents of the file **DeveloperInfo.java** from **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/services/DeveloperInfo.java**
13. In Eclipse, double-click **DeveloperInfo.java** to open it in the Java Editor and replace the contents of the file with the copied content.

14. Examine the service **DeveloperInfo.java** interface, as shown:

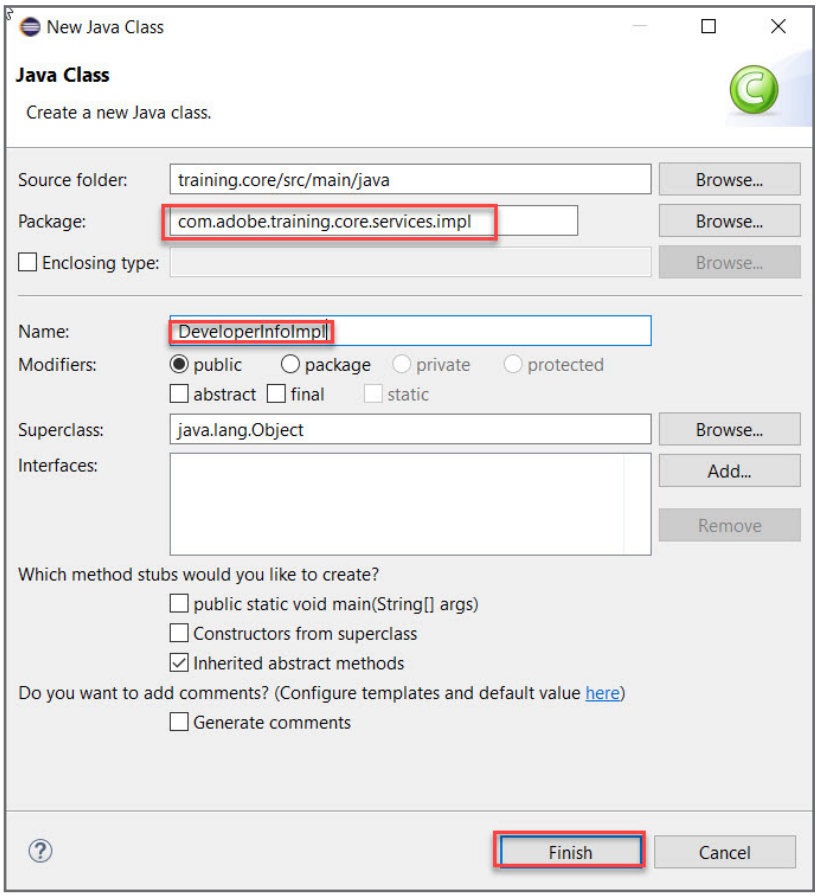
```
1.  package com.adobe.training.core.services;
2.  /**
3.   * Service interface to get the information about the bundle Developer
4.   *
5.   * Example code can be inserted into the Helloworld HTL component:
6.   * /apps/training/components/helloworld/helloworld.html
7.   *
8.   * Example HTL:
9.   * <div data-sly-use.devInfo=»com.adobe.training.core.DeveloperInfo»>Developer Info: ${devInfo.DeveloperInfo}</div>
10.  *
11.  *
12.  */
13.  public interface DeveloperInfo {
14.     public String getDeveloperInfo();

15.  }
```

15. Save (**Ctrl+S** in Windows) your changes.

Note: The code is provided as part of the **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/services/**. Please DO NOT copy the code from this exercise book. This code is for illustrative purposes only.

16. Right-click **com.adobe.training.core.services** and select **New** > **Class.** The **New Java Class** window appears.

17. Type the values, as shown and click **Finish**.
    - Package: **com.adobe.training.core.services.impl**
    - Name: **DeveloperInfoImpl**



18. Copy the contents of the file **DeveloperInfoImpl.simple.txt** from **Exercise_Files** under **\core\src\main\java\com\adobe\training\core\services\impl**.

19. In Eclipse, if not already open, double-click **DeveloperInfoImpl.java** to open it in Java Editor and replace the contents of the file with the copied content.

20. 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 java.util.Map;
4.
5.  import org.osgi.service.component.annotations.Activate;
6.  import org.osgi.service.component.annotations.Component;
7.  import org.osgi.service.component.annotations.Deactivate;
8.  import org.osgi.service.component.annotations.Modified;
9.  import org.slf4j.Logger;
10. import org.slf4j.LoggerFactory;
11.
12. import com.adobe.training.core.services.DeveloperInfo;
13.
14. /**
15.  * Component implementation of the DeveloperInfo Service.
16.  */
17.
18. @Component(service = DeveloperInfo.class,
19.         immediate = true)
20. public class DeveloperInfoImpl implements DeveloperInfo {
21.     private final Logger logger = LoggerFactory.getLogger(getClass());
22.
23.     @Activate
24.     @Modified
25.     protected void activate(Map<String, Object> config) {
26.         logger.info("##############Component config saved");
27.     }
28.
29.     @Deactivate
30.     protected void deactivate() {
31.         logger.info("##############Component (Deactivated) Good-bye ");
32.     }
33.
34.
35.     // Method used to show a simple OSGi service/component relationship
36.     public String getDeveloperInfo()
    { return "Hello! I do not know who my developer is. I am a product of random development!!!";
37.     }
38.

39. }
```

21. Examine the method **getDeveloperInfo()** in the DeveloperInfoImpl.java class.

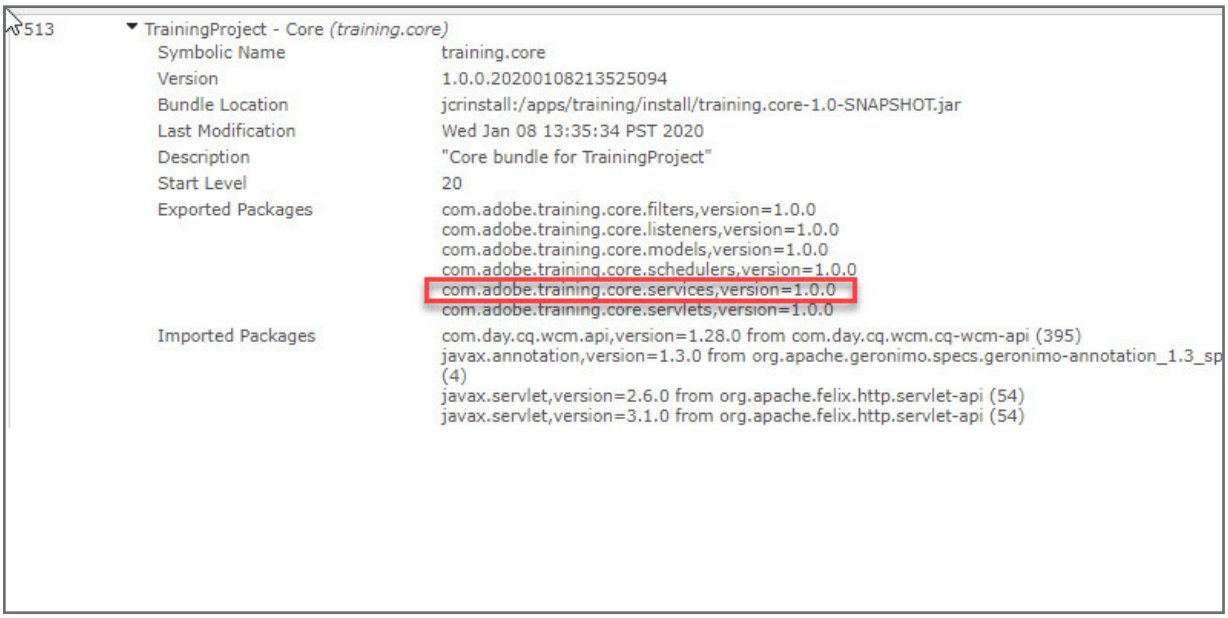
22. Save (**Ctrl+S** in Windows) your changes.

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:

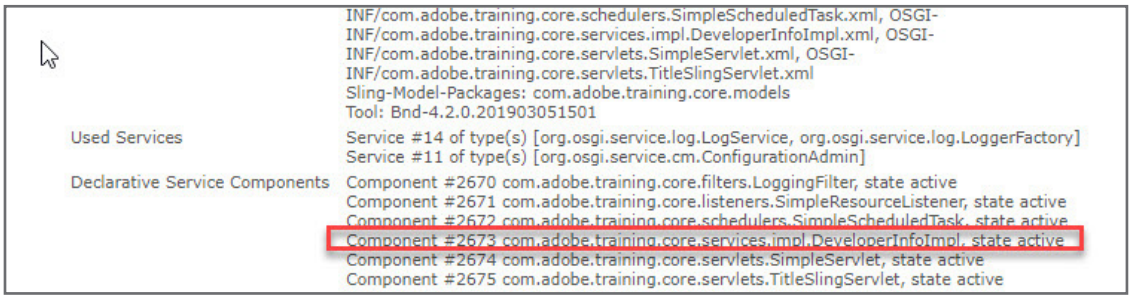
```
[INFO]
[INFO] --- maven-bundle-plugin:3.3.0:bundle (default-bundle) @ training.core ---
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ training.core ---
[INFO] Installing C:\Users\prpurush\Dev1\training\core\target\training.core-0.0.1-SNAPSHOT.jar to C:\Users\prpurush\.m2\repository\com\adobe\training.core\0.0.1-SNAPSHO
[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.p
[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/
[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. Search for and click **TrainingProject – Core** (**training.core**) to expand it. Notice your core services package has been exported, as shown:



5. Scroll down and look for the Service, **com.adobe.training.core.services.impl.DeveloperInfoImpl**.

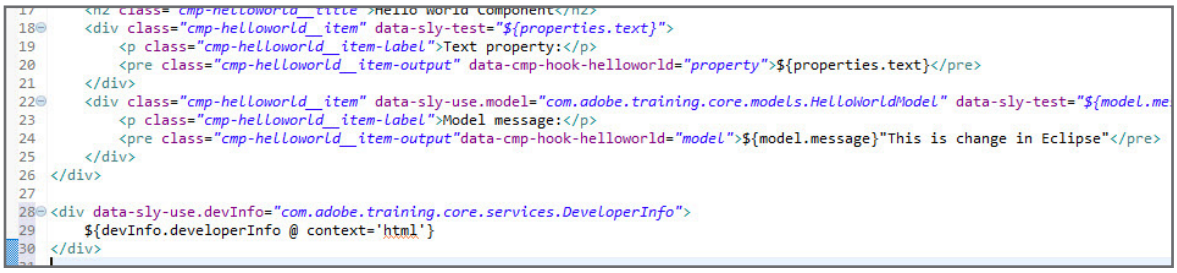
6. Verify that the service exists in the bundles, as shown:



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

7. In Eclipse, under Project Explorer, navigate to **training.ui.apps** > **src/main/content/jcr_root** > **apps** > **training** > **components** > **helloworld [cq:Component]**.
8. Expand **helloworld.html [cq:Component]** and double-click **helloworld.html**. The file opens in the XML Editor.
9. Add the following line to the **helloworld.html** page:

```
<div data-sly-use.devInfo="com.adobe.training.core.services.
DeveloperInfo">
   ${devInfo.developerInfo @ context='html'}
</div>
```



Note: This code is also available in the file under /**Exercise_Files/backend-training-dev/ui.apps/src/main/content/jcr_root/apps/training/components/helloworld/ helloworld .html**.

10. Save the changes to the file.

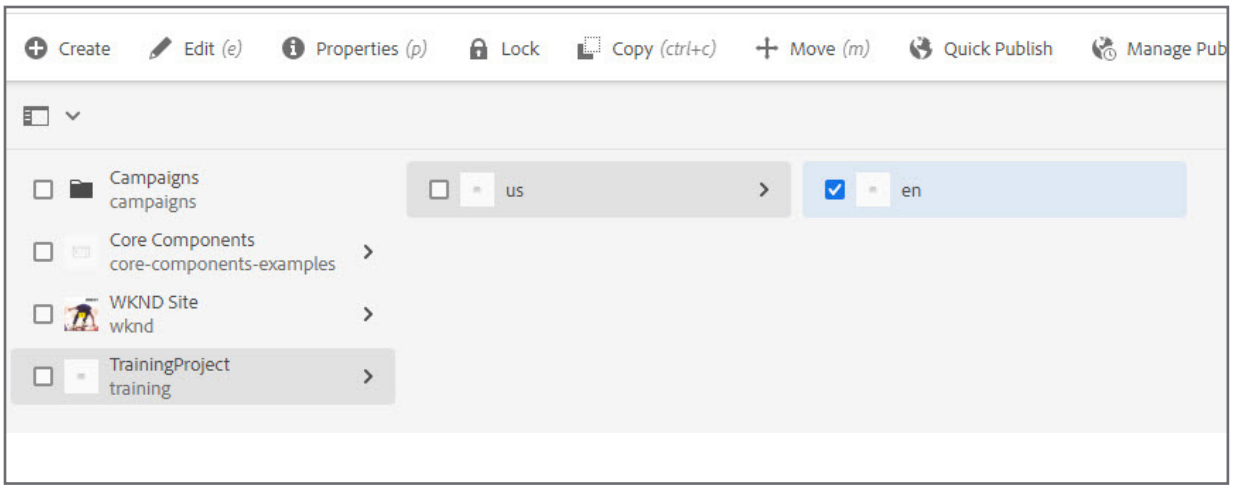
Note: With HTL, you can consume an OSGi service. Here, you are consuming the **DeveloperInfo** service. After the service is called, you can use the method from the service.

Note: The Eclipse AEM server will automatically sync the HTL file after it is saved. Make sure you have your Eclipse AEM server started; otherwise, the file will not sync with the JCR.

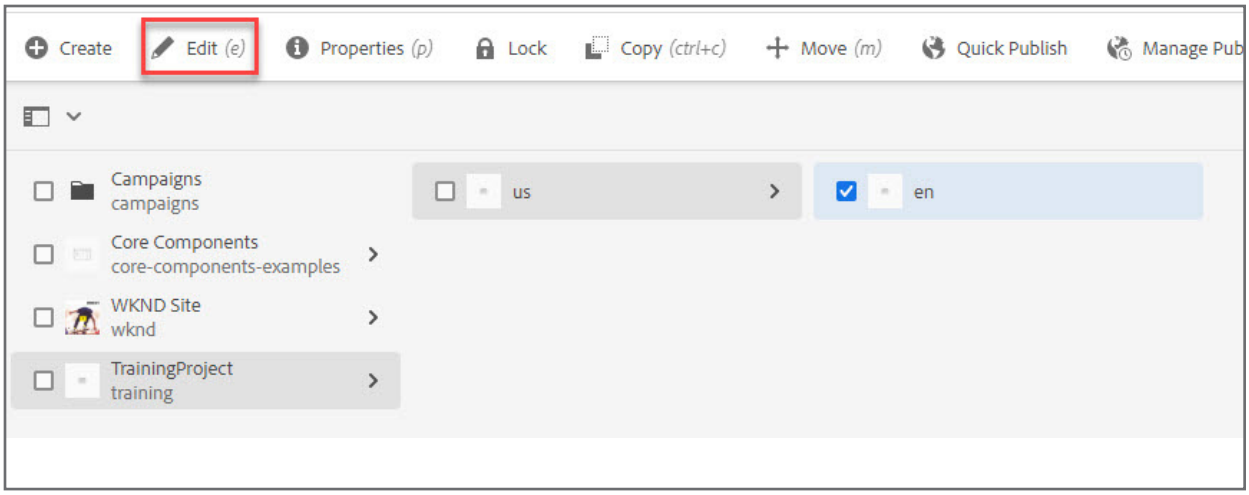
11. To verify whether your file successfully saved to the JCR, open **CRXDE Lite** to **/apps/training/components/helloworld/ helloworld.html** and see if that file contains your changes.
12. If your changes are missing, you can force an update with a full redeploy to the JCR by going back to Eclipse and right-clicking **training.ui.apps** and selecting **Run AS** > **Maven Install.**
13. Verify the bundle is installed successfully.

Task 3: Test the service

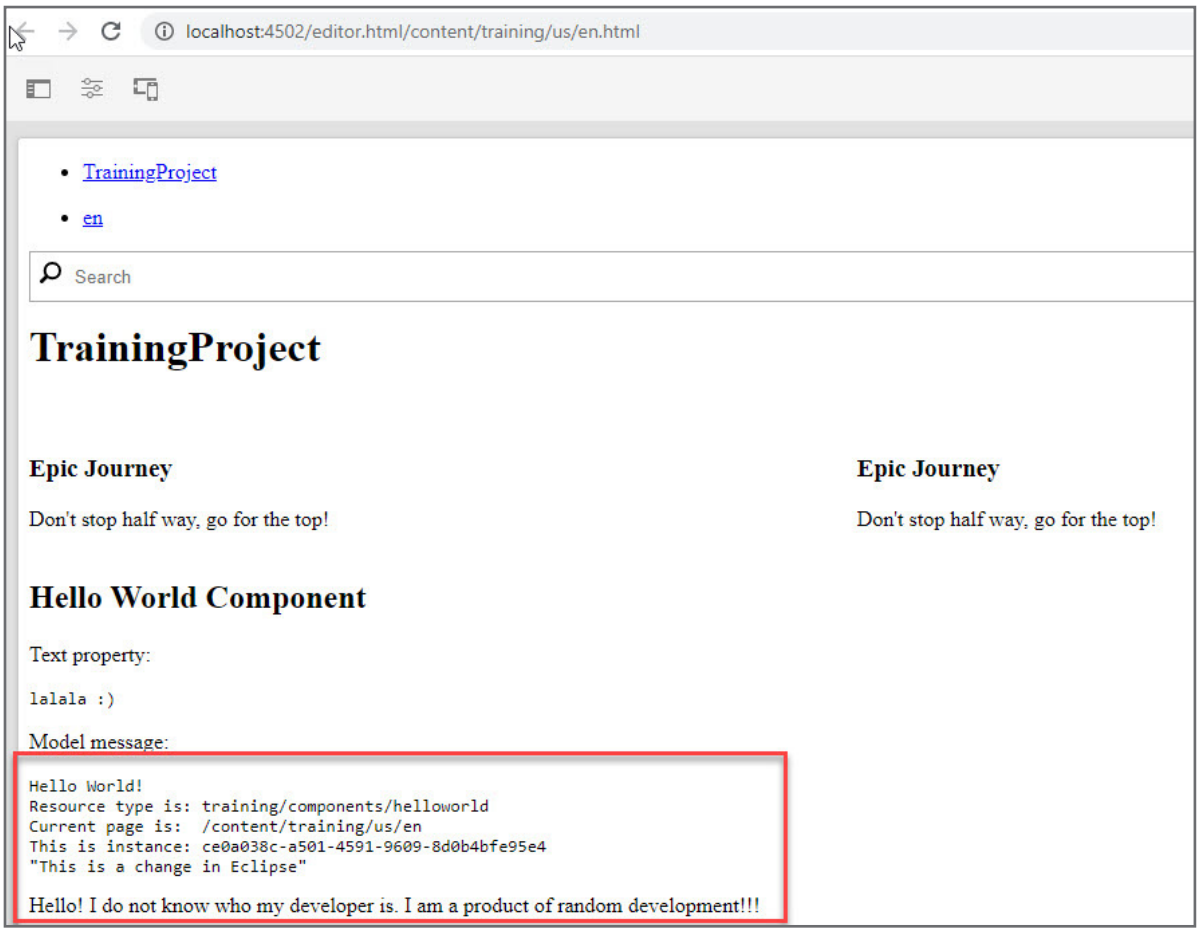
1. Log in to AEM and click **Adobe Experience Manager** in the upper left.
2. Click **Navigation** > **Sites**. The Sites console opens.
3. Select **TrainingProject** > **us** > **en**, as shown:



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



5. Verify you see the following message, as shown:



6. If the **Hello World component** is not on the page, perform the following steps.
7. Click on the Components icon on the page.

8. Scroll down to find and then drag the **HelloWorld Component** to the **Drag components here** area on your page.

Exercise 2: 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 as we did earlier. These configurations can then be further targeted 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**.
2. Double-click **DeveloperInfoImpl.java** to edit the file.
3. Copy the contents from **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/services/impl/DeveloperInfoImpl.java** to **DeveloperInfoimpl.java** in Eclipse, as shown:

```
1.  package com.adobe.training.core.services.impl;
2.  import org.osgi.service.component.annotations.Activate;
3.  import org.osgi.service.component.annotations.Component;
4.  import org.osgi.service.component.annotations.Deactivate;
5.  import org.osgi.service.component.annotations.Modified;
6.  import org.osgi.service.metatype.annotations.Designate;
7.  import org.slf4j.Logger;
8.  import org.slf4j.LoggerFactory;
9.  import java.util.Arrays;
10. import com.adobe.training.core.DeveloperInfo;
11. import com.adobe.training.core.services.DeveloperInfoConfiguration;
12.  /**component implementation
13.  of the DeveloperInfo Service. This gets the developer info from the OSGi Configuration
14.  * There are 4 OSGi Configuration Examples:
```

```
 * -Boolean
 * -String
 * -String Array
 * -Dropdown
 */

@Component(service = DeveloperInfo.class, immediate = true)

@Designate(ocd = DeveloperInfoConfiguration.class)
public class DeveloperInfoImpl implements DeveloperInfo {
    private final Logger logger = LoggerFactory.getLogger(getClass());

    //local variables to hold OSGi config values
    private boolean showDeveloper;
    private String developerName;
    private String[] developerHobbiesList;
    private String langPreference;

    @Activate
    @Modified

    //http://blogs.adobe.com/experiencedelivers/experience-management/osgi_activate_
deactivatesignatures/

    protected void activate(DeveloperInfoConfiguration config) {

        showDeveloper = config.developerinfo_showinfo();
        developerName = config.developerinfo_name();
        developerHobbiesList = config.developerinfo_hobbies();
        langPreference = config.developerinfo_language();
        logger.info("##############Component config saved");
    }

    @Deactivate
    protected void deactivate() {
        logger.info("##############Component (Deactivated) Good-bye " + developerName);
    }

    /**
     * Method used to show how OSGi configurations can be brought into a OSGi component
    **/
    public String getDeveloperInfo(){

        String developerHobbies = Arrays.toString(developerHobbiesList);

        if(showDeveloper)

            return "Created by " + developerName
                    + ". <br>Hobbies include: " + developerHobbies
                  + ". <br>Preferred programming language in AEM is " + langPreference;
        return "";
    }

    /*
     * Method used to show a simple OSGi service/component relationship
    public String getDeveloperInfo(){ return
«Hello! I do not know who my developer is. I am a product of random development!!!»;
    }
    */

      }
```

Note: Ignore errors that mention Developer Info Configuration. We will fix these errors in a few steps.

4. Examine the modified code.
5. 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.

6. In Eclipse **Project Explorer**, right-click on **training.core** > **scom.adobe.training.core.services**. **impl** and select **New** > **Class**.
7. Type the following value for the name:
   - Name: **DeveloperInfoConfiguration**
8. Click **Finish**.
9. Copy the contents of the file **DeveloperInfoConfiguration .java** from **\core\src\main\java\com\ adobe\ training\core\services\impl** to **DeveloperInfoConfiguration.java** in Eclipse.

Note: The code is provided as part of the **Exercise_Files** under **\core\src\main\java\com\adobe\ training\core\services**. 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 for illustrative purposes only.

---

10. Examine **DeveloperInfoConfiguration.java**, as shown:

```
1.  package com.adobe.training.core.services.impl;
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 = "Training DeveloperInfo Config»)
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(
```

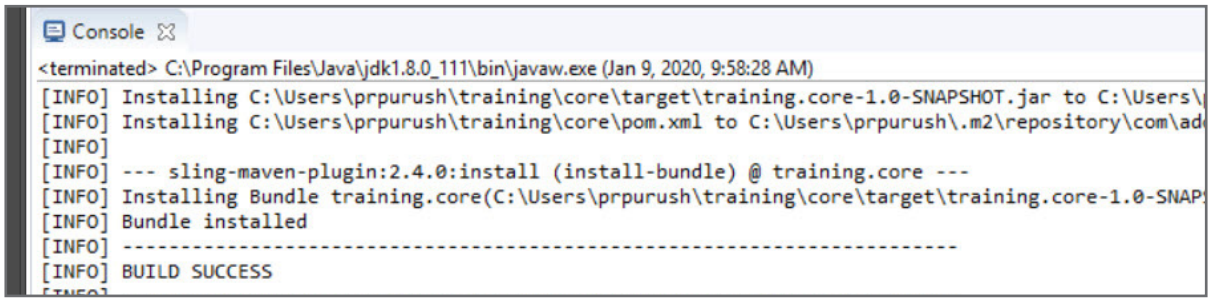
```
33.             name = "Language",
34.             description = "Favorite Language Preference",
35.             options = {
36.                 @Option(label = "HTL", value = "HTL"),
37.                 @Option(label = "Java", value = "Java"),
38.                 @Option(label = "JSP", value = "JSP"),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

11. Save the changes.

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

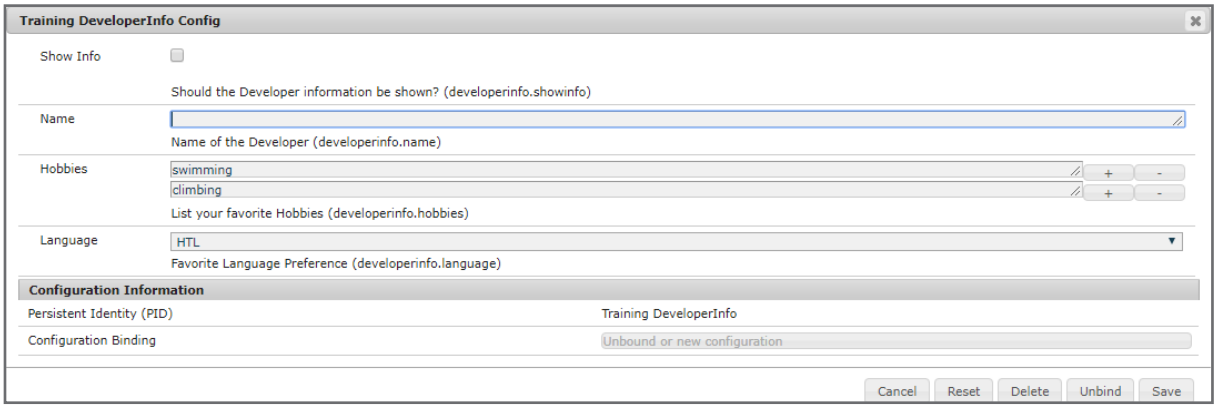
Task 2: Deploy the bundle

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:



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 **DeveloperInfo**.
3. Click **Training DevelopInfo Config**. The **Training DeveloperInfo Config** opens, as shown:



4. Carry out the following steps:
   a. Select the **Show Info** checkbox.
   b. Type **Scott Reynolds** in the **Name** field.
5. Save the changes.
6. Go back to your AEM author instance and navigate to **Sites** (http://localhost:4502/sites.html/content). The **Sites** console opens.
7. Select **TrainingProject** > **us** > **en** > **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 real-life situation, you should create an OSGi config node in the JCR.

References

For more information on OSGi, refer:

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

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

OSGi Components – Simply Simple – Part I

https://blog.osoco.de/2015/08/osgi-components-simply-simple-part-i/

Configuring Sling Web Framework

Introduction

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 uses a Java Content Repository (JCR)
- 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.

Objectives

After completing this course, you will be able to:

- Describe Apache Sling
- Describe the Sling resolution process
- Create Sling servlets
- Create system users

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.

The following are some of the common properties that a resource can have:

- 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 a 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. Map the Request to resources
3. Resolve the Resource
4. Resolve the rendering script/servlet
5. Decompose 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	
http://	myhost	tools/spy	.printable.a4	html	/	a/b	?

The following table describes the components.

Component	**Description**
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

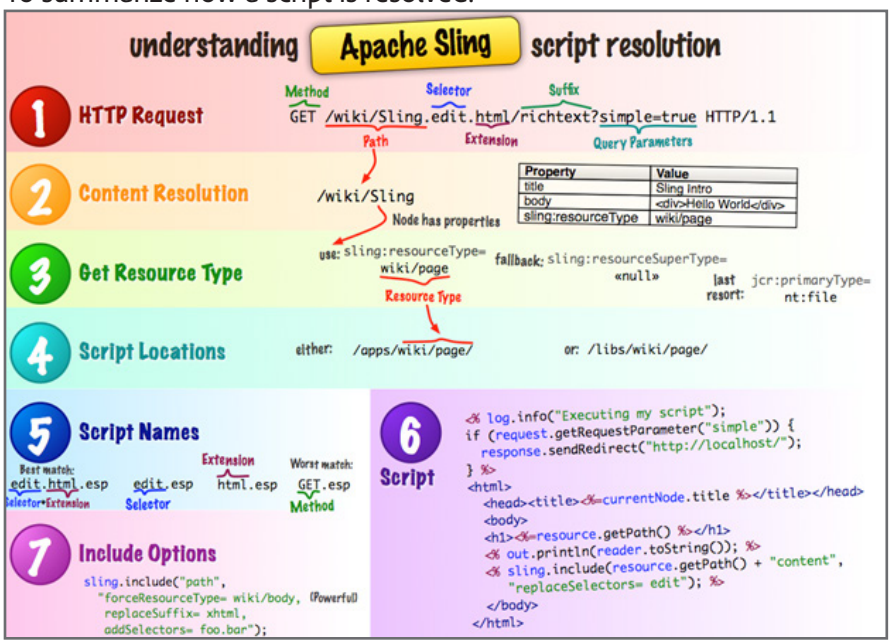
- Sling checks whether a node exists at the location specified in the request.
- In the above example, it searches for the spy.html node.
- If no node is found at that location, the extension is dropped and the search is repeated. For example, it searches for the node spy.
- If no node is found, then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

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:



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, for 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.

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 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.
For a Servlet registered as an OSGi service to be used by the Sling Servlet Resolver, the sling.servlet.resourceTypes service reference property must be set. If it is not set, the Servlet service is ignored.

Each path to be used for registration constructed from the other sling.servlet.* properties - must be absolute. Any relative path is made absolute by prefixing it with a root path. This prefix may be set with the sling.servlet.prefix service registration property. If this property is not set, the first entry in the ResourceResolver search path for the ResourceResolver.getResource(String) method is used as the prefix. If this entry cannot be derived, a simple slash - / - is used as the prefix.
If sling.servlet.methods is not specified, the servlet is only registered for handling GET and HEAD requests. Make sure to list all methods you want to be handled by this servlet.

Exercise 1: Create a Sling servlet

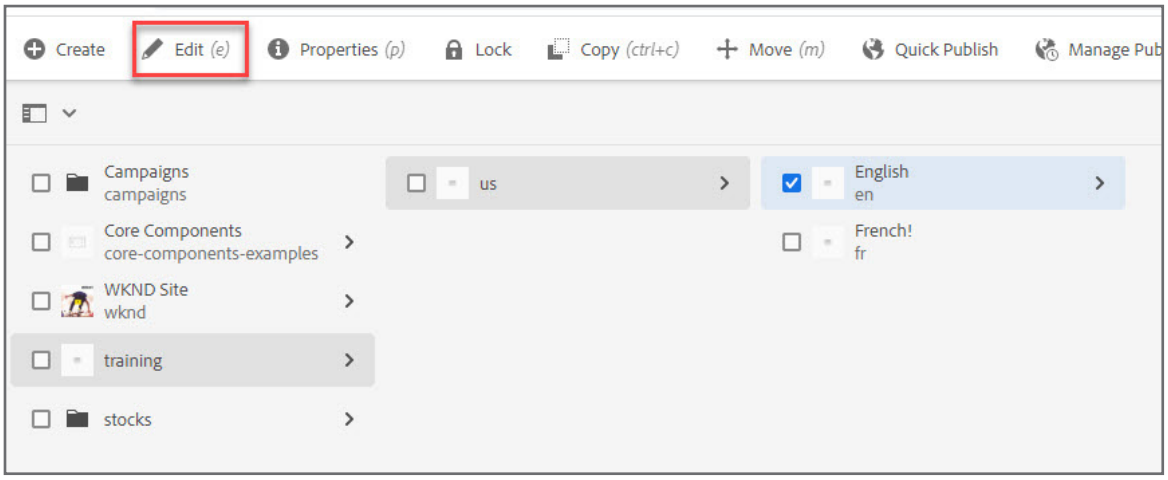
In this exercise, you will write a servlet that serves only doGet requests with a specific selector or resource type. The response shall contain the JCR repository properties as JSON.

This exercise includes two tasks:

1. Create a servlet with a resource type
2. Use selector to trigger the servlet

Task 1: Create a servlet with a resource type

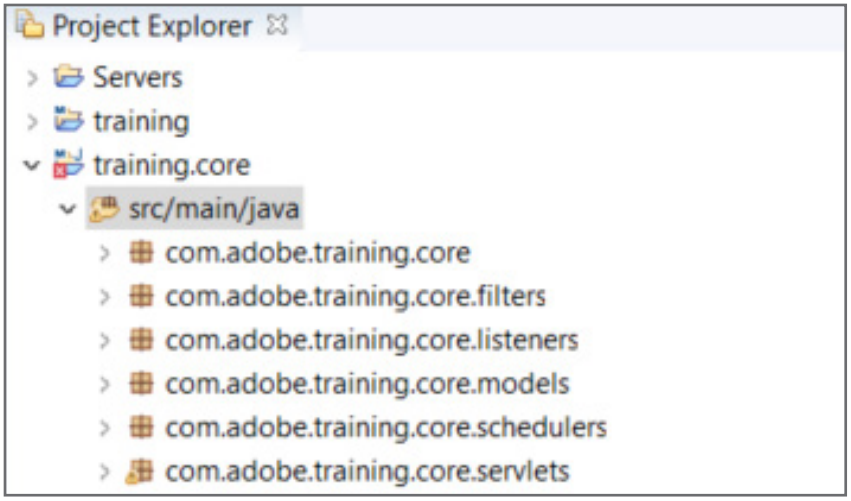
1. In AEM, navigate to **Sites** > **training** >**us** and select the **English** page.
2. Click **Edit**, as shown: The English page opens in edit mode.



3. Click the Components icon.

4. Drag the **Title** component to the **en** Page and add some dummy text ,as shown:

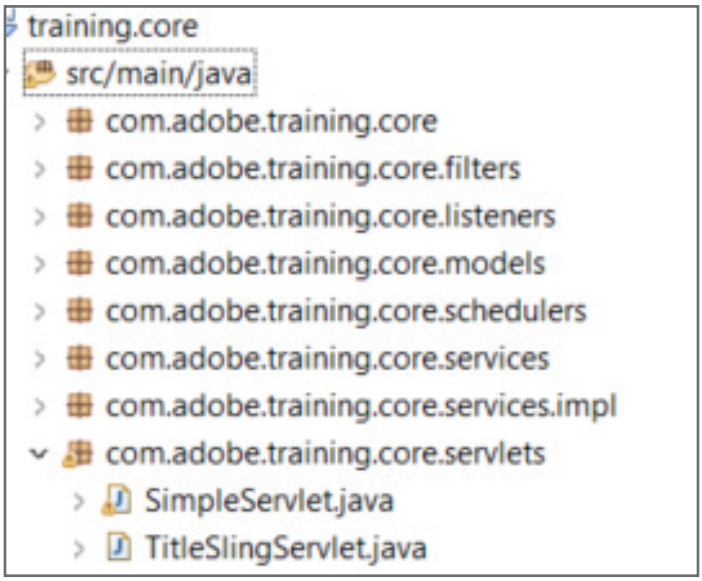
5. Click **Done** to close the **Title** window.
6. Launch **Eclipse** by double-clicking the Eclipse shortcut on the desktop. The Eclipse Launcher opens.
7. Click **Launch**. The Workspace opens.
8. In Project Explorer, navigate to **training.core** > **src/main/java** and expand, as shown:



9. Copy the file **TitleSlingServlet.java** from **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/servlets/**.

Note: The code is provided as part of the **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/servlets/**. 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 for illustrative purposes only.

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



11. Observe that the code, as shown:

```
1.  package com.adobe.training.core.servlets;
2.
3.  import java.io.IOException;
4.  import javax.servlet.Servlet;
5.  import javax.servlet.ServletException;
6.  import org.apache.sling.api.SlingHttpServletRequest;
7.  import org.apache.sling.api.SlingHttpServletResponse;
8.  import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
9.  import org.apache.sling.api.resource.Resource;
10.
11. import org.osgi.service.component.annotations.Component;
12. import org.slf4j.Logger;
13. import org.slf4j.LoggerFactory;
14.
15. import com.day.cq.wcm.api.Page;
16. import com.day.cq.wcm.api.PageManager;
17.
18.
19. /**
20.  * TitleSlingServlet can use resourceType, selector, method, and extension to bind to URLs
21.  *
22.  * Example URL: http://localhost:4502/content/training/us/en.html
23.  * Example URL with selector: http://localhost:4502/content/training/us/en.foobar.html
24.  *
25.  */
26. @Component( service = Servlet.class,
27.            property = {
28.               "sling.servlet.resourceTypes="+TitleSlingServlet.RESOURCE_TYPE,
29. //              "sling.servlet.selectors=foobar",
30.               "sling.servlet.extensions=html"
31.            })
32. public class TitleSlingServlet extends SlingSafeMethodsServlet {
33.    private static final long serialVersionUID = 1L;
34.
35.    protected static final String RESOURCE_TYPE = "training/components/title";
36.
37.    @Override
38. protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException {
39.        //Get the resourceResolver from the request and adapt it to the PageManager
40.        PageManager pm  = request.getResourceResolver().adaptTo(PageManager.class);
41.        //Use the PageManager to find the containing page of the resource (component)
42.        Page curPage = pm.getContainingPage(request.getResource());
```

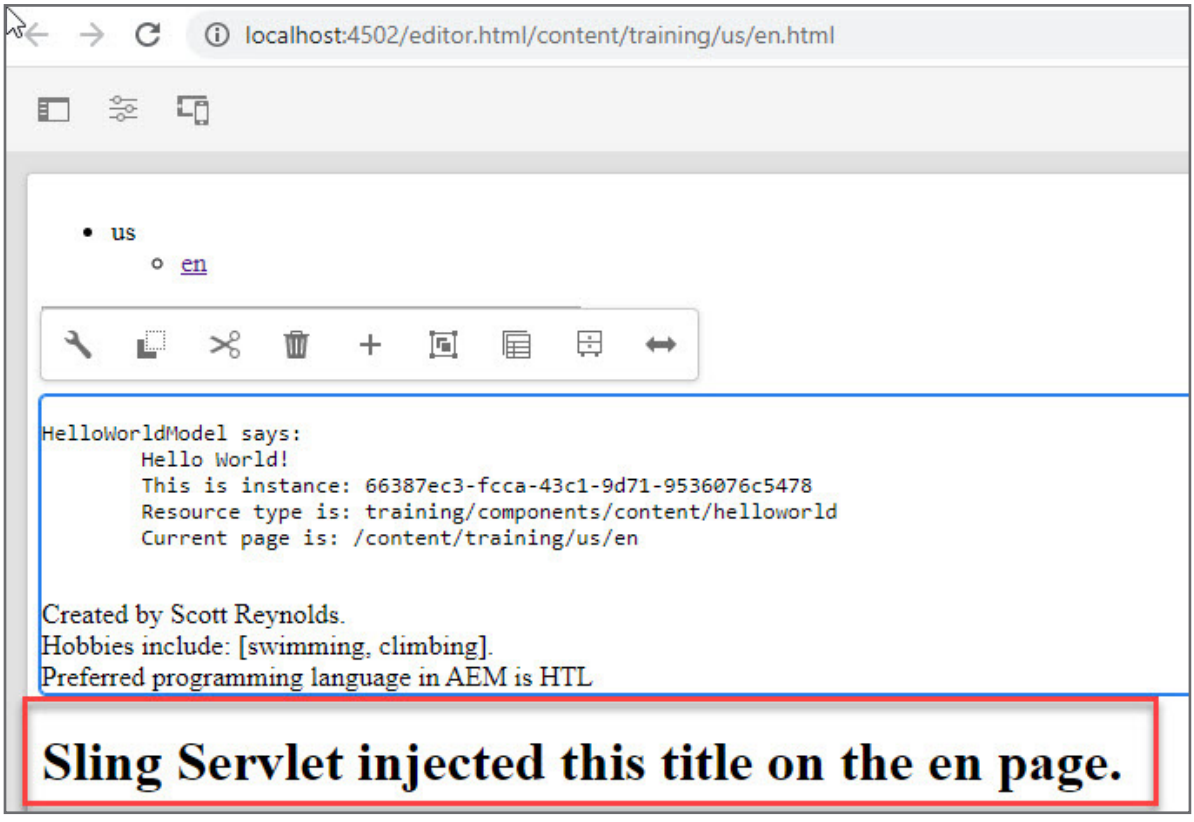
```
43.
44.         //Verify the page exists and it is a site page and not an XF
45.         if(curPage != null && !curPage.getName().equals("master")) {
46.             String responseStr = "";
47.             response.setHeader("Content-Type", "text/html");
48.             responseStr = "<h1>Sling Servlet injected this title on the " + curPage.getName() + " page.</h1>";
49.             response.getWriter().print(responseStr);
50.             response.getWriter().close();
51.         }
52.     }
53. }
```

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] ------------------------------------------------------------------------
```

12. Open http://localhost:4502/content/training/us/en.html and see the output, as shown:



Task 2: Use selector to trigger the servlet

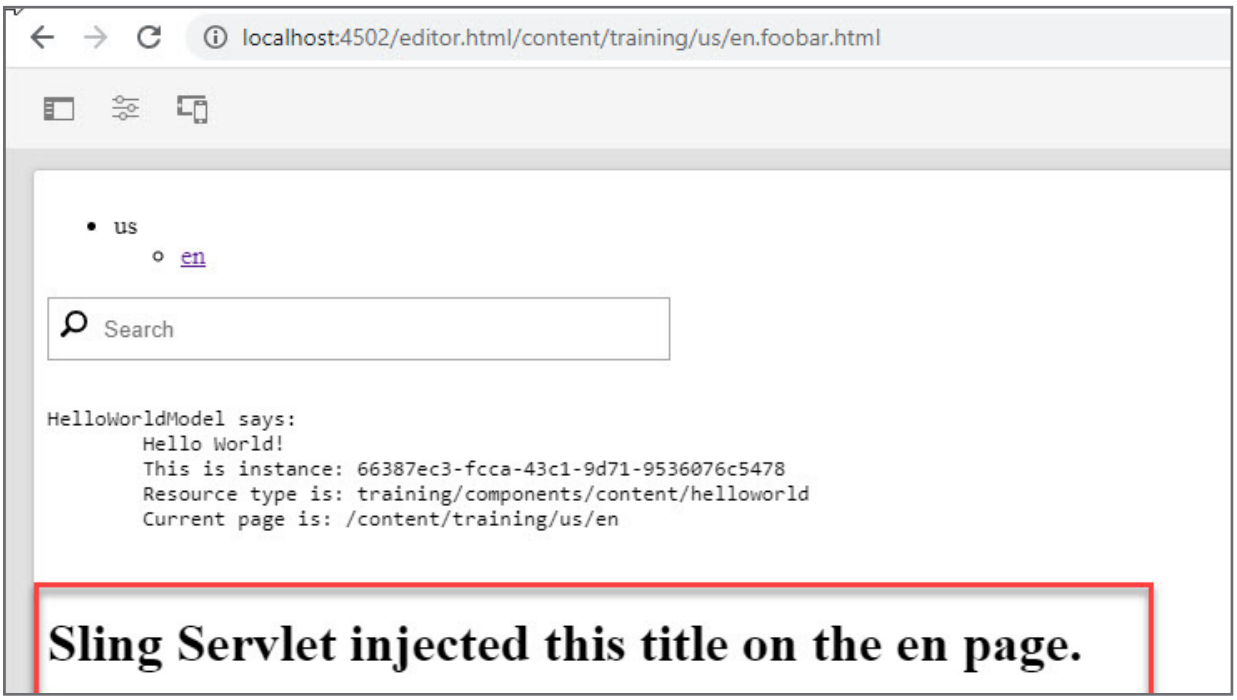
In this task you will add a selector to the servlet so that the servlet is only injected to the title component when the selector is also added.

1. In **Project Explorer,** edit **TitleSlingServlet(training.core/src/main/java/com.adobe.training.core.servlets/TitleSlingServlet.java**) to change the servlet implementation by uncommenting line 29 (by deleting //), as shown:

```
24
25  */
26 @Component( service = Servlet.class,
27             property = {
28                 "sling.servlet.resourceTypes="+TitleSlingServlet.RESOURCE_TYPE,
29                 "sling.servlet.selectors=foobar",
30                 "sling.servlet.extensions=html"
31             })
```

2. Save the changes.
3. Right-click **training.core** and select **Run As > Maven install** to build the project. The project is built.
4. Go to http://localhost:4502/content/training/us/en.html and notice how the servlet is no longer being injected. The change is now "My Title Component on English Page" is back to being displayed on the page, as shown:

5. Open http://localhost:4502/content/training/us/en.foobar.html and see the output, as shown:



Notice how the servlet is triggered when the request has a foobar selector in it.
Congratulations! You have successfully used selector to trigger the servlet on your page.

Creating System users

Adobe suggests developers create a system user that map to a subsystem using a 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. In general, such tasks cannot be configured with user names and passwords. Hard coding the passwords in the code or having the passwords in – more or less – plain text in some configuration is NOT 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 the need 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.

Exercise 2: Create a System user

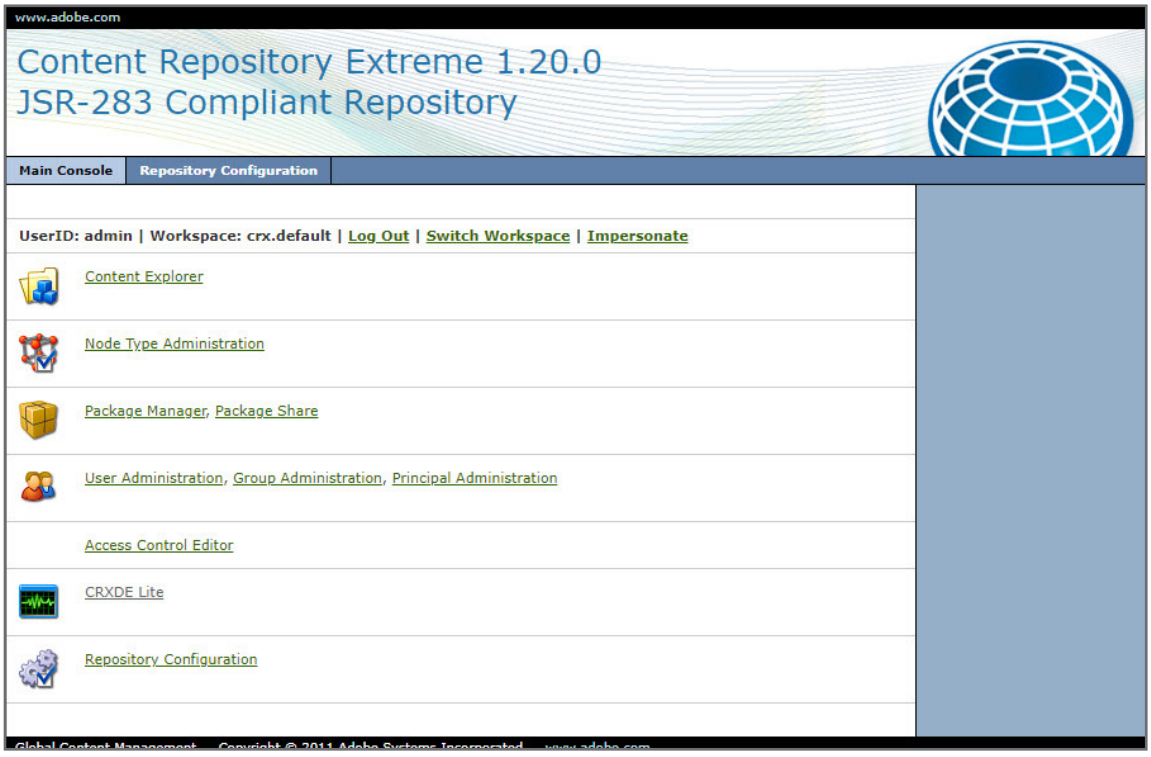
In this exercise, you will create a system user and provide access rights to the user.

This exercise includes two tasks:

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

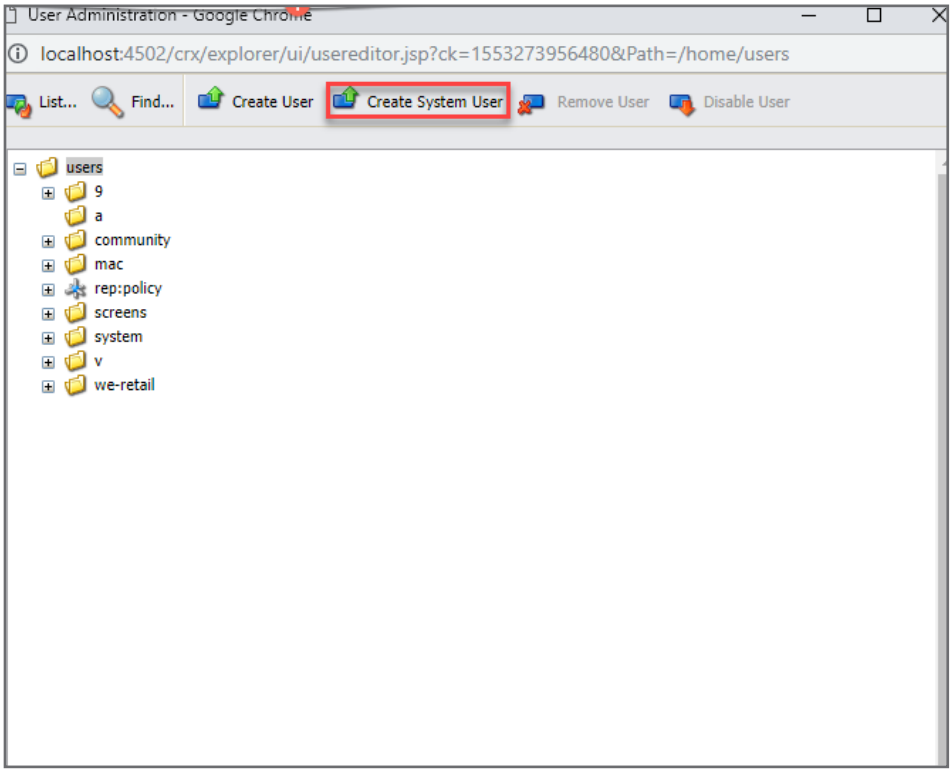
Task 1: Create a service user

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

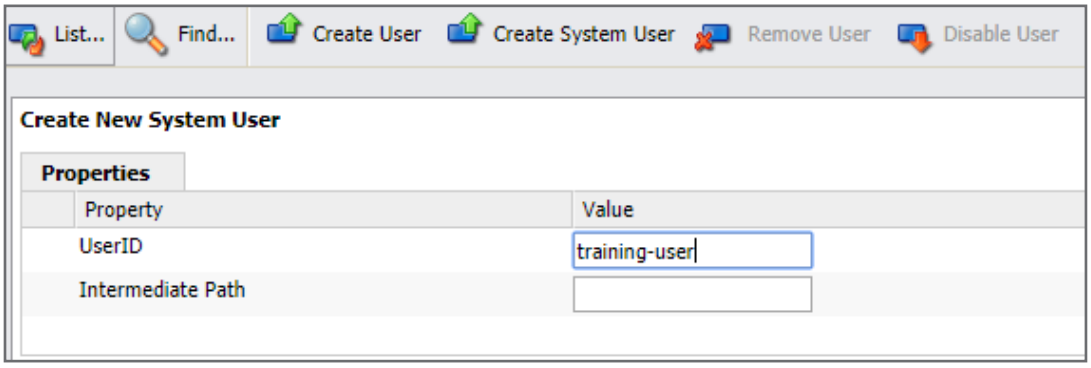


3. Click **User Administration**. The **User Administration** opens.

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

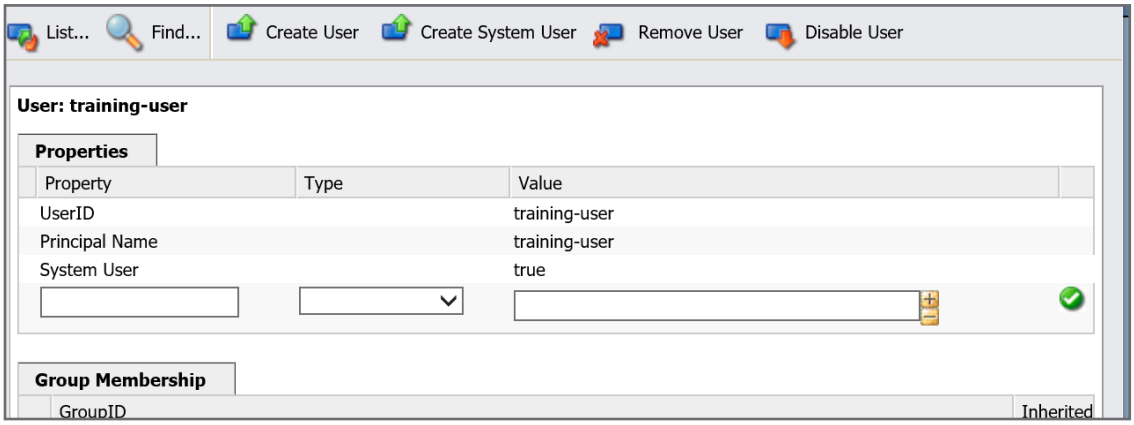


5. In the **UserID** field, type **training-user**, as shown:

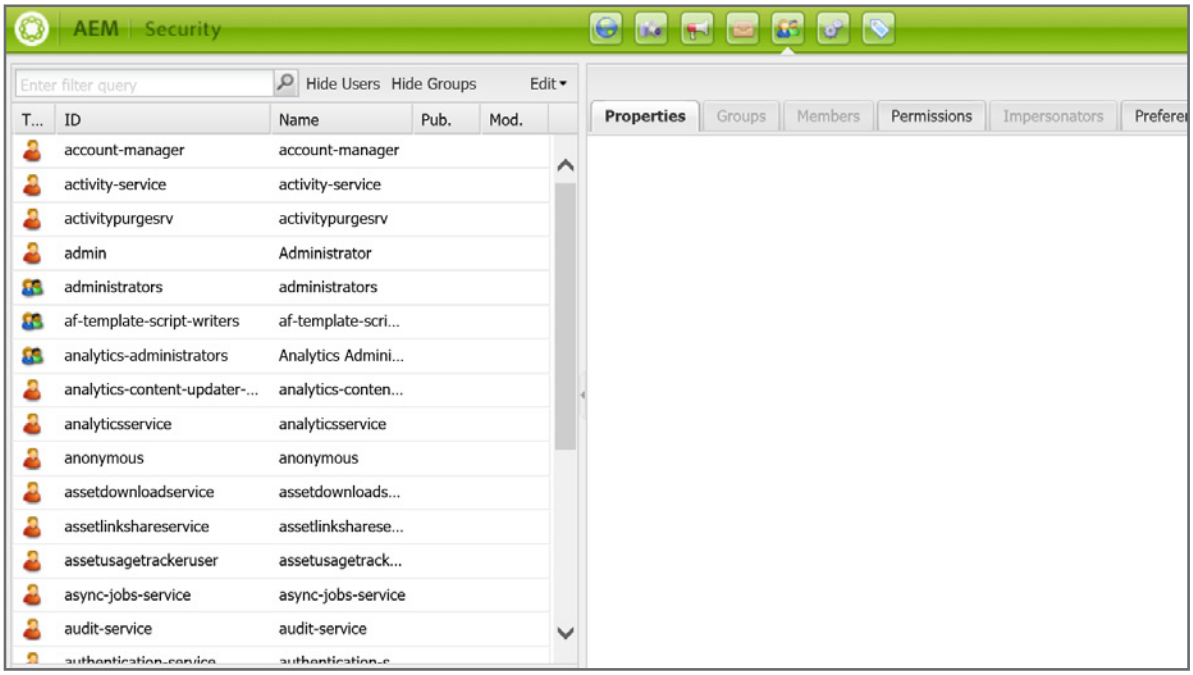


6. Click the green checkmark in the lower-right corner of the dialog box.

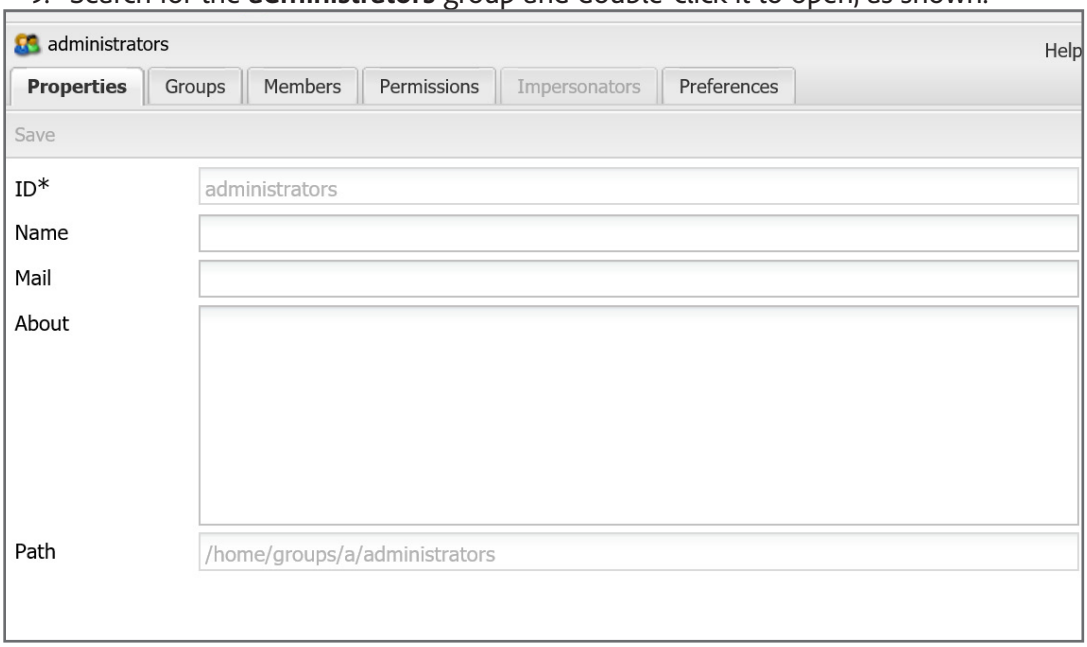
7. Click **Close** in the lower-right corner. The **training-user** is created.



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

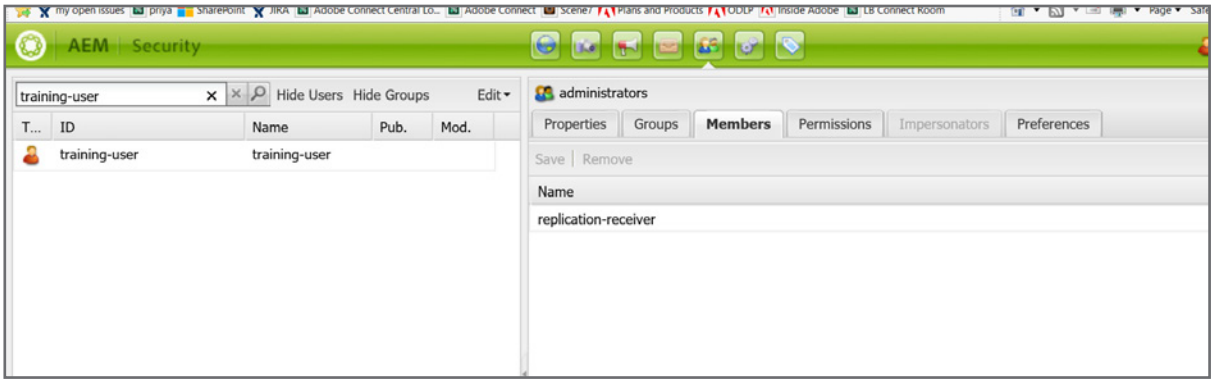


9. Search for the **administrators** group and double-click it to open, as shown:

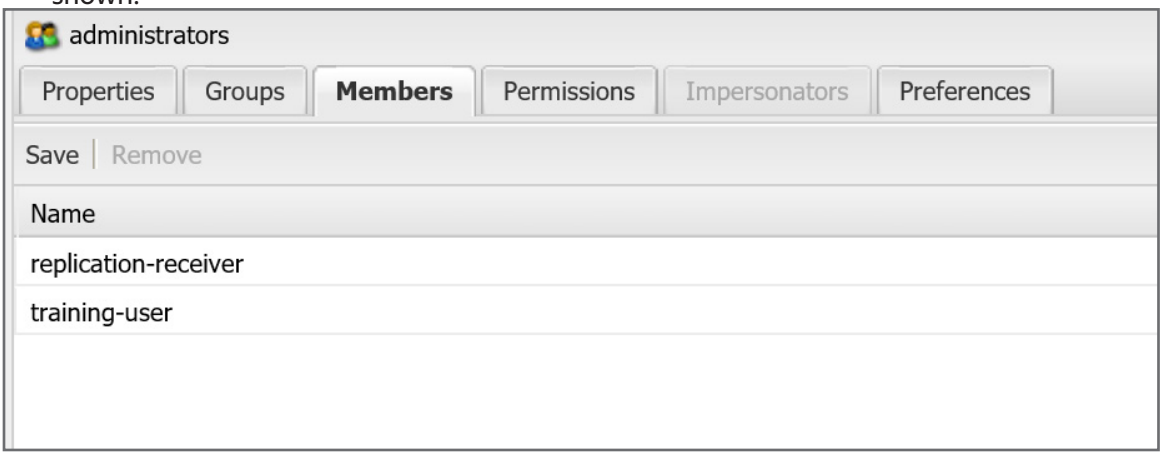


10. Click the **Members** tab. The **Members** tab opens.

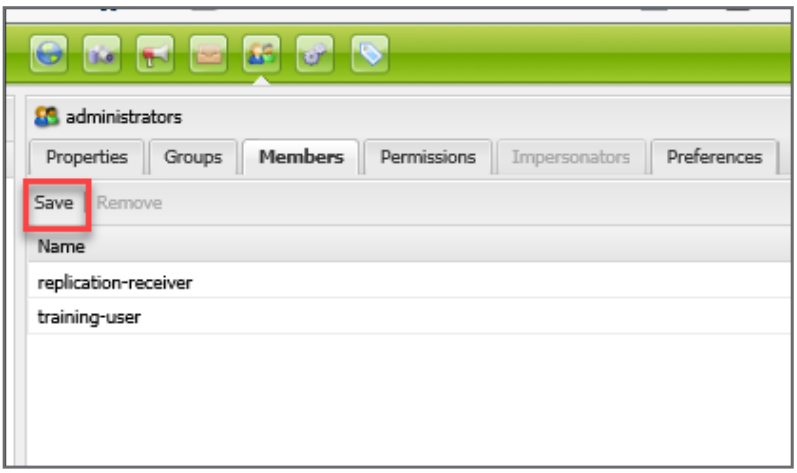
11. On the left, search for **training-user** by using the search option, as shown:



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



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

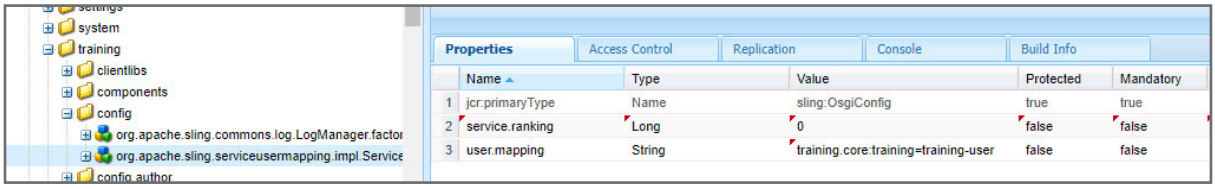


Task 2: Create the mapping to the user

In this task, you will create the mapping from our bundle to the user with an OSGi configuration node. The Bundles tab is the mechanism for installing the OSGi bundles required for AEM. The configuration we are going to setup is the ServiceUserMapperImpl.

1. Navigate to: http://localhost:4502/crx/de.
2. Under **/apps/training/config**, right-click **config** and select **Create** > **Create node**.
3. Provide the following details:
   - Name: **org.apache.sling.serviceusermapping.impl.ServiceUserMapperImpl.amended-training**
   - Type: **sling:OsgiConfig**
4. Click **OK** and save the changes. The node **org.apache.sling.serviceusermapping.impl.ServiceUserMapperImpl.amended-training** is created.
5. Add the following two properties to the node with the values provided:
   - Name=**service.ranking**, type=**Long**, value=**0**
   - Name=**user.mapping,** type=**String**, value=**training.core:training=training-user**

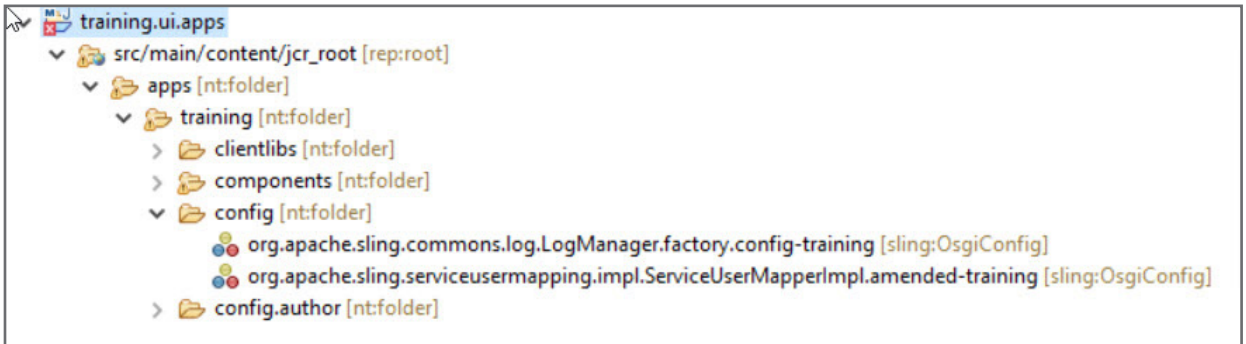
The user.mapping allows us to map a bundle with a subservice name to a service user, as shown: Bundle:training.core, Subservice name: training, and Service User: training-user



6. Save the changes. The properties are saved.

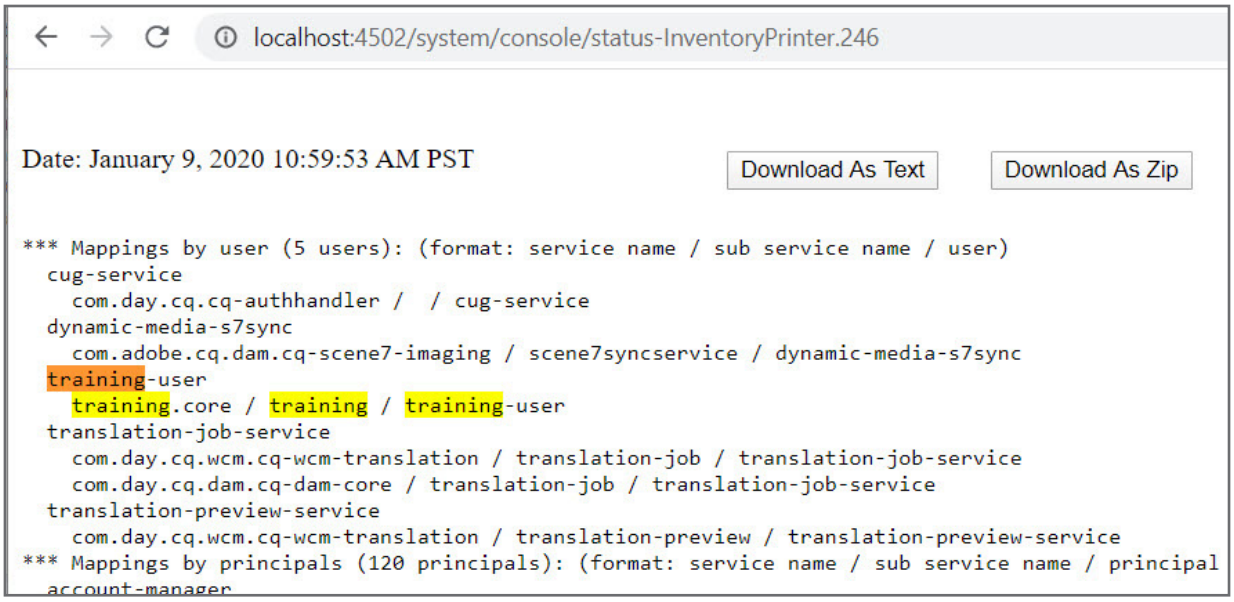
Note: Since you used CRXDE Lite to create the node, you need to syncronzie it back to the maven project in eclipse.

7. In Eclipse, go to **training.ui.apps.**
8. Right click on **src/main/content/jcr_content > apps > training > config** and select **Import from server...**.
9. Accept the default values and click **Finish**. The changes are imported from the server, as shown:



10. Navigate to the **Web Console** (http://localhost:4502/system/console/configMgr) and choose **Status** > **Sling Service User Mappings**. The user mappings are displayed.

11. Use the browser's search to look for and 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 created from a configuration factory and, therefore, needs a unique identifier.

Note: You can create a node with the serialized XML document instead. Working with XML files(that represent nodes) outside of AEM is a typical developer practice in an IDE. The XML file named **org.apache.sling.serviceusermapping.impl.ServiceUserMapperImpl.amended-training.xml** is provided as part of **Exercise_Files** under **/training-files/Sling_Web_Framework**.

References

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

- Apache Sling: https://sling.apache.org/
- Apache Felix: https://felix.apache.org/

Event Handling in AEM

Introduction

With Adobe Experience Manager (AEM), there are different approaches that can be taken regarding event management. Depending on the complexity, the nature of the payload or intensity of process, and inherent requirements or business roles involved, we can consider the following techniques:

- Events at the OSGi container level, with the OSGi Event Admin
- Events at the Sling level, with Sling Jobs (that you can schedule) and the ResourceChangeListener
- Events at the level of AEM, with different high-level objects (for example, workflow process steps, polling importers, replication actions, notifications, or auditing - all based on Sling Jobs)
- Events at the repository level, with JCR observation

Objectives

After completing this course, you will be able to:

- Describe Sling scheduler
- Create a Job consumer
- Discuss event modeling
- Create a resource listener

Sling Jobs Scheduler

Sling offers the ability to start jobs at specified times or periodically, creating a Sling Job, configuring a firing schedule, and then adding it to the Jobs queue. The Sling Jobs Scheduler mechanism can use a cron expression for setting a schedule for the Job to be kicked off.

A cron expression is defined with this pattern:

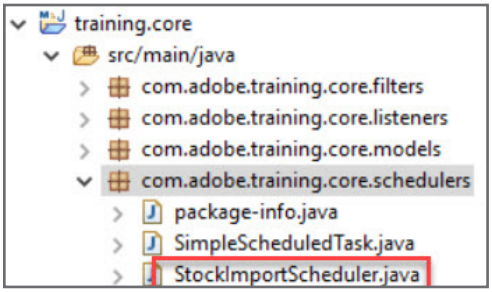
<Seconds> <Minutes> <hours> <Day of Month> <Month> <Day of Week> <Year>

Character	Description
*	Used to specify all values
?	Allowed for the day-of-month and day-of-week fields. It is used to specify 'no specific value'.
-	Used to specify ranges
,	Used to specify additional values
/	Used to specify increments

Exercise 1: Schedule a Sling job

In this exercise, you will create a OSGi component that creates a new scheduled Sling Job based on the OSGi configs provided. This OSGi component is also a config factory implying there can be many instances. Note that this component only adds scheduled jobs to the queue and does not actually process the Jobs..

1. In Project Explorer, navigate to **training.core** > **src/main/java**.
2. Copy the file **StockImportScheduler.java** from **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/schedulers/.**
3. Right-click **com.adobe.training.core.schedulers** and select **Paste** to paste the file. The **StockImportScheduler.java** class is created, as shown:



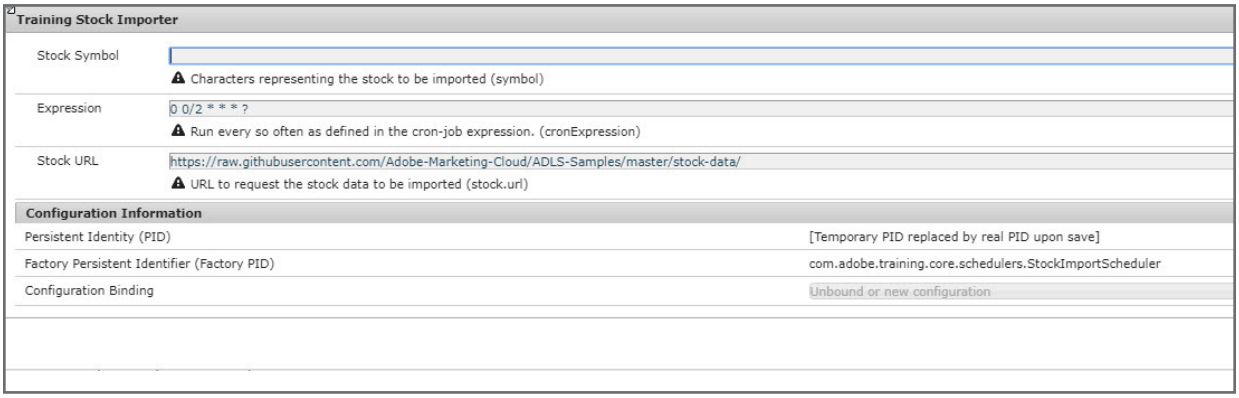
4. Double-click **StockImportScheduler.java**. The file opens in the editor, as shown:

```
1.  package com.adobe.training.core.schedulers;
2.  import java.util.ArrayList;
3.  import java.util.HashMap;
4.  import java.util.List;
5.  //org.apache.sling.event.*  api available in Sling 9 documentation:
6.  import org.apache.sling.event.jobs.JobBuilder;
7.  import org.apache.sling.event.jobs.JobBuilder.ScheduleBuilder;
8.  import org.apache.sling.event.jobs.JobManager;
9.  import org.apache.sling.event.jobs.ScheduledJobInfo;
10. import org.osgi.service.component.annotations.Activate;
11. import org.osgi.service.component.annotations.Component;
12. import org.osgi.service.component.annotations.ConfigurationPolicy;
13. import org.osgi.service.component.annotations.Deactivate;
14. import org.osgi.service.component.annotations.Modified;
15. import org.osgi.service.component.annotations.Reference;
16. import org.osgi.service.metatype.annotations.AttributeDefinition;
17. import org.osgi.service.metatype.annotations.AttributeType;
18. import org.osgi.service.metatype.annotations.Designate;
19. import org.osgi.service.metatype.annotations.ObjectClassDefinition;
20. import org.slf4j.Logger;
21. import org.slf4j.LoggerFactory;
22. /**
23.  * This class adds a Sling Job to the job queue so that a job consumer can process
24.  * work. Sling Jobs are guaranteed to be proceed and the scheduler can be configured
25.  * based on an OSGi config node.
26.  */
```

```
@Component(immediate = true,
                configurationPid = "com.adobe.training.core.schedulers.StockImportScheduler",
                configurationPolicy = ConfigurationPolicy.REQUIRE)
@Designate(ocd = StockImportScheduler.Configuration.class, factory=true)
public class StockImportScheduler {
    private final Logger logger = LoggerFactory.getLogger(getClass());
    @ObjectClassDefinition(name = "Training Stock Importer")
    public @interface Configuration {
        @AttributeDefinition(
                name = "Stock Symbol",
                description = "Characters representing the stock to be imported",
                type = AttributeType.STRING
        )
        String symbol() default "";
        @AttributeDefinition(
            name = "Expression",
            description = "Run every so often as defined in the cron-job expression.",
            type = AttributeType.STRING
        )
        String cronExpression() default "0 0/2 * * * ?";

        @AttributeDefinition(
                name = "Stock URL",
                description = "URL to request the stock data to be imported",
                type = AttributeType.STRING
        )
        String stock_url() default "https://raw.githubusercontent.com/Adobe-Marketing-Cloud/ADLS-Samples/master/stock-data/";
    }
    @Reference
    private JobManager jobManager;

    private int schedulerID;

    private JobBuilder jobBuilder;
    private ScheduleBuilder scheduleBuilder;
    private ScheduledJobInfo theScheduledJob;

    public static final String STOCKIMPORT_JOB_TOPIC = "com/adobe/training/core/jobs/stockimportjob";

    public static final String STOCKIMPORT_SYMBOL = "symbol";
    public static final String URL_FOR_STOCKIMPORT = "url";
    @Activate @Modified
    protected void activate(Configuration config) {
        logger.info("****StockImport ScheduledJob '{}' with ID: '{}' Activated", config.symbol(), schedulerID);
        schedulerID = config.symbol().hashCode();
        startScheduledJob(config);
    }
    @Modified
    protected void modified(Configuration config) {
        removeScheduler(config);
        schedulerID = config.symbol().hashCode() + 1;  //updates schedulerID
        startScheduledJob(config);
    }
    @Deactivate
    protected void deactivate(Configuration config) {
        removeScheduler(config);
    }
    private void startScheduledJob(Configuration config){
        jobBuilder = jobManager.createJob(StockImportScheduler.STOCKIMPORT_JOB_TOPIC);
 //Create  a properties map that contains the configurations we want to pass to the job
        HashMap<String, Object> jobProps = new HashMap<>();
        jobProps.put(STOCKIMPORT_SYMBOL, config.symbol());
        jobProps.put(URL_FOR_STOCKIMPORT, config.stock_url());
        jobBuilder.properties(jobProps);
        scheduleBuilder = jobBuilder.schedule();
        scheduleBuilder.cron(config.cronExpression());
        theScheduledJob = scheduleBuilder.add();
```

```
92.         if(theScheduledJob == null){
93.             List<String> errors = new ArrayList<>();
94.             scheduleBuilder.add(errors);
95.         }
96.         logger.info("ScheduledJob added to the Queue.  Topic: " + theScheduledJob.
    getJobTopic() + "
97.                 + "Properties: " + theScheduledJob.getJobProperties().toString() + " "
98. toString());    + "Next Execution: " + theScheduledJob.getNextScheduledExecution().
99.     }
100.        private void removeScheduler(Configuration config) {
101.            logger.info("*******Removing '{}' ScheduledJob, with ID: '{}'", config.
    symbol(), schedulerID);
102.            theScheduledJob.unschedule();
103.        }
104.    }
```

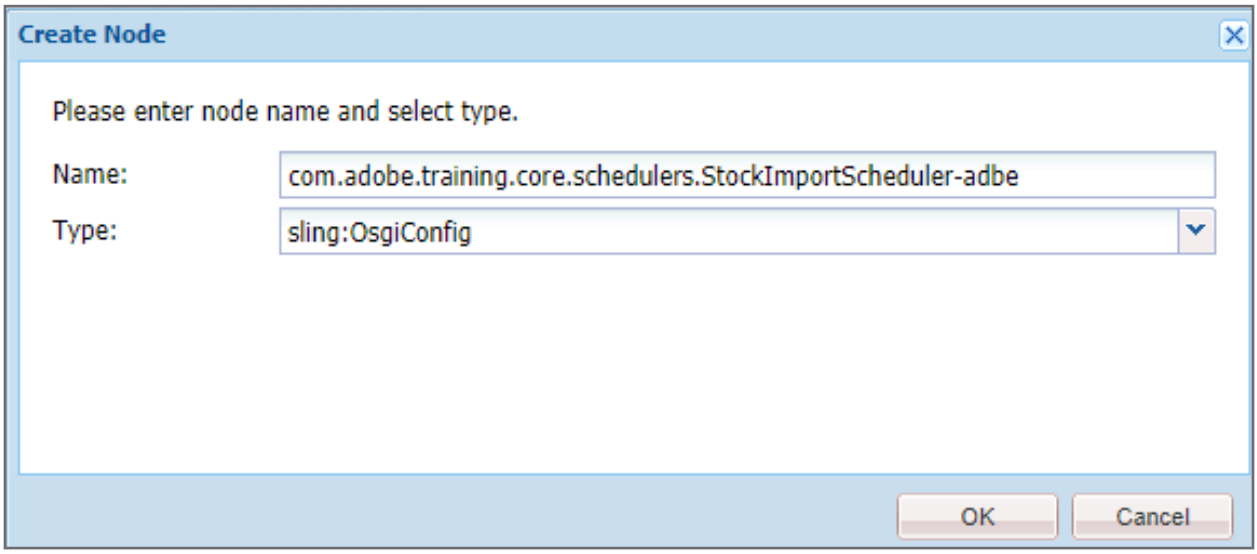
5. Right-click **training.core**, and select **Run As** > **Maven install**. The project is built.
6. Go to http://localhost:4502/system/console/configMgr and search for **Training Stock Importer**.
7. Click on **Training Stock Importer**. The **Training Stock Importer** window opens, as shown:



8. Notice how you now have the ability to add a Stock Symbol and then a cron expression. In order to set these values, you must create OSGi configuration nodes.
9. Copy the Factory Persistent Identifier (Factory PID), **com.adobe.training.core.schedulers.StockImportScheduler**, from the **Training Stock Importer** window**.** Paste this PID in the **Name** field when you create the node in the next step.

10. In **CRXDE Lite**, navigate to **/apps/training**. Expand **training** by clicking the plus sign (+). Right-click **config** and select **Create** > **Create Node**, and provide the following values, as shown:

- Name: **com.adobe.training.core.schedulers.StockImportScheduler-adbe**
- Type: **sling:OsgiConfig**

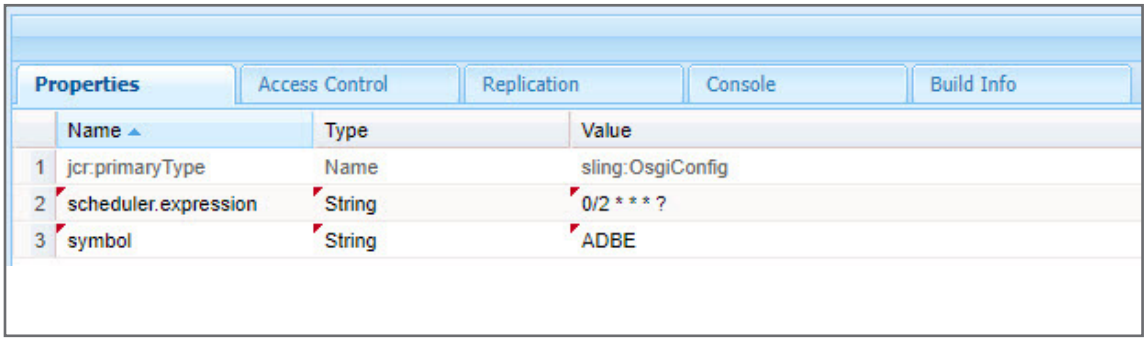


11. Click **OK**.
12. **Save All**.

NOTE: **adbe** was added as a unique identifier (you can make the unique identifier anything; however, for this exercise, it is **adbe**) to the PID. The unique identifier is needed to make an instance of the config factory.

13. In the new node you just created, add these two properties, using the values as shown:

- Name: **scheduler.expression** Type: (**String**) Value: **0/2 * * * ?**
- Name: **symbol** Type: (**String**) Value: **ADBE**

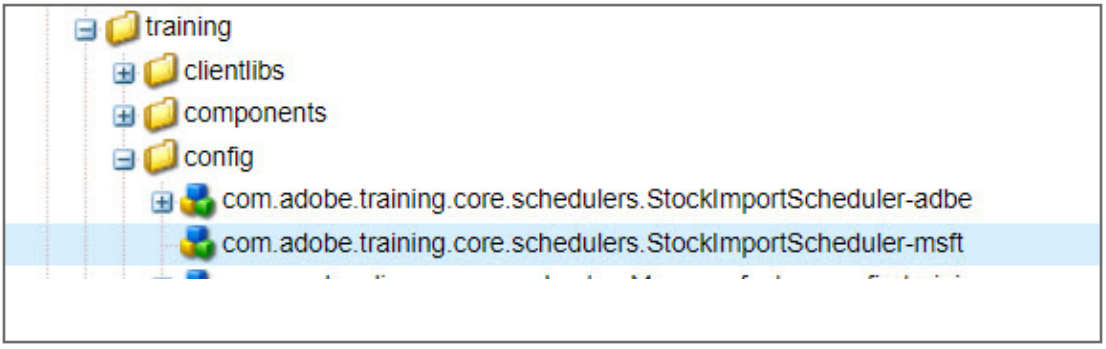


Note: A cron expression(scheduler.expression) is a string consisting of six or seven subexpressions (fields) that describe individual details of the schedule. Make sure these fields are separated by white space.

14. Click **Save All** to save the changes.

Note: Once you save the changes, the scheduler will start, then every 2 minutes, the scheduler will find the stock information which will be saved into the JCR.

15. To add another node, copy and paste the OSGi config node **com.adobe.training.core.schedulers.StockImportScheduler-adbe** and rename the unique identifier. For this exercise, use this example:
    - /apps/training/config/**com.adobe.training.core.schedulers.StockImportScheduler-msft**



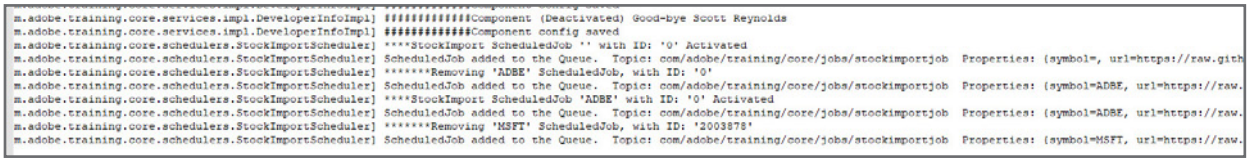
16. Update the **symbol** property:

- Name: **symbol** Type: (String) Value: **MSFT**

17. Save the changes.

18. Navigate to **/crx-quickstart/logs/** and open **project-training.log.**

19. Verify you can see logged messages for the scheduled jobs, as shown:



Working with Sling jobs

Every time there is a need to process a given event only once and ensure the event is not missed, you can leverage the Sling Job processing mechanism. You can see it as if you wanted to print a document with a specific printer on a network: you do not care if the printer is already busy or even switched off, any printing job sent to it will be processed once the printer will be ready, and only that chosen printer will do it.

Contrary to OSGi events, Sling Jobs are guaranteed to be processed only once. On the other hand, they add some overhead, so they would be used only when really necessary (the other methods discussed in this module could be a better choice in some cases).

Working with Sling Jobs requires the following components:

- A job manager, used to add a job to a queue with data associated to that job
- A job consumer, to process that job and remove it from the queue

Because of the uniqueness of the process, once a job consumer has processed, it gets removed from the queue and no other consumer would be able to process it (which is a case we should avoid anyway). Similarly, if no job consumer is registered to process a given job topic, the job stays in the queue (which is again something that would be considered a deployment error).

Adding a job consists of calling the method **addJob()** of the **org.apache.sling.event.jobs.JobManager** service and passing to it a topic (a string identifying the job) and a HashMap to carry the job information.

A component that needs to consume a job for processing must implement the **org.apache.sling.event.jobs.consumer.JobConsumer** interface and override the process method to get the Job object **org.apache.sling.event.jobs.Job**, as shown below:

```
import org.apache.sling.event.jobs.Job;
import org.apache.sling.event.jobs.consumer.JobConsumer;
@Component  (service = JobConsumer.class,
        Property = JobConsumer.PROPERTY_TOPICS + "=com/adobe/training/core/myjobtopic")

public class MyTopicProcessor implements JobConsumer {
    @override
    public JobResult process(final Job job) {
        // do something
}
}
```

By convention, the topic name is built by using the bundle symbolic name for the prefix (with slashes instead of dots) and suffixed with a custom label identifying the topic.

Monitoring

To view all registered topics with their statistics, go to the Web Console under **Sling** > **Jobs** in the Web Console:

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

Note: This console only shows topics that were processed by a Job consumer. Therefore, if a job is added and never consumed, the corresponding topic will not be visible. For debugging purposes, you can find custom jobs in the repository under **/var/eventing/jobs**.

Exercise 2: Consume Sling job

In the previous exercise, you added a scheduled Sling Job with the JobManager. In this exercise, you will process the Sling Job that was put in the Job Queue. The Job consumer you will build imports data from an external API endpoint and then writes it into the JCR. For this example we are using a dummy stock data endpoint located in git, though in realistic scenarios this method could be used for integrations and connecting other 3rd party sources to AEM.

1. In the **Eclipse Project Explorer**, navigate to **training.core** > **src/main/java**.
2. Copy the file **StockDataWriterJob.java** from **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/**.
3. In **Eclipse**, right-click **com.adobe.training.core** and paste the file. The **StockDataWriterJob.java** class is created.

Note: The code is provided as part of the **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/**. Copy and paste the file from the exercise files referenced to **StockDataWriterJob.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 **StockDataWriterJob.java**. The file opens in the editor, as shown:

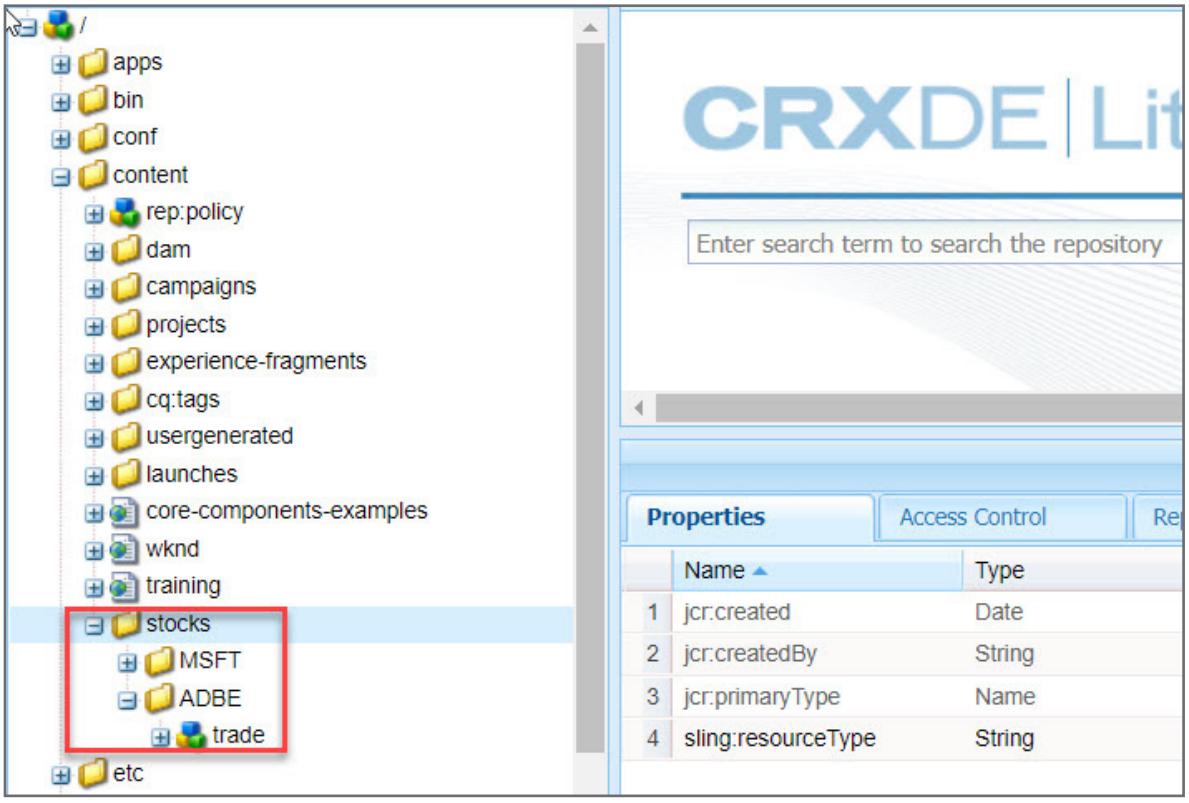
```
1.  package com.adobe.training.core;
2.  import java.io.IOException;
3.  import java.io.InputStream;
4.  import java.io.InputStreamReader;
5.  import java.net.SocketTimeoutException;
6.  import java.net.URL;
7.  import java.time.Instant;
8.  import java.time.LocalDateTime;
9.  import java.time.ZoneId;
10. import java.time.ZonedDateTime;
11. import java.time.format.DateTimeFormatter;
12. import java.util.HashMap;
13. import java.util.Map;
14. import javax.jcr.RepositoryException;
15. import javax.net.ssl.HttpsURLConnection;
16. import org.apache.sling.api.resource.LoginException;
17. import org.apache.sling.api.resource.ModifiableValueMap;
18. import org.apache.sling.api.resource.PersistenceException;
19. import org.apache.sling.api.resource.Resource;
20. import org.apache.sling.api.resource.ResourceResolver;
21. import org.apache.sling.api.resource.ResourceResolverFactory;
22. import org.apache.sling.api.resource.ResourceUtil;
23. import org.apache.sling.event.jobs.Job;
24. import org.apache.sling.event.jobs.consumer.JobConsumer;
25. import org.osgi.service.component.annotations.Component;
```

```
26. import org.osgi.service.component.annotations.Reference;
27. import org.slf4j.Logger;
28. import org.slf4j.LoggerFactory;
29. import com.day.cq.commons.jcr.JcrConstants;
30. import com.fasterxml.jackson.core.JsonFactory;
31. import com.fasterxml.jackson.core.JsonParseException;
32. import com.fasterxml.jackson.core.JsonParser;
33. import com.fasterxml.jackson.core.type.TypeReference;
34. import com.fasterxml.jackson.databind.ObjectMapper;
35.  import com.adobe.training.core.schedulers.StockImportScheduler;
36.  /**
37. * This job consumer takes in a data source url and stock symbol
38. * and creates the node structure below.
39. *
40. * /content/stocks/
41. *   + <STOCK_SYMBOL> [sling:OrderedFolder]
42. *     + trade [nt:unstructured]
43. *             - companyName = <value>
44. *             - sector = <value>
45. *           - lastTrade = <value>
46. *           - timeOfUpdate = <value>
47. *           - dayOfLastUpdate = <value>
48. *           - openPrice = <value>
49. *           - rangeHigh = <value>
50. *           - rangeLow = <value>
51. *           - volume = <value>
52. *           - upDownPrice = <value>
53. *           - week52High = <value>
54. *           - week52Low = <value>
55. *           - ytdChange = <value>
56. */
57. @Component(
58.         immediate = true,
59.         service = JobConsumer.class,
60.         property ={
61. TOPIC             JobConsumer.PROPERTY_TOPICS +"="+ StockImportScheduler.STOCKIMPORT_JOB_
62.         }
63. )
64. public class StockDataWriter implements JobConsumer {
65.     private final Logger logger = LoggerFactory.getLogger(getClass());
66.
67.     //Public values for stock data
68.     public static final String STOCK_IMPORT_FOLDER = "/content/stocks";
69.     public static final String COMPANY = "companyName";
70.     public static final String SECTOR = "sector";
71.     public static final String LASTTRADE = "lastTrade";
72.     public static final String UPDATETIME = "timeOfUpdate";
73.     public static final String DAYOFUPDATE = "dayOfLastUpdate";
74.     public static final String OPENPRICE = "openPrice";
75.     public static final String RANGEHIGH = "rangeHigh";
76.     public static final String RANGELOW = "rangeLow";
77.     public static final String VOLUME = "volume";
78.     public static final String UPDOWN = "upDown";
79.     public static final String WEEK52LOW = "week52Low";
80.     public static final String WEEK52HIGH = "week52High";
81.     public static final String YTDCHANGE = "ytdPercentageChange";
82.      @Reference
83.     private ResourceResolverFactory resourceResolverFactory;
84.
85.     /**
86.      * Method that runs on the desired schedule.
87.      * Request the data with the stock symbol and get the returned JSON
88.      * Write the JSON to the JCR
89.      */
90.     @Override
91.     public JobResult process(Job job) {
92.         //extract properties added to the Job in Scheduler:
93.         String symbol = job.getProperty(StockImportScheduler.STOCKIMPORT_SYMBOL).
    toString();
94.         String stock_url = job.getProperty(StockImportScheduler.URL_FOR_STOCKIMPORT).
    toString();
95.         //https://raw.githubusercontent.com/Adobe-Marketing-Cloud/ADLS-Samples/master/
    stock-data/
96.         String stockUrl = stock_url + symbol + ".json";
97.         HttpsURLConnection request = null;
98.         try {
99.             URL sourceUrl = new URL(stockUrl);
100.                 request = (HttpsURLConnection) sourceUrl.openConnection();
```

```
                    request.setConnectTimeout(5000);
                    request.setReadTimeout(10000);
                    request.connect();
                } catch (SocketTimeoutException e) {
                    logger.error("5 Second Timeout occured.");
                    return JobConsumer.JobResult.FAILED;
                }
                catch (IOException e) {
                    logger.error("The stock symbol: " + symbol + " does not exist...");
                    return JobConsumer.JobResult.FAILED;
                }

                // Convert data return to a JSON object
                ObjectMapper objMapper = new ObjectMapper();
                JsonFactory factory = new JsonFactory();
                JobResult jobResult = null;
                if(request != null) {
                    //Create a JsonParser based on the stream from the request content
                    try(JsonParser parser  = factory.createParser(new InputStreamReader((InputStream) request.getContent()))){

                        //Create a Map from the JsonParser
                        Map<String, String> allQuoteData = objMapper.readValue(parser,
                                new TypeReference<Map<String,String>>(){});

                        logger.info("Last trade for stock symbol {} was {}", symbol, allQuoteData.get("latestPrice"));

                        //Use the map to write nodes and properties to the JCR
                        jobResult = writeToRepository(symbol, allQuoteData);
                    }
                    catch (RepositoryException e) {
                        logger.error("Cannot write stock info for " + symbol + " to the JCR: ", e);
                        return JobConsumer.JobResult.FAILED;
                    } catch (JsonParseException e) {
                        logger.error("Cannot parse stock info for " + symbol, e);
                        return JobConsumer.JobResult.FAILED;
                    } catch (IOException e) {
                        logger.error("IOException: ", e);
                        return JobConsumer.JobResult.FAILED;
                    }
                }
                return jobResult;
            }

            /**
             * Creates the stock data structure
             *
             *  + <STOCK_SYMBOL> [sling:OrderedFolder]
             *     + trade [nt:unstructured]
             *       - companyName = <value>
             *       - sector = <value>
             *       - lastTrade = <value>
             *       - timeOfUpdate = <value>
             *       - dayOfLastUpdate = <value>
             *       - openPrice = <value>
             *       - rangeHigh = <value>
             *       - rangeLow = <value>
             *       - volume = <value>
             *       - upDownPrice = <value>
             *       - week52High = <value>
             *       - week52Low = <value>
             *       - ytdChange = <value>
             * @return
             */
        private JobResult writeToRepository(String stockSymbol, Map<String, String> quoteData) throws RepositoryException {

                logger.info("Stock Symbol: " + stockSymbol);
                logger.info("JsonObject to Write: " + quoteData.toString());

                //Get the service user that belongs to the com.adobe.training.core:training subservice

                //Returns the training-user service user
                Map<String, Object> serviceParams = new HashMap<>();
                serviceParams.put(ResourceResolverFactory.SUBSERVICE, "training");
```

```
168.                try (ResourceResolver resourceResolver = resourceResolverFactory
169.                        .getServiceResourceResolver(serviceParams)) {
170.                    // Transform the time stamp into a readable format
171.                    ZoneId timeZone = ZoneId.of("America/New_York");
172.                    long latestUpdateTime = Long.parseLong(quoteData.
    get("latestUpdate"));
173.                    LocalDateTime timePerLatestUpdate = LocalDateTime.ofInstant(Instant.
    ofEpochMilli(latestUpdateTime),
174.                                    timeZone);
175.                    ZonedDateTime timeWithZone = ZonedDateTime.
    of(timePerLatestUpdate, timeZone);
176.                    DateTimeFormatter timeFormatter = DateTimeFormatter.
    ofPattern("hh:mm a zz");
177.                    //will store timeOfUpdate as:  Hour:Minute AM/
    PM, TimeZone    e.g.   11:34 AM, EDT
178.                    String UpdateTimeOfDay = timeWithZone.format(timeFormatter);
179.                    DateTimeFormatter dayFormatter = DateTimeFormatter.
    ofPattern("E MMMM d, yyyy");
180.                    String dayOfUpdate = timeWithZone.format(dayFormatter);
181.                    //Create variables in specific data type and put them into a map
182.                    Double lastPrice = Double.parseDouble(quoteData.get("latestPrice"));
183.                    Double open = Double.parseDouble(quoteData.get("open"));
184.                    Double high = Double.parseDouble(quoteData.get("high"));
185.                    Double low = Double.parseDouble(quoteData.get("low"));
186.                    Long latestVolume = Long.parseLong(quoteData.get("latestVolume"));
187.                    Double change = Double.parseDouble(quoteData.get("change"));
188.                    Double week52High = Double.parseDouble(quoteData.get("week52High"));
189.                    Double week52Low = Double.parseDouble(quoteData.get("week52Low"));
190.                    Double ytdChange = Double.parseDouble(quoteData.get("ytdChange"));
191.
192.                    String stockPath = STOCK_IMPORT_FOLDER + "/" + stockSymbol;
193.                    String tradePath = stockPath + "/trade";
194.                    Resource trade = resourceResolver.getResource(tradePath);
195.                    //Test if stock import folder exists, otherwise create it
196.                    Resource stockFolder = ResourceUtil.
    getOrCreateResource( resourceResolver, stockPath, "", "", false );
197.                    if (trade == null) {
198.        //set jcr:primaryType to nt:unstructured when resource is created
199.                        Map<String,Object> stockData = new HashMap<String,Object>() {{
200.UNSTRUCTURED);           put(JcrConstants.JCR_PRIMARYTYPE, JcrConstants.NT_
201.                        }} ;
202.                        trade = resourceResolver.
    create(stockFolder, "trade", stockData);
203.                    }
204.class);             ModifiableValueMap stockData = trade.adaptTo(ModifiableValueMap.
205.                    stockData.put(COMPANY, quoteData.get("companyName"));
206.      stockData.put(SECTOR, quoteData.get("sector"));
207.      stockData.put(UPDATETIME, UpdateTimeOfDay);
208.      stockData.put(DAYOFUPDATE, dayOfUpdate);
209.      stockData.put(LASTTRADE, lastPrice);
210.      stockData.put(OPENPRICE, open);
211.      stockData.put(RANGEHIGH, high);
212.      stockData.put(RANGELOW, low);
213.      stockData.put(VOLUME, latestVolume);
214.      stockData.put(UPDOWN,change);
215.      stockData.put(WEEK52HIGH,week52High);
216.      stockData.put(WEEK52LOW,week52Low);
217.      stockData.put(YTDCHANGE,ytdChange);
218.                    logger.info("Update trade data");
219.                    //Write data into the JCR
220.                    resourceResolver.commit();
221.                } catch (LoginException | PersistenceException e) {
222.                    logger.error("Exception with writing resource: ", e);
223.                    return JobConsumer.JobResult.FAILED;
224.                }
225.
226.                return JobConsumer.JobResult.OK;
227.            }
228.        }
```

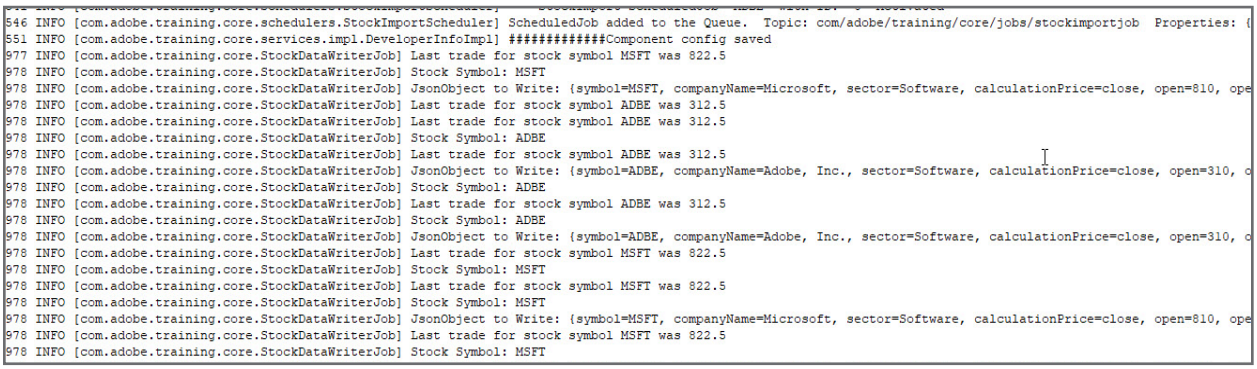
5. Right-click **training.core** and select **Run As** > **Maven install**. The project is built.
6. Refresh **CRXDE Lite.** Verify the **stocks** folder has been created under the **/content** folder, as shown:



Note: Because the stocks folder exists under /content, it can be seen within the Sites console (sties.html). Unfortunately, this is read-only since stocks are autocreated with an OSGi config. If you want to learn more about how to make the stocks folder more user-friendly, refer to the Optional exercise below for the StockListener.

7. Navigate to **/crx-quickstart/logs/** and open **project-training.log.**

8. Verify you can see logged messages consuming the trade data, as shown:



Sling Resource Change Listening

For a component to be notified of changes happening in the repository, some techniques with fewer overhead than Sling Jobs and a richer interface than OSGi events can be considered. For example, the JCR Observation, which is a request in the JCR specification, is typically a mechanism that allows the ability to monitor changes on nodes or properties in a JCR repository, at a certain path and under certain conditions on nodetypes or UUIDs. This can be achieved by implementing the **javax.jcr.observation.EventListener interface** and creating an event listener with the javax.jcr.observation.ObservationManager interface.

However, because it is usually a better practice to use higher-level APIs, we can conveniently do the same with Sling and the **org.apache.sling.api.resource.observation.ResourceChangeListener** interface.

The **ResourceChangeListener** is registered with a mandatory property specifying the paths under which the change of the resource (or resource provider) is to be monitored, and optionally with the type of changes expected.
Resource changes can then be caught by overriding the method onChange() as shown in this example:

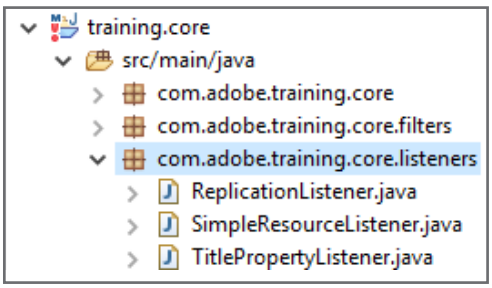
```
import org.apache.sling.api.resource.observation.ResourceChange;
import org.apache.sling.api.resource.observation.ResourceChangeListener;
@Component  (immediate = true,
service = ResourceChangeListener.class
    property = {"resource.paths=/content/mypath ",
                "resource.change.types=CHANGED"})

public class MyResourceListener implements ResourceChangeListener {
    @override
    public void onChange(List<ResourceChange> changes) {
        // do something
}
}
```

The advantage of this method over OSGi Events or Sling Jobs is that it allows the ability to restrict events to one (or several) specified path(s). Another advantage is that bulk listening operations will result in a single thread (the listener will only be executed once), whereas with OSGi Events or Sling Jobs, the handler would be called as many times as the amount of operations.
For these reasons, listening to only resource-related events is preferable, using the ResourceChangeListener for performance considerations.

Exercise 3: 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**.
2. Copy the file **TitlePropertyListener.java** from **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/listeners/.**
3. Right-click **com.adobe.training.core.listeners** and paste the file. The **TitlePropertyListener.java** class is created, as shown:



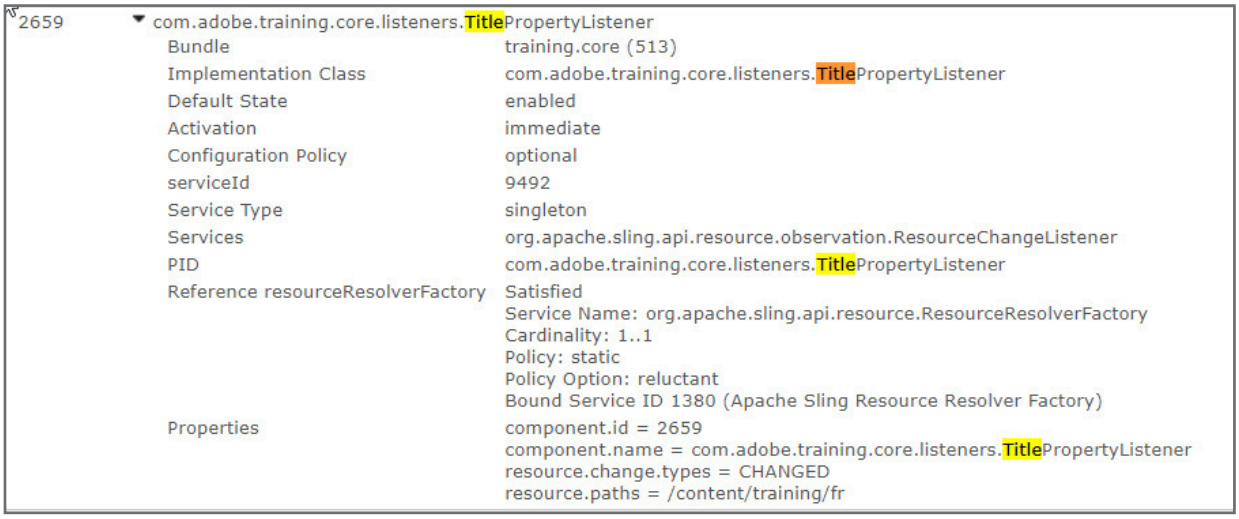
4. The file opens in the editor, as shown:

```
1.  package com.adobe.training.core.listeners;
2.  import org.apache.sling.api.resource.*;
3.  import org.osgi.service.component.annotations.Component;
4.  import org.osgi.service.component.annotations.Reference;
5.  import org.apache.sling.api.resource.observation.ResourceChange;
6.  import org.apache.sling.api.resource.observation.ResourceChangeListener;
7.  import java.util.HashMap;
8.  import java.util.List;
9.  import java.util.Map;
10. import org.slf4j.Logger;
11. import org.slf4j.LoggerFactory;
12. import com.day.cq.commons.jcr.JcrConstants;
13. @Component( immediate = true,
14.         property = {"resource.paths=/content/training/us/fr",
15.                 "resource.change.types=CHANGED"} )
16. public class TitlePropertyListener implements ResourceChangeListener{
17.     private final Logger logger = LoggerFactory.getLogger(getClass());
18.     @Reference
19.     private ResourceResolverFactory resourceResolverFactory;
20.     @Override
21.     public void onChange(List<ResourceChange> changes) {
22.         for (final ResourceChange change : changes) {
23.             logger.info("Change type: {}", change);
24.             Map<String,Object> serviceParams = new HashMap<String,Object>(){
```

```
                private static final long serialVersionUID = 1L;
                {put(ResourceResolverFactory.SUBSERVICE, "training");}
            };

            try (ResourceResolver rr = resourceResolverFactory.getServiceResourceResolver(serviceParams))
            {
                logger.error("Repository login success");
                Resource changedRes = rr.getResource(change.getPath());
                String titleProperty = (changedRes != null) ? changedRes.getValueMap().get(JcrConstants.JCR_TITLE, String.class) : "";
// If the title property is not empty and the current title value does not end in !
                if(changedRes != null) {
                    if (!titleProperty.isEmpty() && !titleProperty.endsWith("!")) {
                        ModifiableValueMap modChangedResource = changedRes.adaptTo(ModifiableValueMap.class);
                        modChangedResource.put(JcrConstants.JCR_TITLE, titleProperty + "!");
                        rr.commit();
                        logger.info("*************Property updated: {}", change.getPath());
                    }
                }
            } catch(PersistenceException e ){
                logger.error("Failed to write the resource change.");
            }  catch(LoginException e ){
                logger.error("Repository login failed.");
            }
        }
    }
}
```

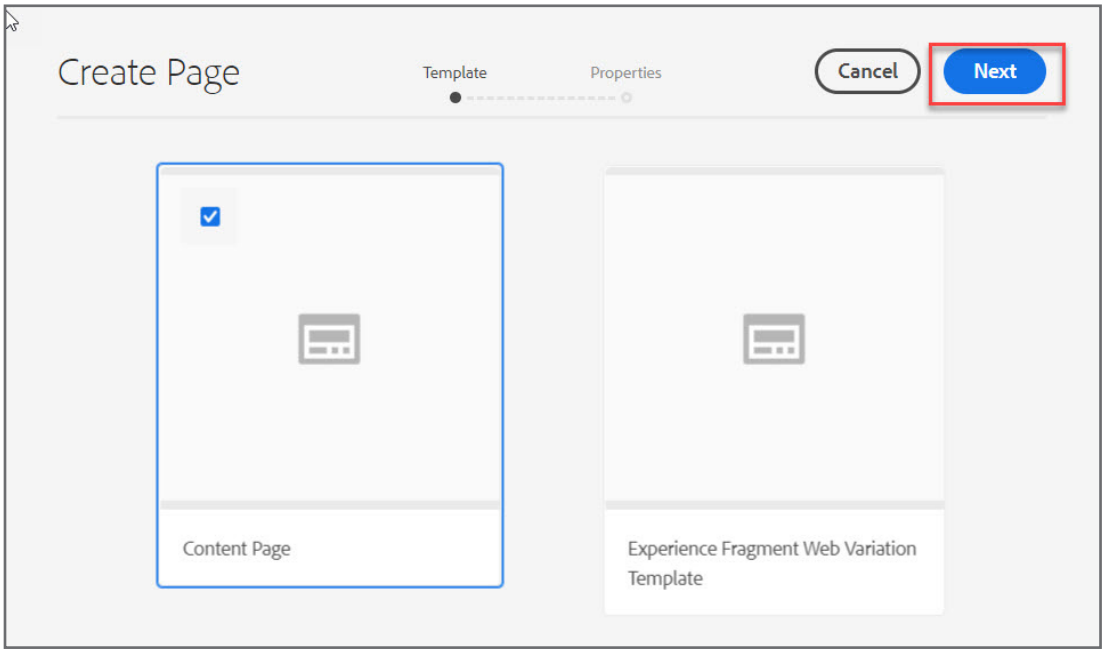
5. Right-click **training.core** and select **Run As** > **Maven install**. The project is built.

6. Verify that the **TitlePropertyListener** component is available in AEM:
   at: http://localhost:4502/system/console/components, as shown:

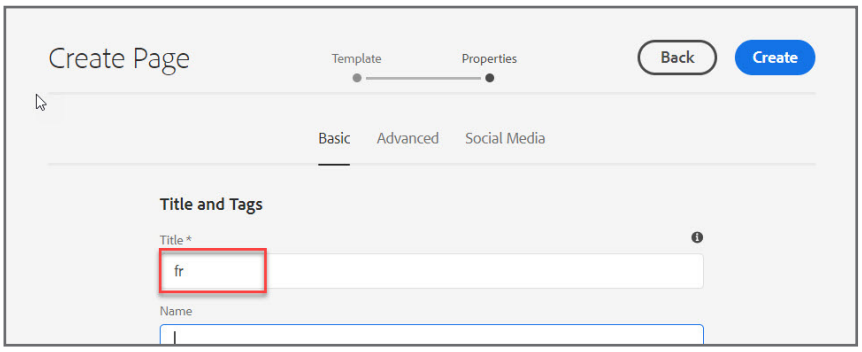


7. In your browser, navigate to **Sites**.
8. In the column view, click the right-pointing arrow next to **training** and navigate to **training** > **us,** as shown:

9. Click **Create** > **Page**. After you click **Page**, a wizard appears where you need to select a template for your page.
10. Click the **Content Page** template to select it, and then click **Next**, as shown.



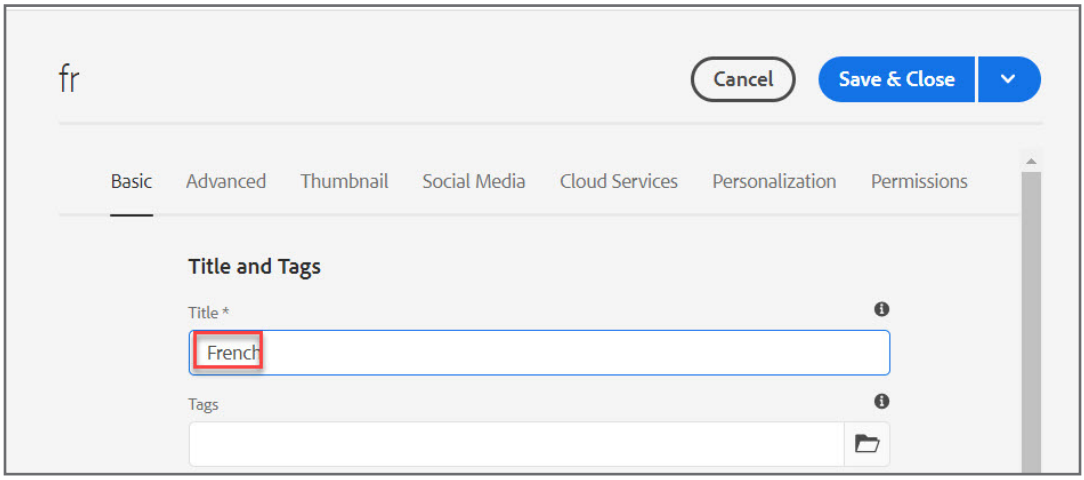
11. In the **Properties** step of the page creation wizard, provide the following value for the corresponding field:
    a. Title: **fr**



12. Click **Create** in the upper right to create the page. A **Success** dialog box is displayed with a message that your page has been created.
13. Click **Done**. The new page appears as a child page of the **us** Site, as shown.

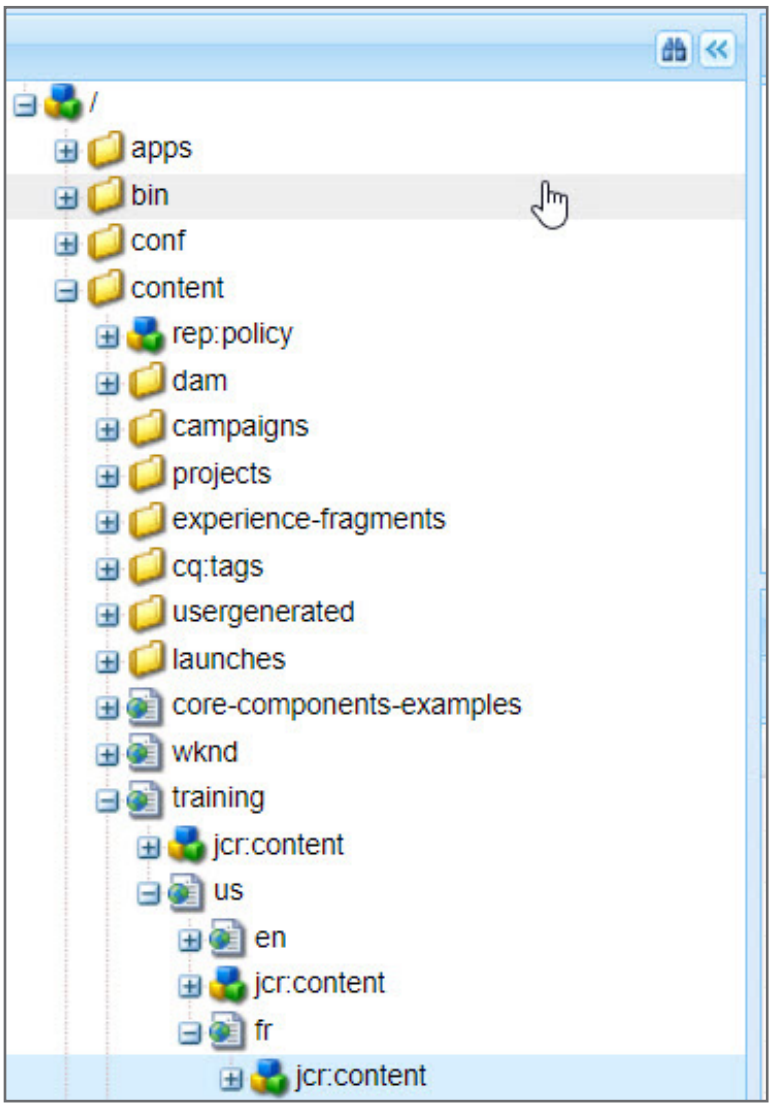
14. Select the **fr** page and click **Properties (p)** in the actions bar. The **Page Properties** window opens.

15. Change the **Title** to **French**, as shown:

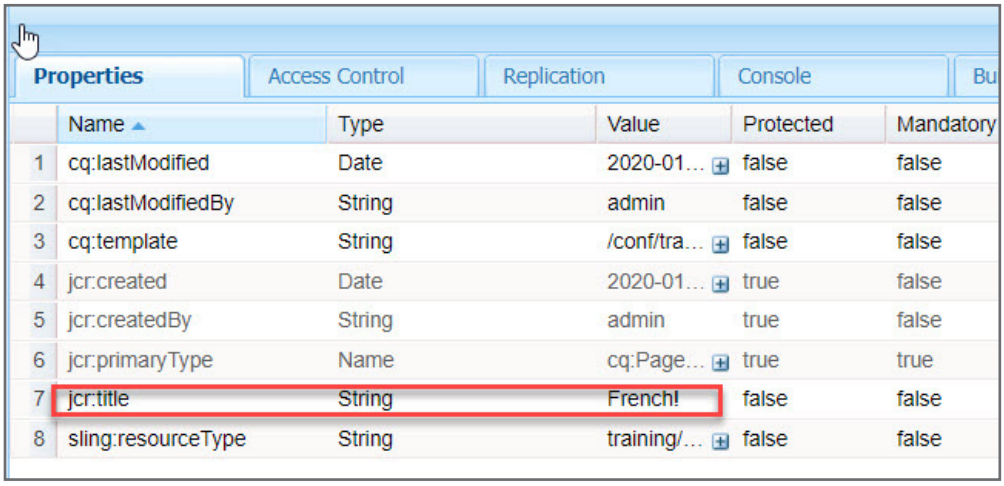


16. Click **Save & Close**. The page is reloaded with the name **French!,** as shown:

17. In CRXDE Lite, navigate to **/content/training/us/fr/jcr:content**, as shown:



18. Notice the value of the property **jcr:title**, as shown:



Optional Exercise 4: Use resource listener to control Stock schedulers

The first two exercises in this module created a scheduler and a job consumer to pull stock data and write it to the JCR on a regular schedule.

Each stock imported was configured by an OSGi config node that specified the symbol, schedule (by CRON), and source URL. Every new stock import needs to have a OSGi config.

In this optional exercise you will create a StockListener class that will listen for newly created folders under Sites > stocks and auto-create a new OSGi config for that stock symbol. It will also listen for deleted folders that will remove the OSGi config too.

Based on your knowledge of creating a Sling resource listener from the previous exercise, follow these sudo steps to create the stock resource listener:

1. In **Eclipse**, create a new Java class called **StockListener**.
2. You can find the Java class in your **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/listeners/ StockListener.java.**
3. Right-click **training.core** and select **Run As > Maven install**. The project is built.
4. Go to the browser that has the AEM author service running, and then go to **Sites** > **stocks.** Click **Create** > **Folder.**
5. Enter a stock symbol as the title (for example, GOOG, AMZN, APPL, or WDAY).
6. Click **Create**.

To test whether the listener is successful:

7. Go to the logs and observe the output.
8. Go to http://localhost:4502/system/console/configMgr and verify the new **Training Stock Importer** config.
9. Wait for 2 minutes and then go to **CRXDE Lite** > **/content/stocks/<yourSymbol>** to observe the imported data.
10. Similarly, delete a **stock** folder under **Sites** > **stocks** and verify the **Training Stock Importer** config is deleted.

References

For more information on the events in your AEM Web Console:

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

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

Access the data layer in AEM 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 Sling Models
- Create a Sling Model
- Discuss the Sling Model Exporter in AEM
- Extend a core component
- Configure queries to search resources

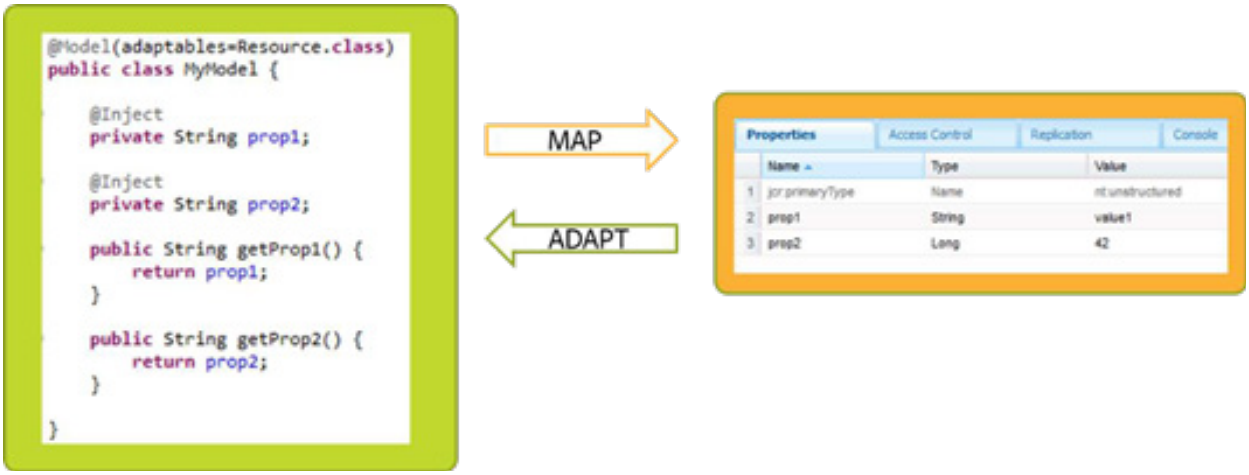
Working with Sling Models

Introduced with Sling 7 (2014), Sling Models allow you to define Java model objects (classes or interfaces) and map those objects to Sling resources (like a Java Content Repository (JCR) data structure or a SlingHttpServletRequest), or even to OSGi services, expanding the use of the adaptTo() method to any Java object.

A Sling Model can be seen as an abstraction layer that allows AEM components to consume back-end logic, just like the JS-Use, WCMUsePojo, or Sling Resource APIs would, but for complex components with the need for reusability.

Using Sling Models

A Sling Model is a Java class located in an OSGi bundle that is annotated with @Model and the adaptable class (for example, @Model(adaptables = Resource.class) for a resource).
In the example below, data members (the fields of the Java class) are annotated with @inject to map to node properties.



Benefits of using Sling Models

- Saves time from creating your own adapters (avoiding boilerplate code)
- Supports both classes and interfaces
- Allows you to adapt multiple objects—minimal required objects are 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

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 propertyName 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	@Named @Inject@ Named("title") private String page Title;	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.
@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

Sling Models are easily extensible, with custom injectors and 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 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 list of annotations maps 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

You can see a list of the available injectors at: http://localhost:4502/system/console/status-slingmodels

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.

Sling Model Exporter

Primarily used to easily expose data to HTML Template Language (HTL) scripts with minimal code, Sling Models are becoming a major development business object in AEM.

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, to be consumed by your front-end application (for example, headless Content Management Systems (CMSs)).

Apache Sling provides a Jackson JSON exporter to cover the most common case of exporting Sling Models as JSON objects. The exporter scans through all the getters and serializes them into JSON. The model needs to define the resourceType of the component as a modifier, in order to be bound to it.

The snippet below shows the declaration of a model for an AEM component, that gets exported with the Jackson exporter using the specific component adapter interface, com.adobe.cq.export.json. ComponentExporter:

```
@Model(adaptables=SlingHttpServletRequest.class,
    adapters= {ComponentExporter.class},
    resourceType=TitleWithSubtitle.RESOURCE_TYPE,
    defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL)
@Exporter(name = "jackson", extensions = "json")
public class TitleWithSubtitle implements ComponentExporter{
    protected static final String RESOURCE_TYPE = "trainingproject/components/content/titlewithsubtitle";
```

Exercise 1: Create a custom Sling Model

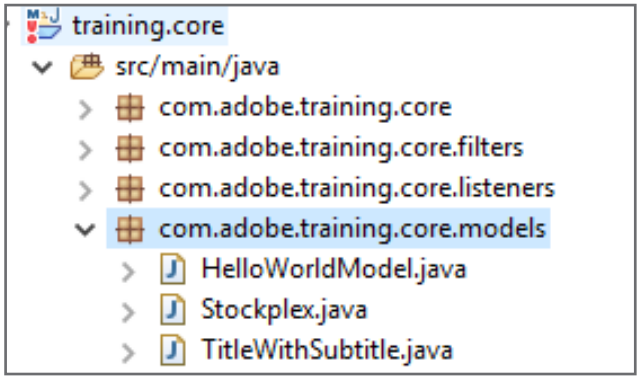
In this exercise, you will upload and install a stockplex component. This will be the front-end HTL code for your component. You will then write a Sling Model to support the business logic.

This exercise includes three tasks:

1. Create a Sling Model
2. Install the stockplex component
3. Test the component

Task 1: Create a Sling Model

1. In Project Explorer, navigate to **training.core** > **src/main/java**.
2. Copy the **Stockplex.java** file from **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/models/**.
3. Right-click **com.adobe.training.core.models** and select **Paste** to paste the file. The **Stockplex.java** class is created, as shown:



4. The **Stockplex.java** opens in the editor, as shown:

```
1.  package com.adobe.training.core.models;
2.
3.  import java.util.HashMap;
4.  import java.util.Map;
5.
6.  import javax.annotation.PostConstruct;
7.
8.  import org.apache.sling.api.SlingHttpServletRequest;
9.  import org.apache.sling.api.resource.Resource;
10. import org.apache.sling.api.resource.ValueMap;
11. import org.apache.sling.models.annotations.DefaultInjectionStrategy;
12. import org.apache.sling.models.annotations.Exporter;
13. import org.apache.sling.models.annotations.Model;
14. import org.apache.sling.models.annotations.injectorspecific.ResourcePath;
15. import org.apache.sling.models.annotations.injectorspecific.ScriptVariable;
16. import org.apache.sling.models.annotations.injectorspecific.ValueMapValue;
17.
18. import com.adobe.cq.export.json.ComponentExporter;
19. import com.adobe.training.core.StockDataWriterJob;
20. import com.day.cq.wcm.api.designer.Style;
21.
22.
23. /**
24.  * This model is used as the backend logic for the stockplex component. Using a Sling model allows the component
25.  * to be exportable via JSON for a headless scenarios. Stock data that this model uses is imported into the JCR
26.  * via StockImportScheduler.java
27.  *
28.  * The stock data that is expected is in the form:
29.  * /content/stocks
30.  * + ADBE [sling:OrderedFolder]
31.  *   + lastTrade [nt:unstructured]
32.  *     - companyName = <value>
33.  *     - sector = <value>
34.  *     - lastTrade = <value
35.  *     - ..
36.  *     LK, updated for GITHUB data, 190710
37.  */
38.
39. @Model(adaptables=SlingHttpServletRequest.class,
40.         adapters= {ComponentExporter.class},
41.         defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL,
42.         resourceType = Stockplex.RESOURCE_TYPE)
43. @Exporter(name="jackson", extensions = "json")
44. public class Stockplex implements ComponentExporter{
45.
46.     protected static final String RESOURCE_TYPE = "training/components/content/stockplex";
47.
48.     //HTL global object in the model
49.     //Learn more  at Helpx > HTL Global Objects
50.     @ScriptVariable
51.     private Resource resource;
52.
53.     @ScriptVariable
54.     private Style currentStyle;
55.
56.     //property on the current resource saved from the dialog of a component
57.     @ValueMapValue
58.     private String symbol;
59.
60.     //property on the current resource saved from the dialog of a component
61.     @ValueMapValue
62.     private String summary;
63.
64.     //property on the current resource saved from the dialog of a component
65.     @ValueMapValue
66.     private String showStockDetails;
67.
68.     //content root of for stock data. /content/stocks
69.     @ResourcePath(path = StockDataWriterJob.STOCK_IMPORT_FOLDER)
70.     private Resource stocksRoot;
71.
72.     private double currentPrice;
73.     private Map<String,Object> data;
74.
75.     @PostConstruct
```

```
76.     public void constructDataMap() {
77.         ValueMap tradeValues = null;
78.
79.         //Check to see if stock data has been imported into the JCR
80.         if(stocksRoot != null) {
81.             Resource stockResource = stocksRoot.getChild(symbol);
82.             if(stockResource != null) {
83.                 Resource lastTradeResource = stockResource.getChild("trade");
84.                 if(lastTradeResource != null){
85.                     tradeValues = lastTradeResource.getValueMap();
86.                 }
87.             }
88.         }
89.
90.         data = new HashMap<>();
91.         //If stock information is in the JCR, display the data
92.         if(tradeValues != null) {
93.             currentPrice = tradeValues.get(StockDataWriterJob.LASTTRADE, Double.class);
94.             data.put("Request Date", tradeValues.get(StockDataWriterJob.DAYOFUPDATE, String.class));
95.             data.put("Request Time", tradeValues.get(StockDataWriterJob.UPDATETIME, String.class));
96.             data.put("UpDown", tradeValues.get(StockDataWriterJob.UPDOWN, Double.class));
97.             data.put("Open Price", tradeValues.get(StockDataWriterJob.OPENPRICE, Double.class));
98.             data.put("Range High", tradeValues.get(StockDataWriterJob.RANGEHIGH, Double.class));
99.             data.put("Range Low", tradeValues.get(StockDataWriterJob.RANGELOW, Double.class));
100.            data.put("Volume",  tradeValues.get(StockDataWriterJob.VOLUME, Integer.class));
101.            data.put("Company", tradeValues.get(StockDataWriterJob.COMPANY, String.class));
102.            data.put("Sector", tradeValues.get(StockDataWriterJob.SECTOR, String.class));
103.            data.put("52 Week Low", tradeValues.get(StockDataWriterJob.WEEK52LOW, Double.class));
104.        } else {
105.            data.put(symbol,"No import config found. If the StockListener.java class is apart of your
106. project: Go to Sites console > Create Folder: stocks > Create Folder: ADBE");
107.        }
108.    }
109.
110.    /**
111.     * All getter methods below will be apart of the output by the JSON Exporter
112.     */
113.    //Getter for dialog input
114.    public String getSymbol() {
115.        return symbol;
116.    }
117.    //Getter for dialog input
118.    public String getSummary() {
119.        return summary;
120.    }
121.    //Getter for dialog input
122.    public String getShowStockDetails() {
123.        return showStockDetails;
124.    }
125.
126.    //Calculated current price based on imported stock info
127.    public Double getCurrentPrice() {
128.        return currentPrice;
129.    }
130.    //Calculated trade values based on imported stock info
131.    public Map<String,Object> getData() {
132.        return data;
133.    }
134.    @Override
135.    public String getExportedType() {
136.        return resource.getResourceType();
137.    }

138. }
```

5. Examine the above code. Notice it is using @Model annotation and it is also exportable as JSON.

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

7. Verify the bundle installed successfully.

To observe the Sling Model that you installed:

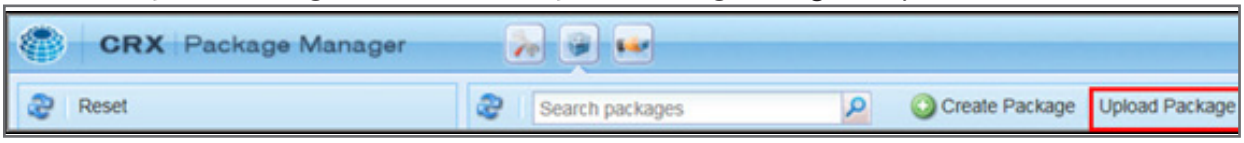
8. Navigate to AEM **Web Console**. Select **Status** > **Sling Models** and search for **stockplex**.

You should notice the **stockplex** model and also the **stockplex** component are exportable as JSON.

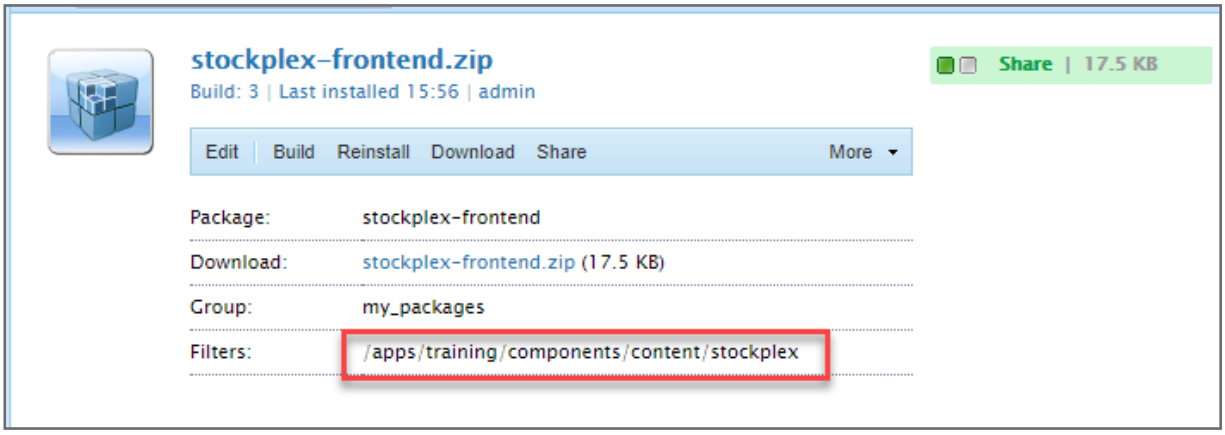
Task 2: Install the stockplex component

Now that we have the Sling model installed, we are going to install the component that uses the Sling model, the **stockplex** component. This component is built out in detail in the "Develop Websites and Components in Adobe Experience Manager" course.

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.

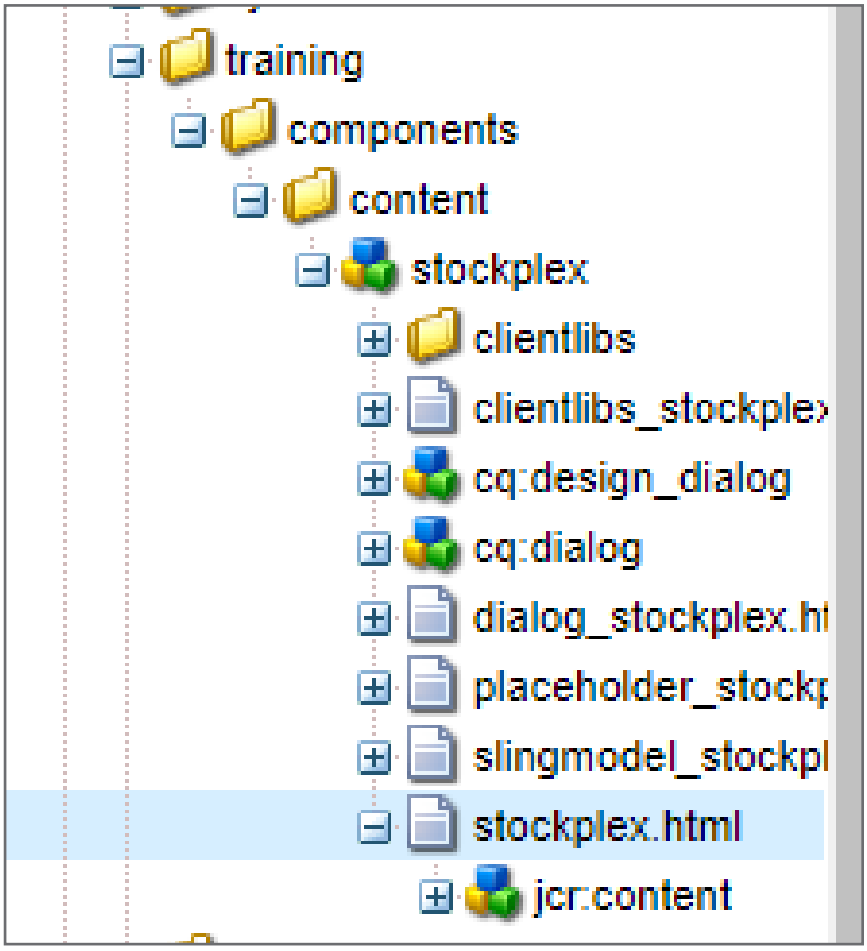


3. In the Upload Package dialog box, click **Browse**, and select the **stockplex-frontend.zip** package file from **Exercise_Files** under **/training-files/Sling-Models**. Click **Open**.
4. Click **OK**.
5. Verify the uploaded package is now available in **CRX Package Manager**.
6. Click **Install**.
7. When the **Install Package** window opens, leave **Advanced Settings** as-is and click Install. The package is installed.
8. Click the Develop icon at the top of CRXDE Lite. Verify the package is now installed under / **apps/training/components/content/stockplex:**

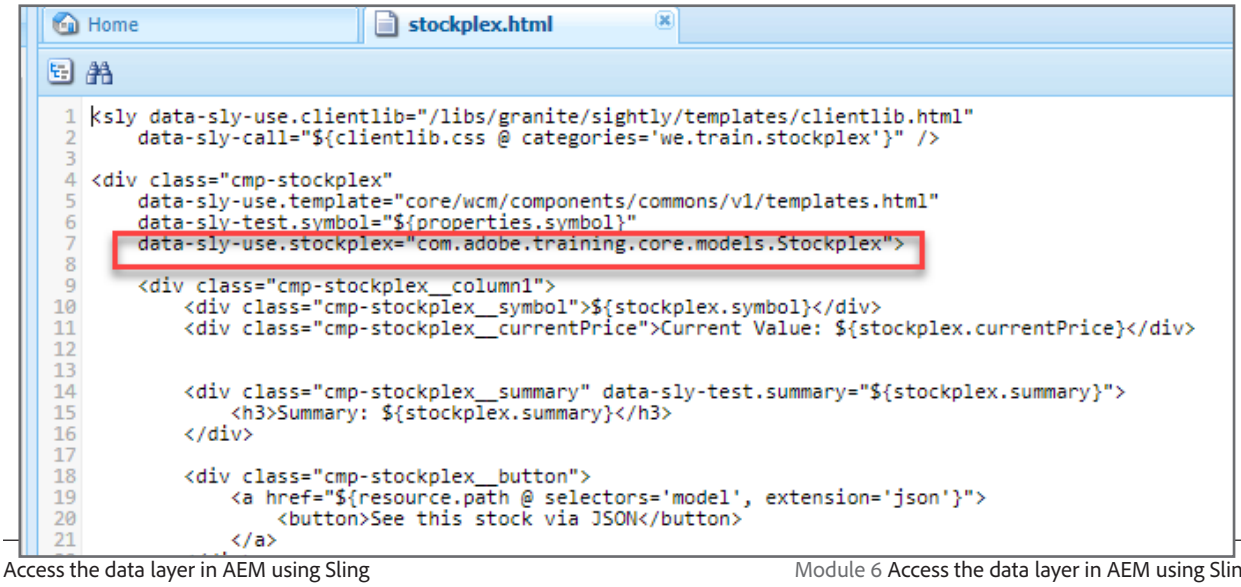


9. In CRXDE Lite, navigate to **/apps/training/components/content/stockplex**.

10. Click the + icon to expand the **stockplex** node and double-click **stockplex.html** to open it in the editor.

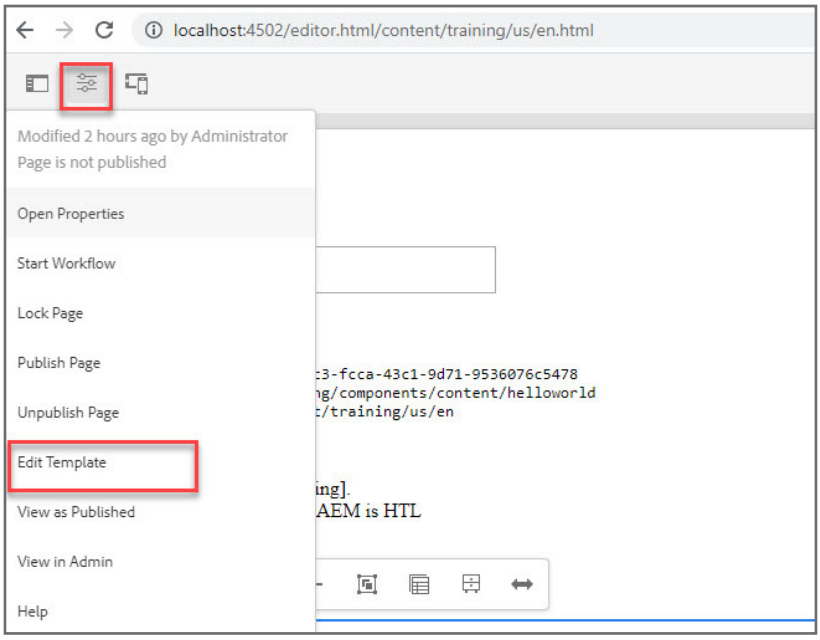


11. In the script, notice how it is requesting your Sling model stockplex on line 7 , as shown:

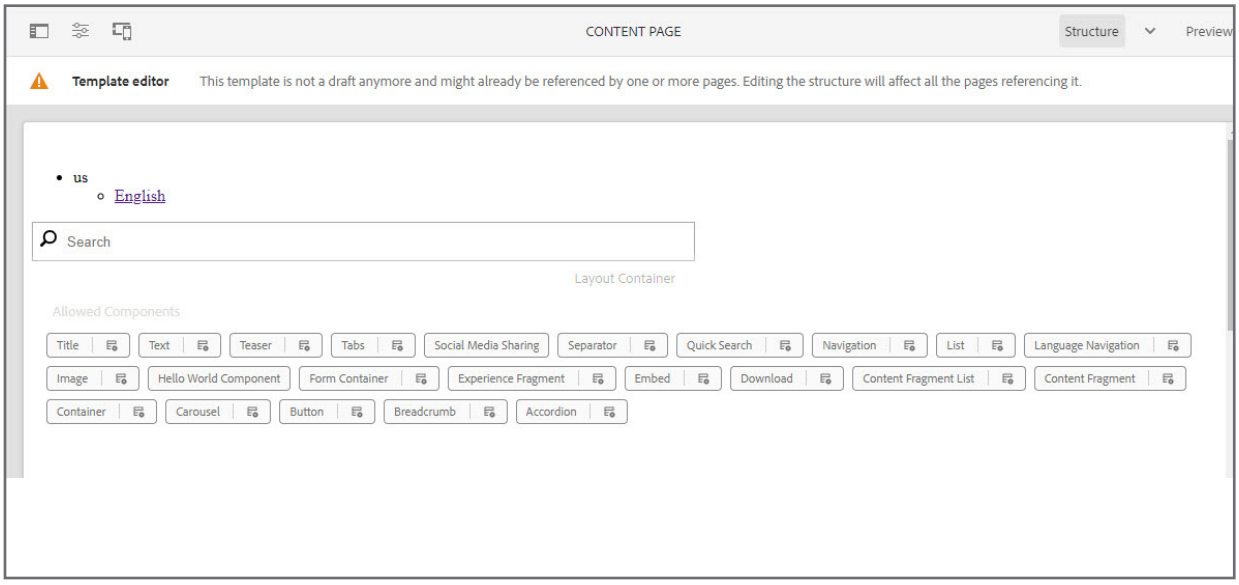


Task 3: Test the stockplex component

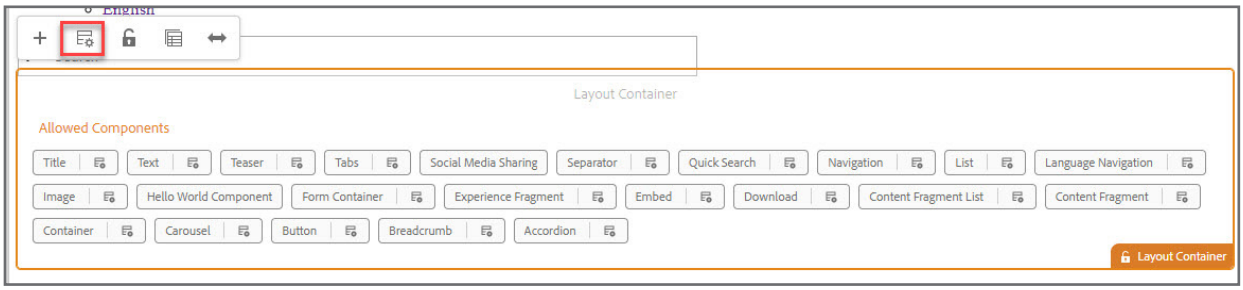
1. Go to: http://localhost:4502/editor.html/content/training/us/en.html
2. Click the **Page information** icon > **Edit Template**, as shown:



The **Template Editor** opens, as shown:

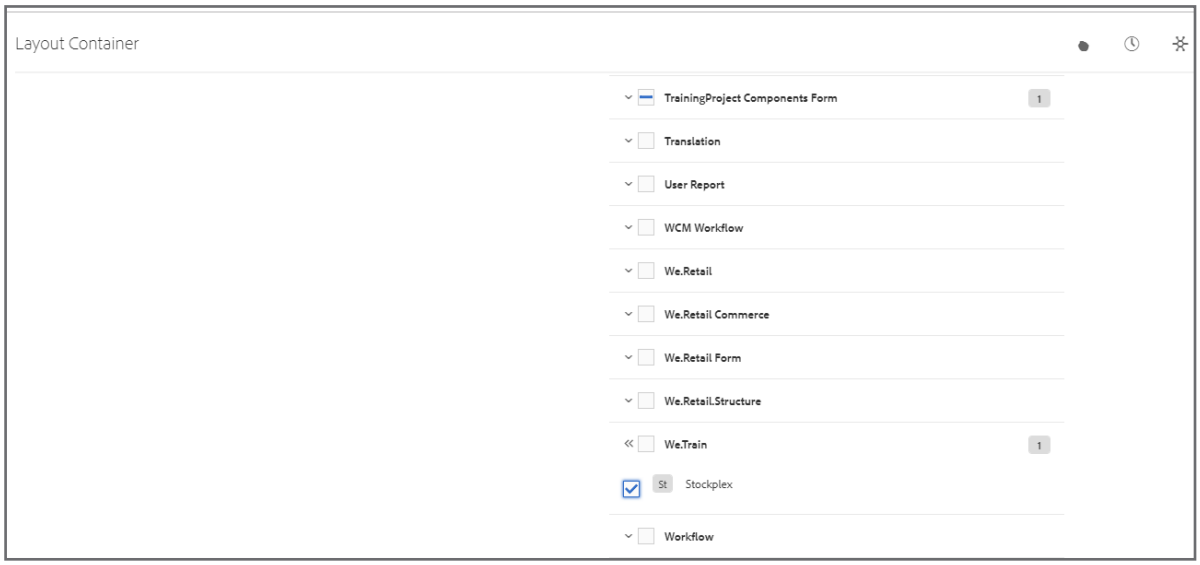


3. Select the **Layout Container** and click on the **Policy** icon, as shown:



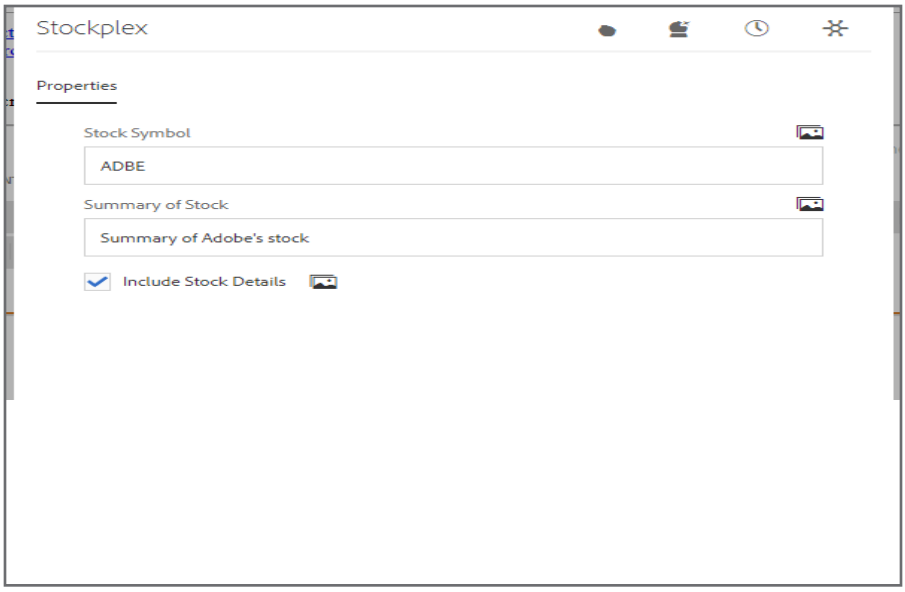
The Layout Container window opens.

4. Under the **Allowed Components** tab, select **We.train** > **Stockplex**, as shown:



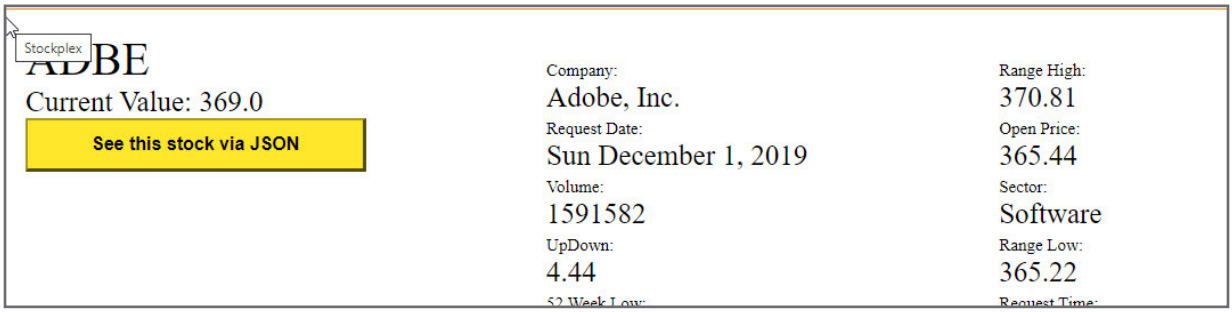
5. Click **Done** (checkmark icon). The changes are saved.
6. Navigate to http://localhost:4502/editor.html/content/training/us/en.html
7. Click **Toggle Side Panel** (if not already done).
8. In the left pane, click the **Components** icon. Use the Filter option to find **Stockplex** component. If you cannot see it, refresh the page by pressing **Ctrl+Shift+R**.
9. Drag and drop the **Stockplex** component onto your page.
10. Double-click the **Stockplex** components box to open it.

11. In the **Stock Symbol** field, type **ADBE** and select **Include Stock Details** ,as shown:



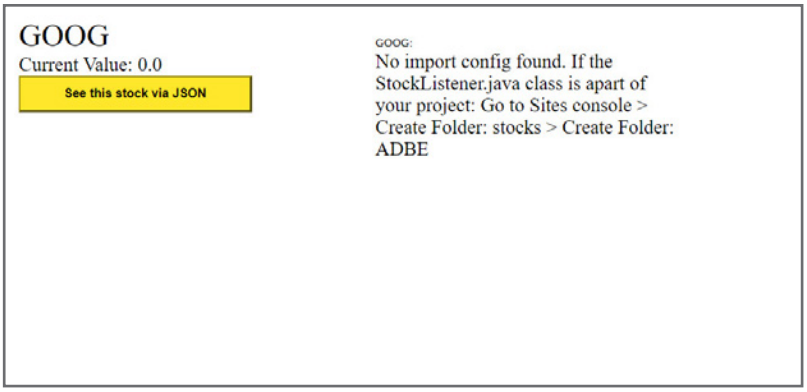
12. Click **Done** to save your changes.

13. Refresh the page. Notice the Sling Models goes into the JCR, finds the ADBE stock information that we imported earlier, and displays it onto the component, as shown:



14. Drag and drop another **Stockplex** component onto the page.
15. Configure the stockplex component to add a stock that is not in the JCR. For example, GOOG, as shown:

16. Click **Done** to save your changes.
17. Refresh the page. Notice the stock data is not imported since the stock data is not stored in the JCR, as shown:



Note: Since you uploaded the **stockplex** component via a content package, it is not a part of your maven project. Make sure to sync your new component back to your maven project using I**mport from Server** in Eclipse

Exercise 2: Extend a core component

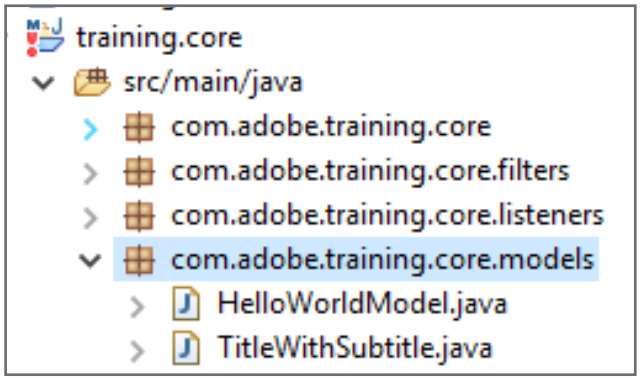
The AEM core components are building blocks for modern web development. Sometime these core components need more functionality and properly extending them is vital to ensure upgradability and maintainability. In this exercise you will learn how to properly extend a sling model that is the business logic for a core component. You will also upload and observe an extended component using your new Sling Model.

This exercise includes three tasks:

1. Install the Java code
2. Install the content package
3. Test the component

Task 1: Install the Java code

1. In Project Explorer, navigate to **training.core** > **src/main/java**.
2. Copy the **TitleWithSubtitle.java** file from **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/models/**.
3. Right-click **com.adobe.training.core.models** and select **Paste**. The **TrainingWithSubtitle.java** class is created, as shown:



4. The **TitleWithSubtitle.java** opens in the editor, as shown:

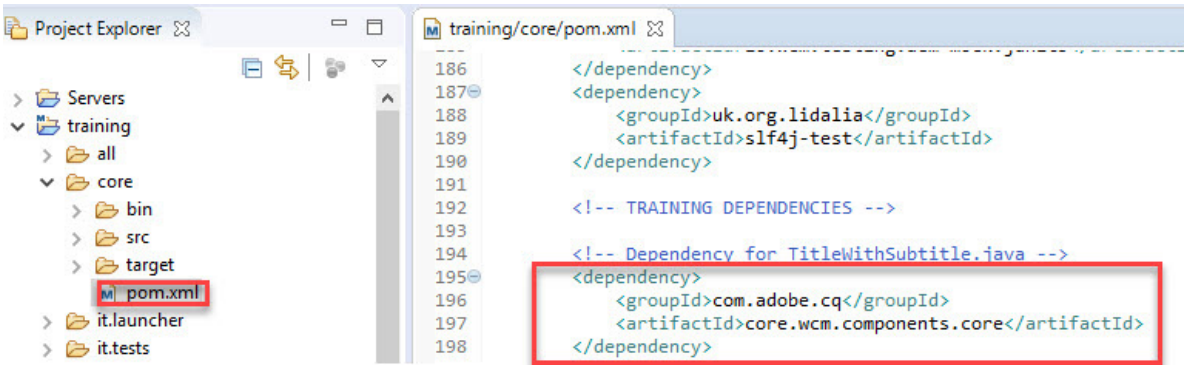
```
1.  package com.adobe.training.core.models;
2.
3.  import javax.annotation.PostConstruct;
4.  import javax.inject.Inject;
5.  import javax.inject.Named;
6.
7.  import org.apache.sling.api.SlingHttpServletRequest;
8.  import org.apache.sling.models.annotations.DefaultInjectionStrategy;
9.  import org.apache.sling.models.annotations.Exporter;
10. import org.apache.sling.models.annotations.Model;
11. import org.apache.sling.models.annotations.Via;
12. import org.apache.sling.models.annotations.injectorspecific.ScriptVariable;
13. import org.apache.sling.models.annotations.injectorspecific.Self;
14. import org.apache.sling.models.annotations.injectorspecific.SlingObject;
15. import org.apache.sling.models.annotations.injectorspecific.ValueMapValue;
16. import org.apache.sling.models.annotations.via.ResourceSuperType;
17. import org.slf4j.Logger;
18.
19. import com.adobe.cq.export.json.ComponentExporter;
20. import com.adobe.cq.wcm.core.components.models.Title;
21. import com.day.cq.wcm.api.Page;
22.
23. /**
24.  * This title model extends the core title model: com.adobe.cq.wcm.core.components.models.Title
25.  * The core title model is used in both v1 and v2 core title components: /apps/core/wcm/components/title
26.  * In this model, we extend the title model to include a subtitle.
27.  *
28.  * In order to extend a core component model, in core/pom.xml add the dependency:
29.         <dependency>
30.             <groupId>com.adobe.cq</groupId>
31.             <artifactId>core.wcm.components.core</artifactId>
32.         </dependency>
33.  *
34.  * Note: You don›t need to add this dependency to the parent pom because it›s already included.
35.  */
36.
37. @Model(adaptables=SlingHttpServletRequest.class,
38.     adapters= {ComponentExporter.class},
39.     resourceType=TitleWithSubtitle.RESOURCE_TYPE,
40.     defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL)
41. @Exporter(name = "jackson", extensions = "json")
42. public class TitleWithSubtitle implements ComponentExporter{
43.     protected static final String RESOURCE_TYPE = "training/components/content/titlewithsubtitle";
44.
45.     @Inject
46.     @Named("log")
47.     private Logger logger;
48.
49.     @SlingObject
50.     private SlingHttpServletRequest request;
51.
52.     //HTL global object in the model
53.     //Learn more  at Helpx > HTL Global Objects
54.     @ScriptVariable
55.     private Page currentPage;
56.
57.     //property on the current resource saved from the dialog of a component
58.     @ValueMapValue
59.     private String subtitle;
60.
61.     //Core Title model we are extending
62.     @Self @Via(type = ResourceSuperType.class)
63.     private Title coreTitle;
64.
65.     //Method called when the model is initialized
66.     @PostConstruct
67.     protected void initModel(){
68.
69.         //setup properties that are extending the title model
70.         if(subtitle == null){
71.             subtitle = "";
72.         }
73.     }
74.     //Next 2 methods required to support JSON display for this Component
75.     public String getTitle() {
76.         return coreTitle.getText();
77.     }
```

```
78.
79.     public String getLinkURL() {
80.         return coreTitle.getLinkURL();
81.     }
82.
83.     public String getSubtitle() {
84.         return subtitle;
85.     }
86.
87.     public boolean isEmpty() {
88.         //Verify there is a subtitle
89.         if(!subtitle.isEmpty()) {
90.             return false;
91.         }
92.         //Verify a title was entered from the dialog and
93.         //Page.title or Page.PageTitle are not being used
94.         String uniqueTitle = coreTitle.getText();
95.         if (!uniqueTitle.equals(currentPage.getTitle())
96.             && !uniqueTitle.equals(currentPage.getPageTitle())) {
97.             return false;
98.         }
99.         return true;
100.    }
101.
102.    @Override
103.    public String getExportedType() {
104.        return request.getResource().getResourceType();
105.    }
106. }
```

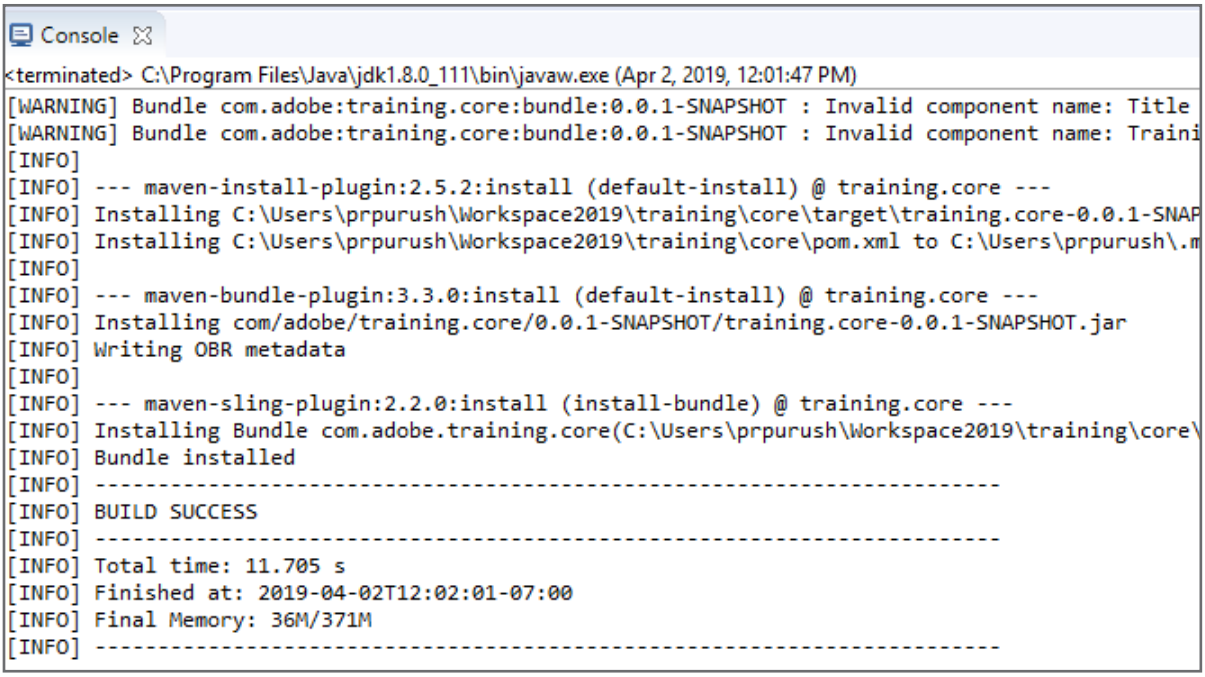
5. In order to develop against the core components Sling Models, we need to add a dependency to the core components bundle. Open **training** > **core** > **pom.xml**. At the bottom of the pom, add the dependency as shown.

Note: You can get this dependency to copy and paste from the comments in the TitleWithSubtitle.java class:

```
<dependency>
    <groupId>com.adobe.cq</groupId>
    <artifactId>core.wcm.components.core</artifactId>
</dependency>
```



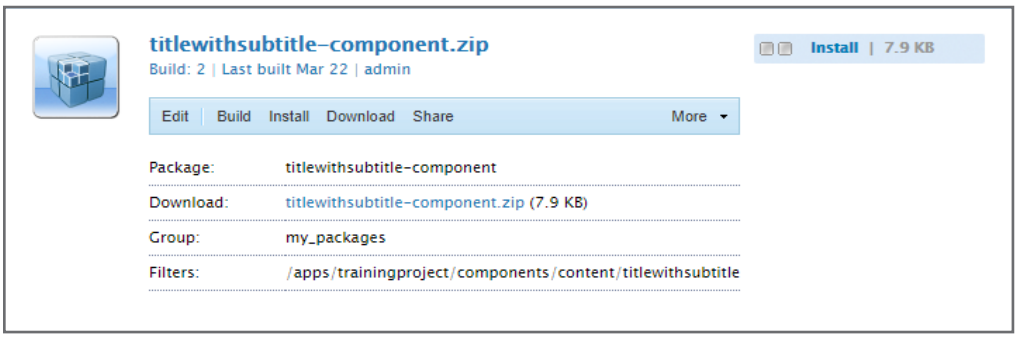
6. Press **Ctrl+S** (or click **File** > **Save**) to save your chages.
7. Right-click **training.core** and select **Run As** > **Maven install** to build the project. The build starts.
8. Verify the bundle installed successfully, as shown:



Task 2: Install the content package

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.

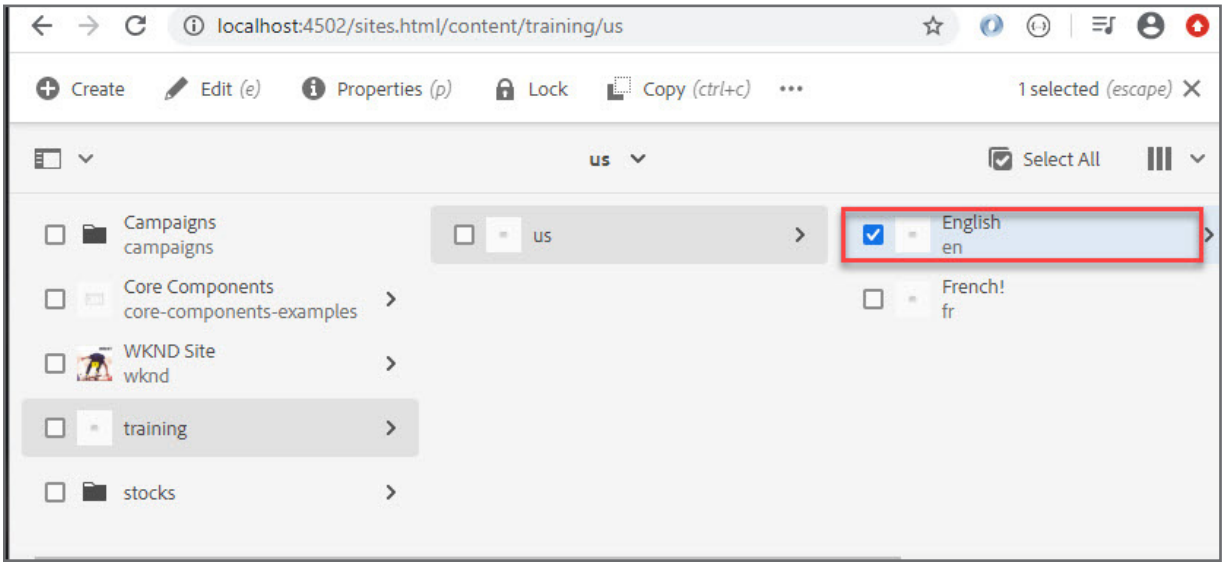
3. In the **Upload Package** dialog box, click **Browse**, and select the **titlewithsubtitle-component.zip** package file from the **Exercise_Files** under **/training-files/Sling-Models/**. Click **Open**.
4. Click **OK**.
5. Verify the uploaded package is now available in CRX Package Manager, as shown:



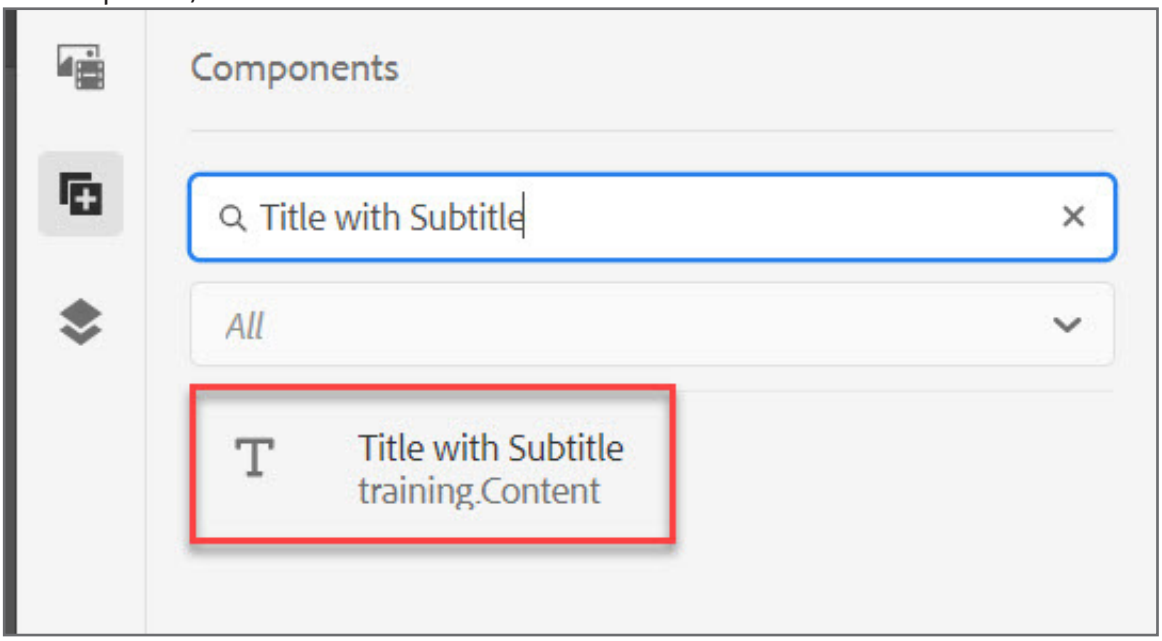
6. Click **Install**.
7. When the **Install Package** window appears, leave the **Advanced Settings** as-is and click **Install**.

Task 3: Test the component

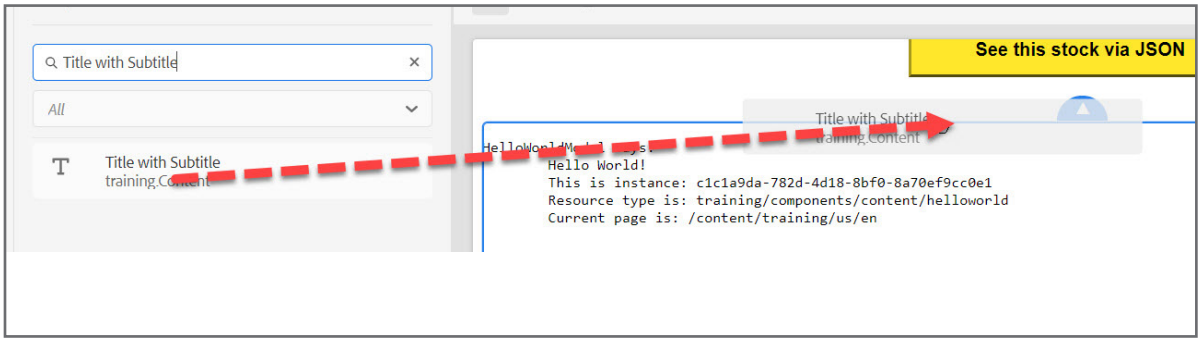
1. Go to the **Sites** Console, http://localhost:4502/sites.html/content.
2. Navigate to **Sites** > **training Site**, and click the thumbnail on **English** and select the page, as shown:



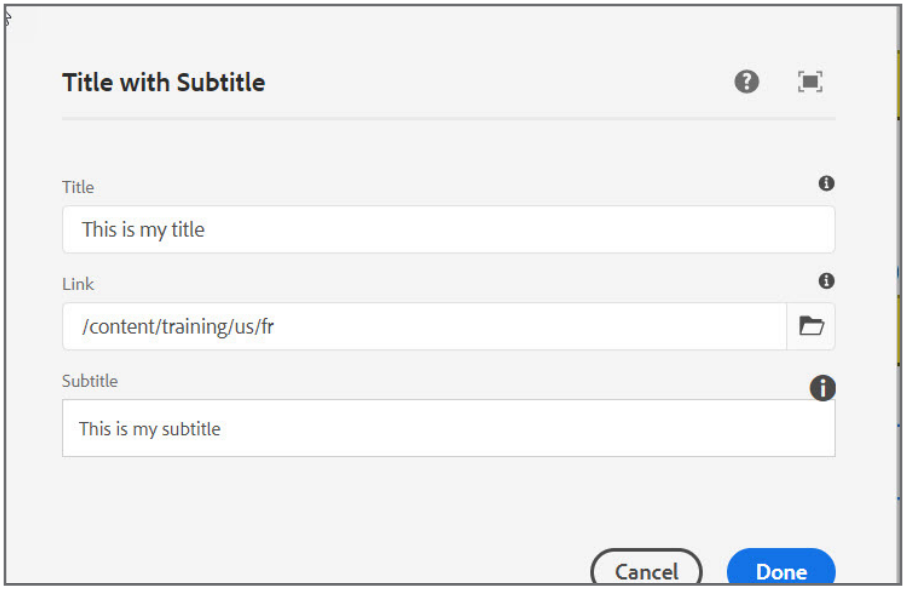
3. Click **Edit (*e*).** The page opens in a new tab for editing.
4. In the left pane, click the **Components** icon. Use the Filter option to find the **TitlewithSubtitle** component, as shown:



5. Click and drag the **Title with Subtitle** onto the Drag components here container, as shown:

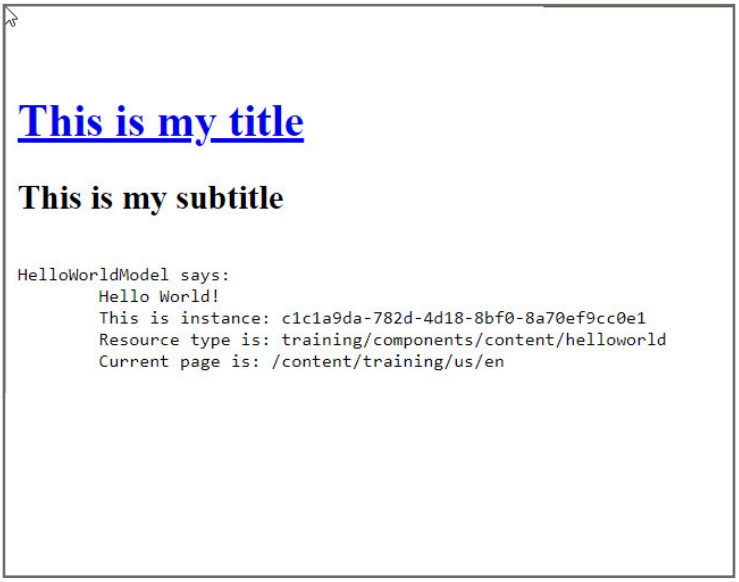


6. Double-click the **Title with Subtitle** components box to open it.
7. Add a **Title**, **Link**, and **Subtitle**, as shown:

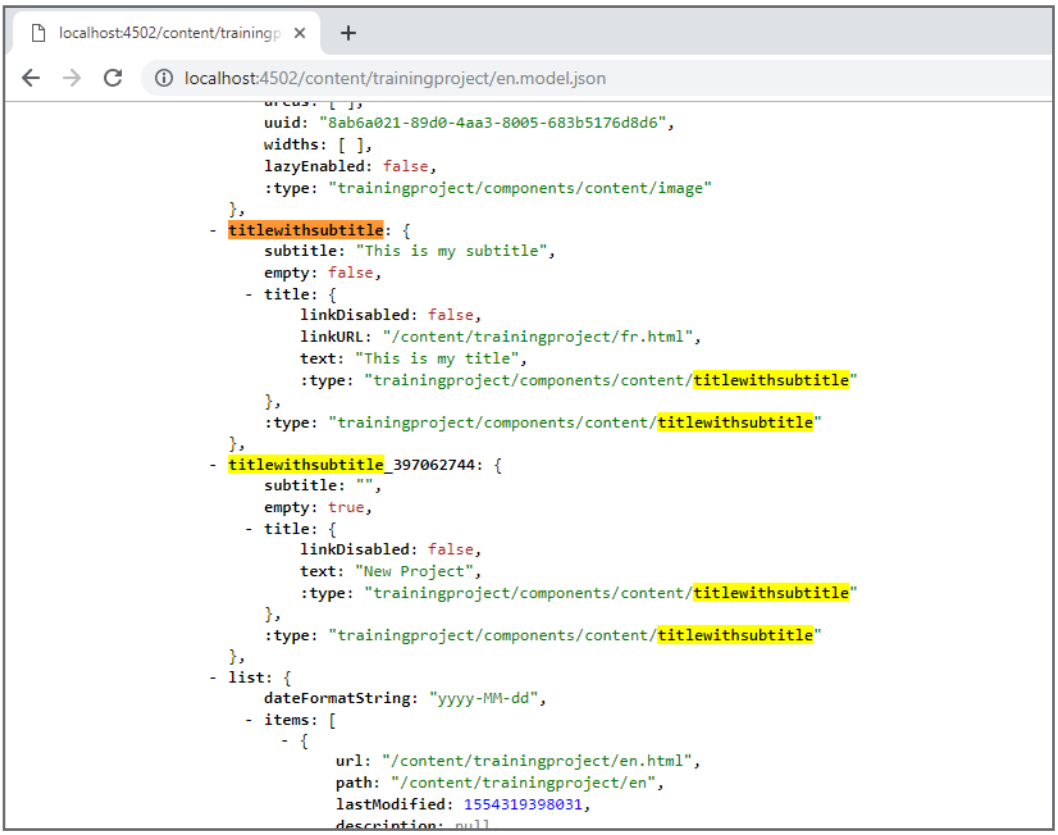


8. Click **Done** to save the changes.

9. Verify the changes, as shown:



10. Click the **Page information** icon > **View as Published**. The page gets published.
11. Verify the component exported as JSON by using the URL, as shown: http://localhost:4502/content/training/us/en.model.json
12. Search for **titlewithsubtitle** to find your component. Notice your model returned the model of the title component wrapped with the special values.



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();

  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])
  ```

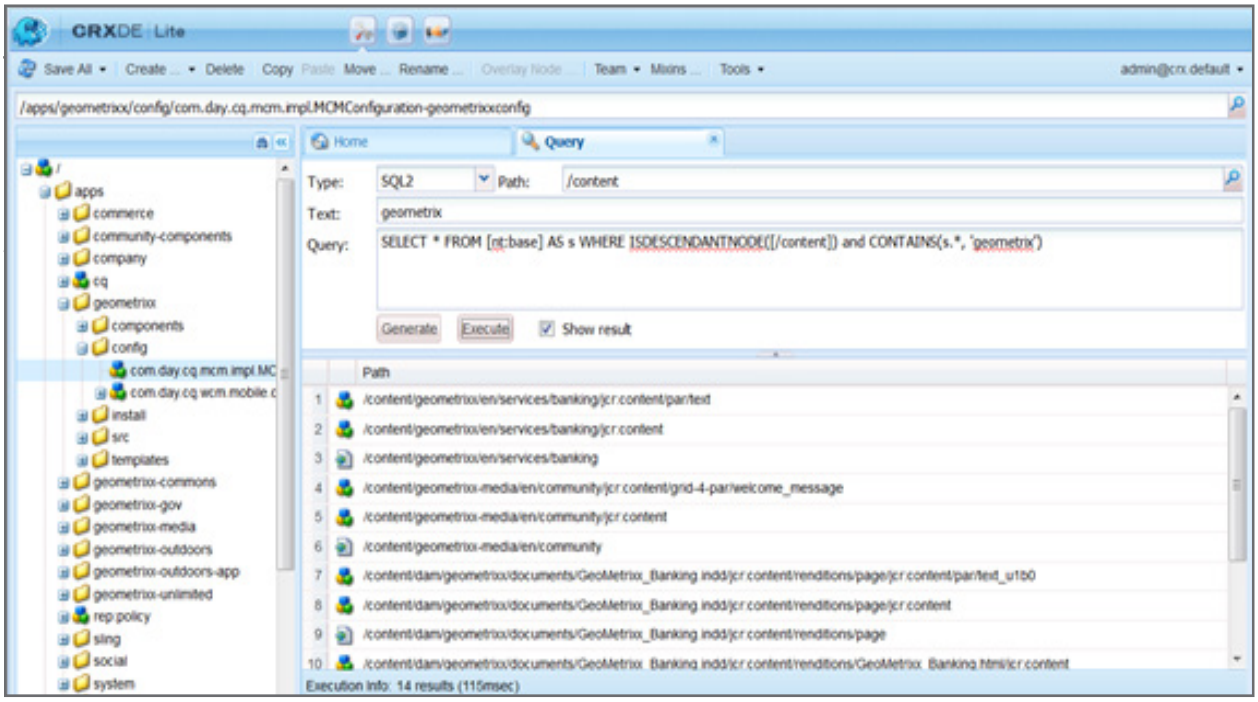
AND (NOT ISDESCENDANTNODE(file, [/var/classes]))

ORDER BY file.[jcr:createdBy], file.[jcr:created]

Search Performance

Fastest JCR search methodologies:

- Constraints on properties, Node types, and full-text
- Typically O(n), where n is the number of results, versus to the total number of nodes
- Constraints on the path



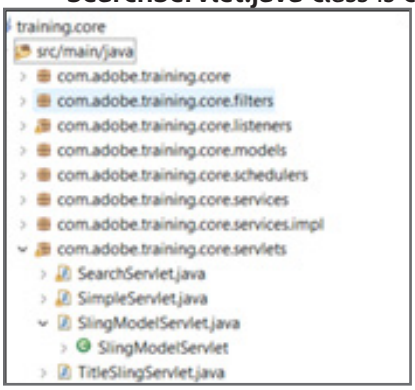
Exercise 3: Search resources with queries

In this exercise, you will do a 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**.
2. Copy the **SearchServlet.java** file from **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/servlets/**.

Note: The code is provided as part of the **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/servlets/**. 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 for illustrative purposes only.

3. Right-click **com.adobe.training.core.servlets** and select **Paste** to paste the file. The **SearchServlet.java** class is created, as shown:



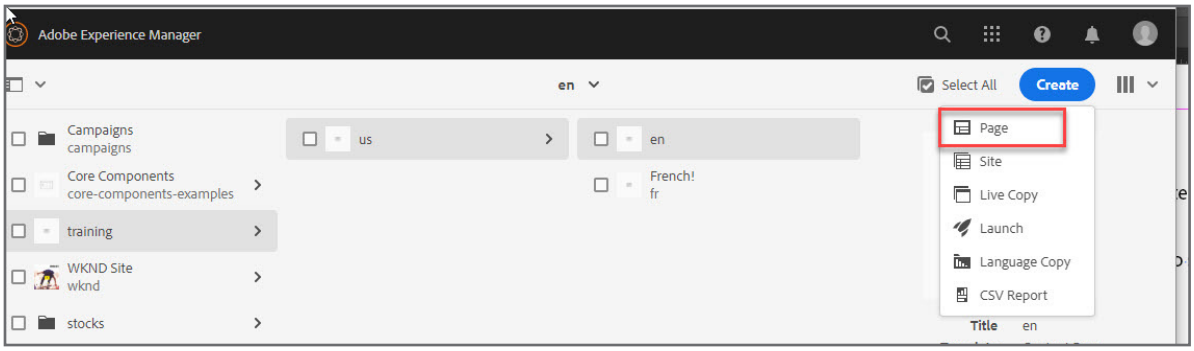
4. The **SearchServlet.java** opens in the editor, as shown:

```
1.  package com.adobe.training.core.servlets;
2.
3.  import com.day.cq.wcm.api.PageManager;
4.  import com.fasterxml.jackson.databind.ObjectMapper;
5.
6.  import org.apache.sling.api.SlingHttpServletRequest;
7.  import org.apache.sling.api.SlingHttpServletResponse;
8.  import org.apache.sling.api.resource.Resource;
9.  import org.apache.sling.api.resource.ResourceResolver;
10. import org.apache.sling.api.servlets.SlingSafeMethodsServlet;
11.
12. import org.osgi.service.component.annotations.Component;
13. import org.slf4j.Logger;
14. import org.slf4j.LoggerFactory;
```

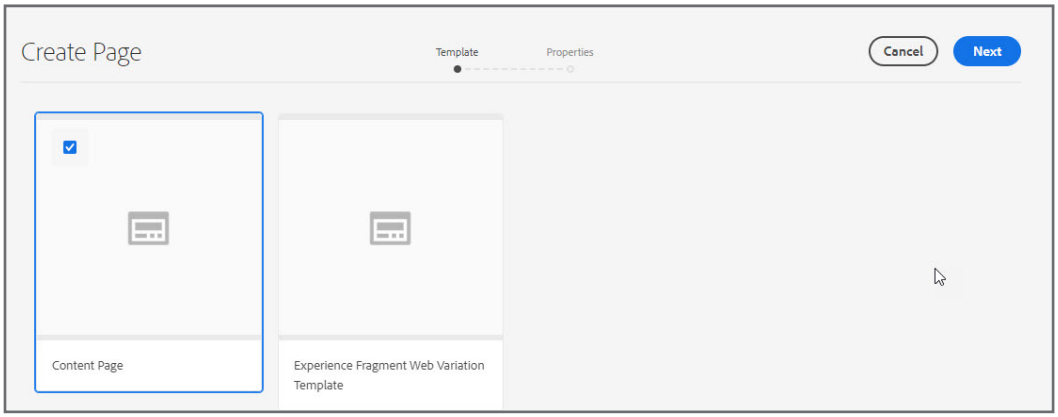
```
15.
16. import javax.jcr.query.Query;
17.
18. import javax.servlet.Servlet;
19. import javax.servlet.ServletException;
20. import java.io.IOException;
21. import java.util.ArrayList;
22. import java.util.Iterator;
23.
24. /**
25.  *
26.  * Example URI: http://localhost:4502/content/training/us/en.search.html?q=New%20Project
27.  * Add a Title component with «Lorem Ipsum» and search http://localhost:4502/content/training/us/en.search.
    html?q=Lorem
28.  *
29.  */
30. @Component(service = Servlet.class,
31.         property = {"sling.servlet.resourceTypes=training/components/page",
32.                 "sling.servlet.selectors=search"})
33. public class SearchServlet extends SlingSafeMethodsServlet {
34.     private static final long serialVersionUID = 1L;
35.
36.     private final transient Logger logger = LoggerFactory.getLogger(getClass());
37.
38.     @Override
39.     public final void doGet(final SlingHttpServletRequest request, final SlingHttpServletResponse response)
40.             throws ServletException, IOException {
41.         response.setHeader("Content-Type", "application/json");
42.
43.         ArrayList<Object> resultArray = new ArrayList<>();
44.         ObjectMapper objMapper = new ObjectMapper();
45.
46.         try (ResourceResolver rr = request.getResourceResolver()) {
47.
48.             // Path of the node, which triggers this servlet
49.             Resource requestingResource = request.getResource();
50.
51.             // Then adapt the resource tree into a page tree
52.
53.             PageManager pageManager = rr.adaptTo(PageManager.class);
54.
55.             // Now we can find the page, that was initiating the search
56.             String queryingPagePath = pageManager.getContainingPage(requestingResource).getPath();
57.
58.             String queryTerm = (request.getParameter("q") != null) ? request.getParameter("q") : "";
59.
60.             String QUERY_STRING = "SELECT * " +
61.                     "FROM [nt:unstructured] AS node " +
62.                     "WHERE ISDESCENDANTNODE([" + queryingPagePath + "]) " +
63.                     "and CONTAINS(node.*, '" + queryTerm + "')";
64.
65.             Iterator<Resource> resourcesIterator = rr.findResources(QUERY_STRING, Query.JCR_SQL2);
66.
67.             //Add the search results into the json array
68.             while (resourcesIterator.hasNext()) {
69.                 Resource foundResource = resourcesIterator.next();
70.                 resultArray.add(foundResource.getPath());
71.             }
72.
73.             String json = objMapper.writerWithDefaultPrettyPrinter().writeValueAsString(resultArray);
74.
75.             response.getWriter().print(json);
76.             response.getWriter().close();
77.
78.         } catch (IOException e) {
79.             logger.error("Could not retrieve the resourceResolver. ", e);
80.         }
81.     }
82. }
```

5. Right-click **training.core** and select **Run As** > **Maven install** to build the project. The build starts.

6. Verify the bundle installed successfully.
7. In your browser, navigate to **Sites**.
8. In the column view, click the right-pointing arrow next to **training** and navigate to **training** > **us** > **en.**
9. Click **Create** > **Page**. After you click **Page**, a wizard appears where you need to select a template for your page.

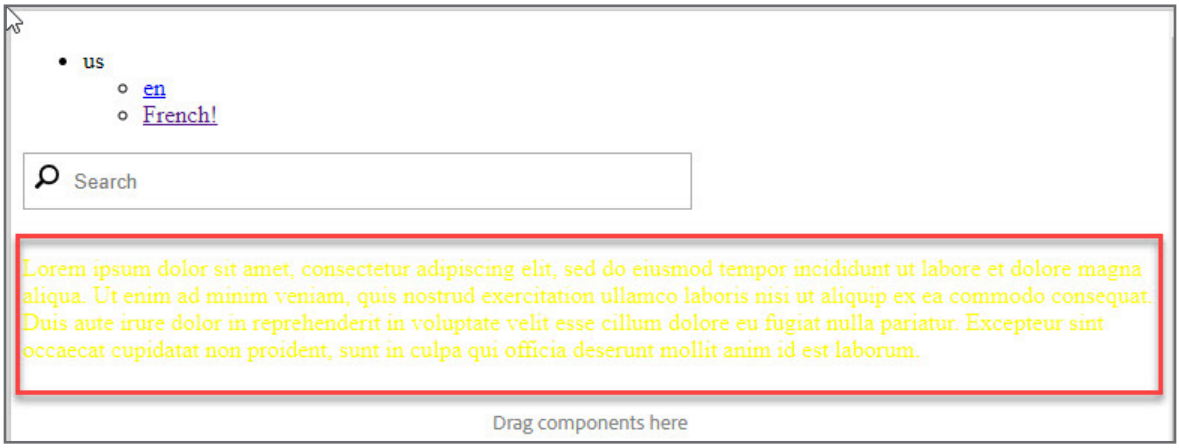


10. Click the **Content Page** template to select it, and then click **Next**.

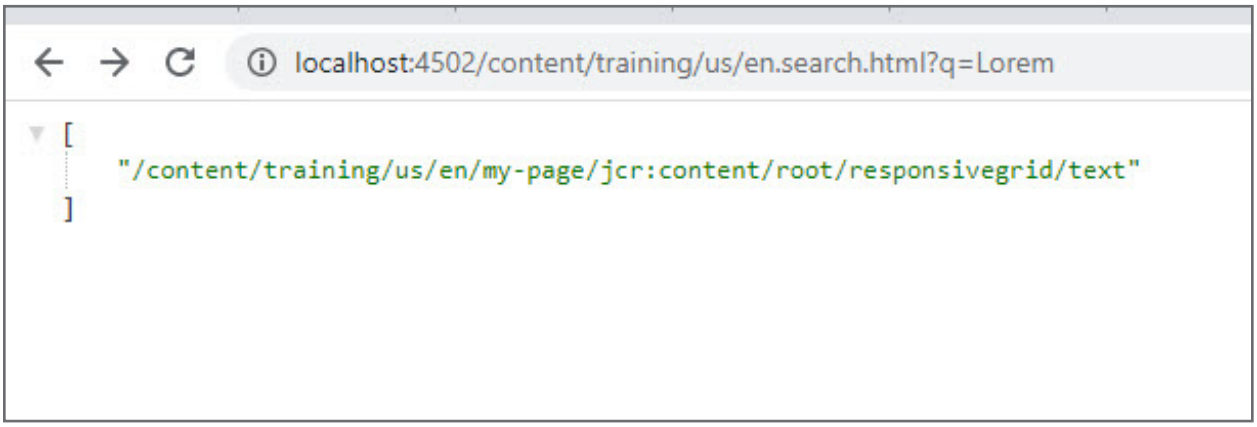


11. In the **Properties** step of the page creation wizard, provide the following value for the corresponding field:
    a. Title: **My Page**
12. Click **Create** in the upper right to create the page. A **Success** dialog box is displayed with a message that your page has been created.
13. Click **Done**. The new page appears as a child page of **en** Site..
14. Select **My Page** page and click **Edit (e)**. The page opens in Edit mode.

15. Click the **Toggle Side Panel** icon at the top of the page.
16. Click the **Components** icon in the left pane.
17. Add a **text** component to the page. Add text **Lorem Ipsum** to the component. Click **Done** and verify the text displays, as shown..



18. Go to http://localhost:4502/content/training/us/en.search.html?q=Lorem and verify the output, as shown:



Dive into AEM APIs

Introduction

Application Programming Interfaces (APIs) are at the highest layer of the Adobe Experience Manager (AEM) architecture stack, and are most closely related to the different 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 use AEM APIs to help build workflows with specific business needs.

Objectives

After completing this course, you will be able to:

- Create an AEM page programmatically
- Import multiple pages
- Describe AEM projects
- Create a project and project templates
- Customize the process step in workflow

Creating AEM pages and assets programmatically

The following are the best practices for data migration:

- 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 (DAM), 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 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 use this method to add metadata to modify the data.

Exercise 1: Create an AEM page programmatically

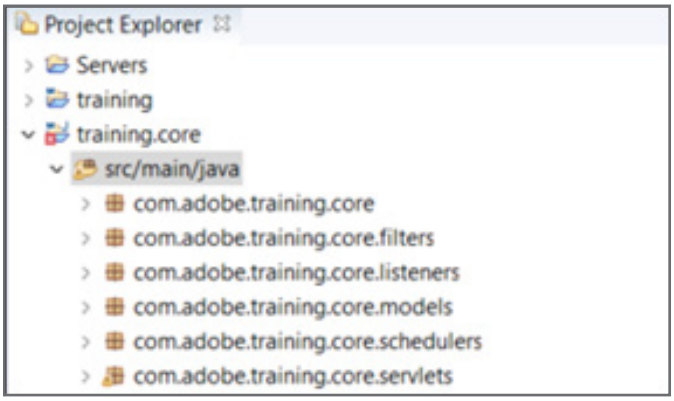
When migrating a website into AEM, it might make sense to create a process to create AEM pages to quickly create a site structure for content authors. This becomes particularly powerful when initial site structure is beyond manual page creation. In this exercise you will create a Java class that takes in a comma-delimited string that is parsed and a page is created based on the input.

This exercise includes two tasks:

1. Create a page creator
2. Deploy and test the page creator
3. (Optional Task): Extend beyond a single page

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** under **/core/src/main/java/com/adobe/training/core/servlets/.**

Note: The code is provided as part of Exercise_Files under **/core/src/main/java/com/adobe/training/core/servlets/**. 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.

6. The **PageCreator.java** file opens in the editor, as shown:

```
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.nio.charset.StandardCharsets;
9.  import java.security.AccessControlException;
10. import java.util.HashMap;
11.
12. import javax.servlet.Servlet;
13.
14. import org.apache.sling.api.SlingHttpServletRequest;
15. import org.apache.sling.api.SlingHttpServletResponse;
16. import org.apache.sling.api.servlets.SlingAllMethodsServlet;
17. import org.osgi.service.component.annotations.Component;
18.
19. import com.day.cq.tagging.InvalidTagFormatException;
20. import com.day.cq.tagging.Tag;
21. import com.day.cq.tagging.TagManager;
22. import com.day.cq.wcm.api.Page;
23. import com.day.cq.wcm.api.PageManager;
24. import com.day.cq.wcm.api.WCMException;
25. import com.fasterxml.jackson.databind.ObjectMapper;
26.
27. /**
28.  * This Servlet take an input parameter in the form of a string or CSV:
29.  *
30.  * New Page Path, Page Title, Template, Page Tag
31.  *
32.  * And outputs AEM page(s) created.
33.  *
34.  * To test, you must create a content node with a resourceType=training/tools/pagecreator
35.  *
36.  * /content/pagecreator {sling:resourceType=training/tools/pagecreator}
37.  *
38.  * Example cURL Command:
39.  * String Example:
40.  * $ curl -u admin:admin -X POST http://localhost:4502/content/pagecreator.json -F importer=»/content/training/
     us/en/community,Our Community,/conf/training/settings/wcm/templates/content-page-template,/content/cq:tags/
     training/community»
41.  *
42.  * CSV example:
43.  * $ curl -u admin:admin -X POST http://localhost:4502/content/pagecreator.json -F importer=@PageCreator.csv
44.  */
45.
46. @Component( service = Servlet.class,
47.         property = {
48.             "sling.servlet.resourceTypes=training/tools/pagecreator",
49.             "sling.servlet.methods=POST"
50.             }
51.         )
52. public class PageCreator extends SlingAllMethodsServlet {
```

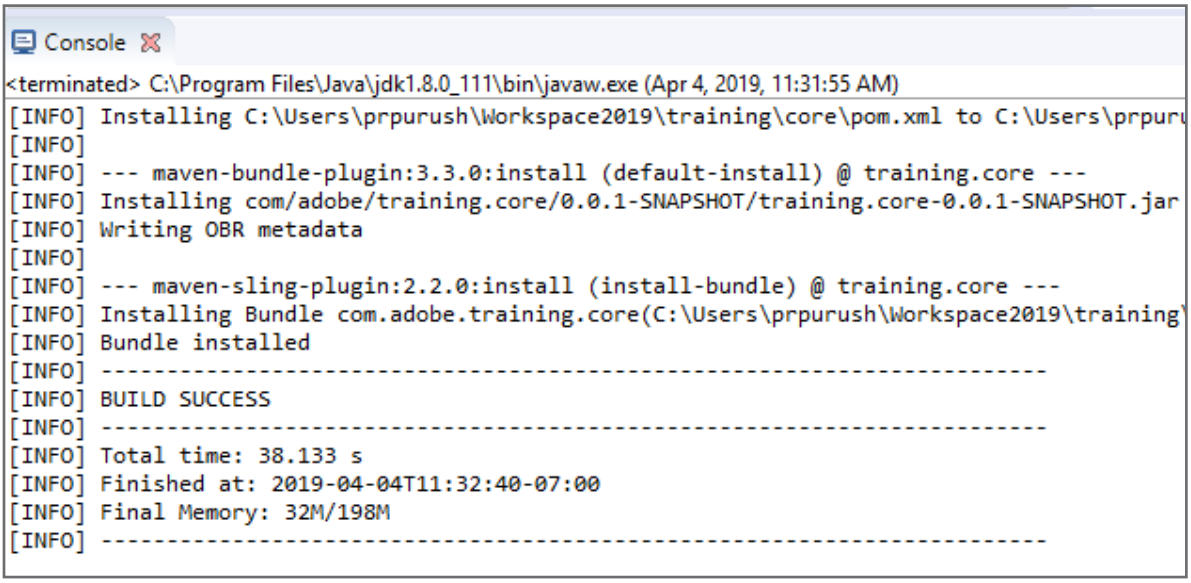
```
53.     private static final long serialVersionUID = 1L;
54.
55.     @Override
56.     public void doPost(SlingHttpServletRequest request, SlingHttpServletResponse response)throws IOException {
57.         response.setContentType("application/json");
58.         HashMap<String, HashMap<String, String>> resultObject = new HashMap<>();
59.
60.         String param = request.getParameter("importer");
61.         byte[] input = param.getBytes(StandardCharsets.UTF_8);
62.         InputStream stream = new ByteArrayInputStream(input);
63.         try {
64.             resultObject = readInput(request, stream);
65.         } catch (IOException e) {
66.             resultObject = null;
67.         }
68.
69.         ObjectMapper objMapper = new ObjectMapper();
70.         if (resultObject != null) {
71.             //Write the result to the page
72.             response.getWriter().print(objMapper.writerWithDefaultPrettyPrinter().writeValueAsString(resultObject));
73.         } else {
74.             HashMap<String, String> errorObject = new HashMap<String, String>() {{
75.                 put("Error","Could not read csv input»);
76.             }};
77.             response.getWriter().print(objMapper.writerWithDefaultPrettyPrinter().writeValueAsString(errorObject));
78.         }
79.         response.getWriter().close();
80.     }
81.
82.     /**
83.      * Reads the input. The input MUST be in the form of:
84.      * <p>
85.      * JCR path, Page Title, Page Template, AEM Tag
86.      *
87.      * @param request resourceResolver will be derived from the request
88.      * @param stream Stream from the input
89.      * @return JSON object that contains the results of the page creation process
90.      */
91.     private HashMap<String, HashMap<String, String>> readInput(SlingHttpServletReque
    st request, InputStream stream) throws IOException {
92.         HashMap<String, HashMap<String, String>> out = new HashMap<>();
93.
94.         String line;
95.         String[] newPage;
96.         HashMap<String, String> createdPageObject = null;
97.
98.         try (BufferedReader br = new BufferedReader(new InputStreamReader(stream))) {
99.             //Read each line of the CSV and if the input is a string, this will loop once
100.            while ((line = br.readLine()) != null) {
101.                newPage = line.split(",");
102.                String inputPath="", inputTitle="", inputTemplatePath="", inputTag="";
103.
104.                //set values from input
105.                if (newPage.length > 0)
106.                    inputPath = newPage[0]; //Add the creation path
107.                if (newPage.length > 1)
108.                    inputTitle = newPage[1]; //Add the Title of the new page
109.                if (newPage.length > 2)
110.                    inputTemplatePath = newPage[2]; //Add the desired template
111.                if (newPage.length > 3)
112.                    inputTag = newPage[3]; //Add requested tags
113.
114.                //Create a new page based on a line of input
115.                createdPageObject = createTrainingPage(request, inputPath, inputTitle, inputTemplatePath, inputTag);
116.
117.                //add the status of the row into the json array
118.                if(createdPageObject.get("Status").contentEquals("Successful")) {
119.                    out.put(inputPath, createdPageObject);
120.                } else {
121.                    out.put(line, createdPageObject);
122.                }
123.                createdPageObject = null;
124.            }
125.        }
126.        return out;
127.    }
128.
```

```
129.    /** Helper method to create the page based on available input
130.     *
131.     * @param request resourceResolver will be derived from the request
132.     * @param path JCR location of the page to be created
133.     * @param title Page Title
134.     * @param template AEM Template this page should be created from. The template must exist in the JCR already.
135.     * @param tagPath Tag must already be created in AEM. The tag will be in the form of a path. Ex /content/cq:tags/
136.     * @return HashMap
137.     */
138.    private HashMap<String, String> createTrainingPage(SlingHttpServletRequest request, String path, String title, String template, String tagPath) {
139.        HashMap<String, String> pageInfo = new HashMap<>();
140.        pageInfo.put("Status","Error");
141.
142.        if (path != null && !path.isEmpty()) {
143.            //Parse the path to get the pageNodeName and parentPath
144.            int lastSlash = path.lastIndexOf("/");
145.            String pageNodeName = path.substring(lastSlash + 1);
146.            String parentPath = path.substring(0, lastSlash);
147.
148.            //Set a default template if none is given
149.            boolean invalidTemplate = false;
150.            if (template == null || template.isEmpty()) { //if no template has been given, assign the default
151.                template = "/conf/training/settings/wcm/templates/content-page-template";
152.            }
153.            else if(request.getResourceResolver().getResource(template) == null) { //check to see if the template exists
154.                invalidTemplate = true;
155.                pageInfo.put("Template","The template " + template + " doesn not exist.");
156.            }
157.
158.            //Create page
159.            PageManager pageManager = request.getResourceResolver().adaptTo(PageManager.class);
160.
161.            Page p = null;
162.            //Verify parentPath exists, a node for this page exists, and the template is valid
163.            if (pageManager != null && !parentPath.isEmpty() && !pageNodeName.isEmpty() && !invalidTemplate) {
164.                if(title == null || title.isEmpty()) {
165.                    pageInfo.put("Warning","No Page title given, using path name: " + pageNodeName);
166.                    title = pageNodeName;
167.                }
168.                try {
169.                    p = pageManager.create(parentPath,
170.                            pageNodeName,
171.                            template,
172.                            title);
173.                } catch (WCMException e) {
174.                    pageInfo.put("Error","Page couldn't be created. Parent path probably doesn't exist.");
175.                }
176.
177.                //Check to see if the page was successfully created
178.                if(p != null) {
179.                    //Add a tag to the page
180.                    if (tagPath != null && !tagPath.isEmpty()) {
181.                        //Make sure tag namespaces are properly formed
182.                        if(tagPath.contains("/content/cq:tags") || tagPath.contains(":") || tagPath.contains("/etc/tags")) {
183.                            //TagManager can be retrieved via adaptTo
184.                            TagManager tm = request.getResourceResolver().adaptTo(TagManager.class);
185.                            Tag tag;
186.                            try {
187.                                tag = tm.resolve(tagPath); //check if tag already exists
188.                                if(tag == null) {
189.                                    pageInfo.put("Warning","Tag doesn't exist, creating new tag: " + tagPath);
190.                                    tag = tm.createTag(tagPath, null, null);
191.                                }
192.                                tm.setTags(p.getContentResource(), new Tag[] {tag}, true);
193.                            } catch (AccessControlException e) {
194.                                pageInfo.put("Warning","Could not access the tags.");
195.                            } catch (InvalidTagFormatException e) {
196.                                pageInfo.put("Warning","Invalid Tag.");
197.                            }
198.                        } else {
199.                            pageInfo.put("Warning", "Tag path malformed and not added: " + tagPath);
```

```
200.                    }
201.                }
202.
203.                pageInfo.put("Status", "Successful");
204.                pageInfo.put("Location", p.getPath());
205.                pageInfo.put("Title", p.getTitle());
206.                pageInfo.put("Template Used", p.getTemplate().getPath());
207.                String tags = "";
208.                for(Tag t : p.getTags() ) { tags = tags + t.getTitle() + " "; }
209.                pageInfo.put("Tagged with", tags);
210.            }
211.        }
212.    } else {
213.        pageInfo.put("Error", "Page path not provided");
214.    }
215.    return pageInfo;
216.  }
217. }
```

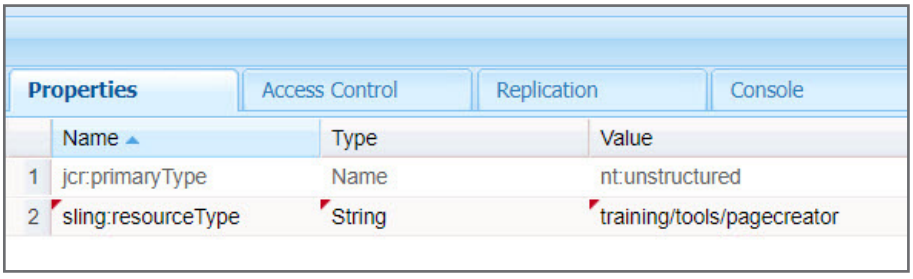
Task 2: Deploy and test the pagecreator

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:



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 pagecreator node is created.
6. Click **Save All.**
7. On the **pagecreator** node, add the following property, as shown:

- Name: **sling:resourceType**
- Type: **String**
- Value: **training/tools/pagecreator**

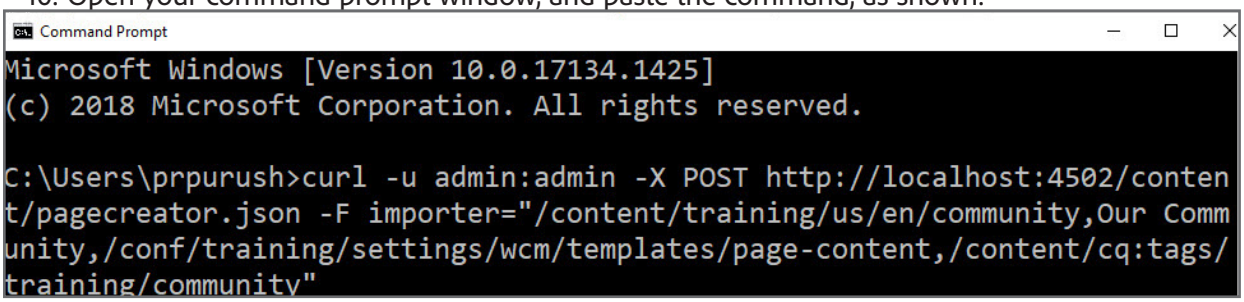


8. Click **Save All** to save the properties created on the **pagecreator** node.

Note: You have content that you can post and a servlet that will be triggered. You can now test with a cURL command.

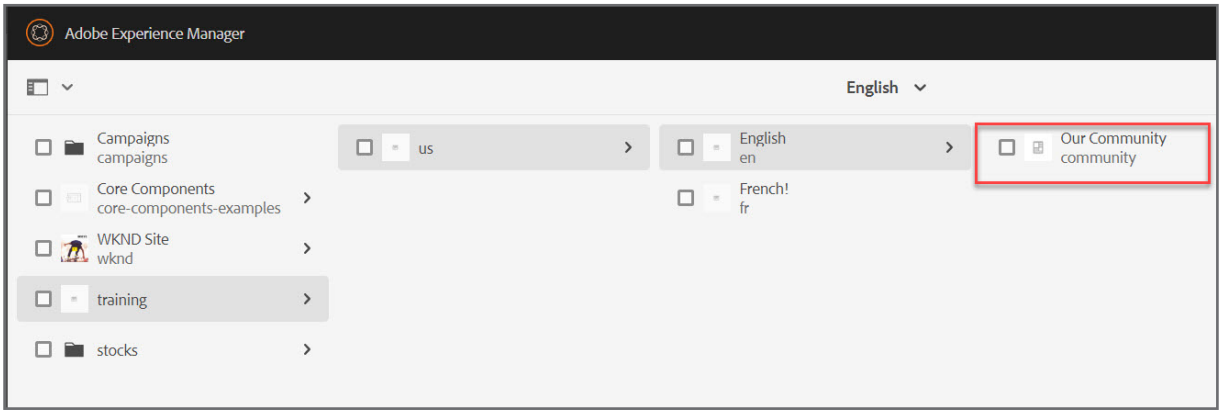
9. Copy the first command from **Exercise_Files** under **/training-files/Dive_into_AEM_APIs/pagecreator-curl-commands.txt.**

10. Open your command prompt window, and paste the command, as shown:



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** > **training** > **us** > **en**, as shown:



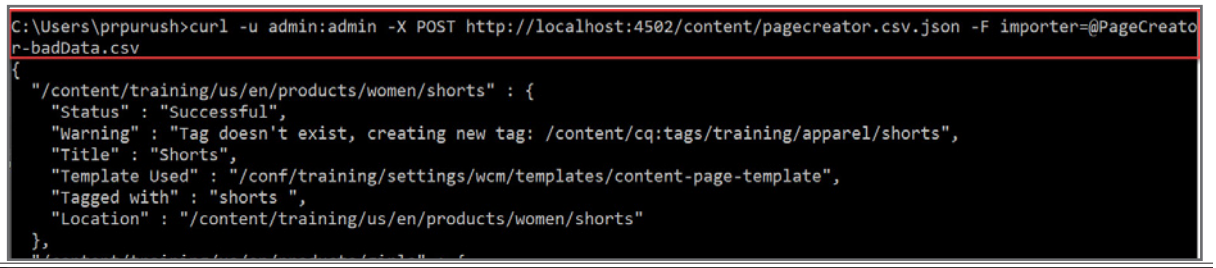
Note: If you want to use the browser rather than cURL, you can install the **http-pagecreator.zip** content package (from the exercise folder) and do a request for http://localhost:4502/content/pagecreator.html.

Task 3: (Optional Task): Extend beyond a single page

As you might have noticed, this java class also supports csv files. In this task, you will use a set of bad csv data for the importer and observe the output and then use good csv data to see it successfully import.

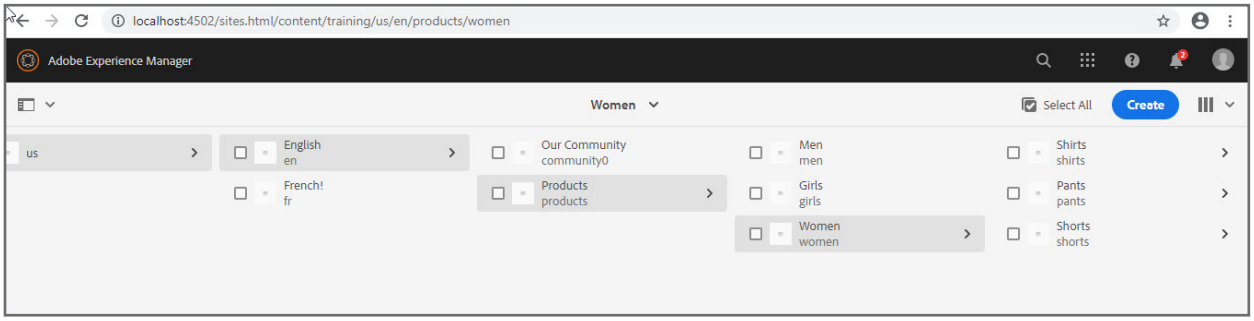
1. Try the servlet with bad data first. Execute the following statement, as shown:

```
curl -u admin:admin -X POST http://localhost:4502/content/
pagecreator.csv.json -F importer=@PageCreator-badData.csv
```



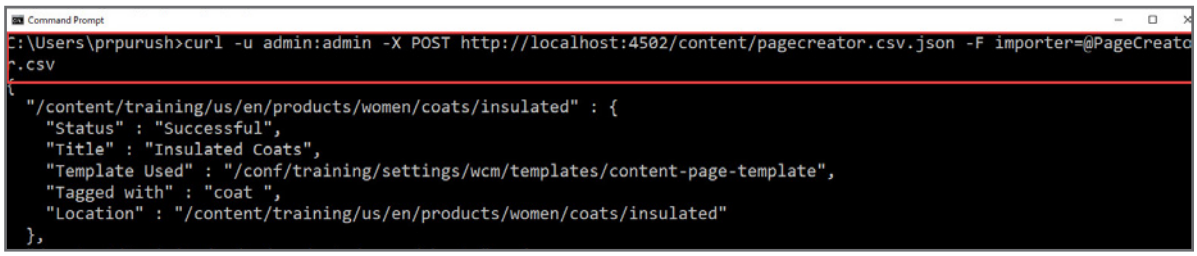
Note: You can also copy the command from **Exercise_Files** under **/training-files/Dive_into_AEM_APIs/pagecreator-curl-commands.txt.**

2. In AEM, navigate to **Sites** > **training** > **us** > **en**. Compare the set of child pages that were created to the pages defined in the csv file. Notice that not all the pages were created.

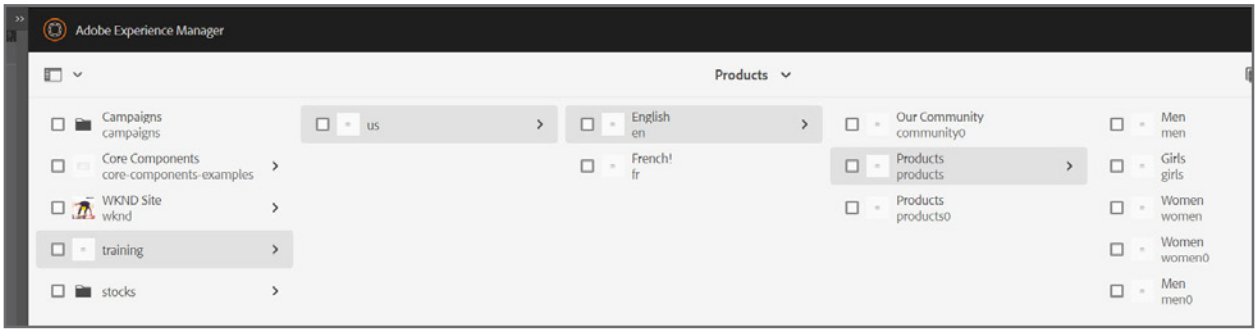


3. Try the servlet with well-formatted data by executing the following statement, as shown:

```
curl -u admin:admin -X POST http://localhost:4502/content/
pagecreator.csv.json -F importer=@PageCreator.csv
```



4. Verify the newly created pages in AEM: **Sites** > **training** > **us** > **en** > **Products**, as shown:



Note: Notice duplicate pages are created. For each page that was created in Step 1, a duplicate page is created in Step 3.

Note: If you would like to use an AEM component to support your page automation process, you can install training-files/Dive-into-AEM-APIs/http-pagecreator.zip in **Package Manager**. You then can access a single page input using http://localhost:4502/content/pagecreator.html. If you would like to upload a csv file, you can simply change the request to http://localhost:4502/content/pagecreator.csv.html.

AEM Projects

The AEM Projects feature helps manage and group all workflows and tasks associated with creating content in AEM Sites and Assets. This exercise includes two tasks:

Project Template

AEM Projects comes with several out-of-the box (OOTB) project templates. When creating a new project, authors can choose from these 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**.

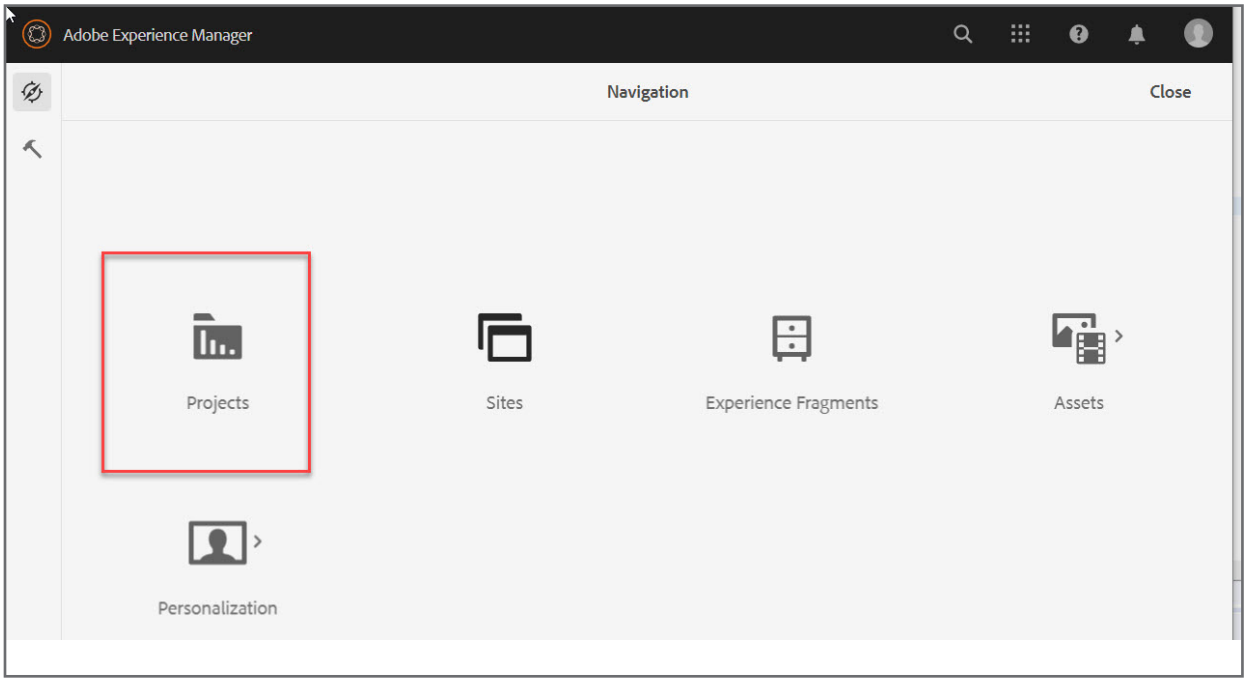
- Gadgets: The child nodes of the gadgets node control which project tiles populate the project's dashboard when you create a new project. The project tiles are simple cards that populate the workplace of a project.
- Roles: There are three default roles for every project: Observers, Editors, and Owners. 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.

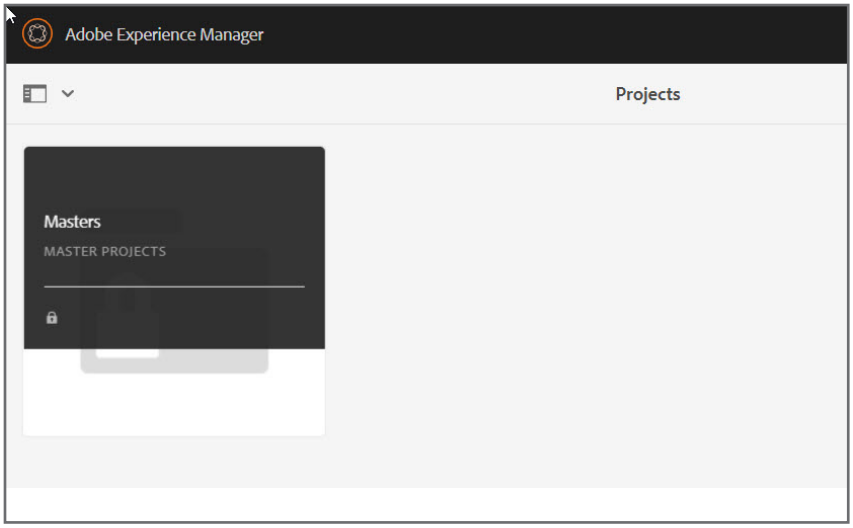
Exercise 2: Create a simple project

In this exercise, you will create a simple project.

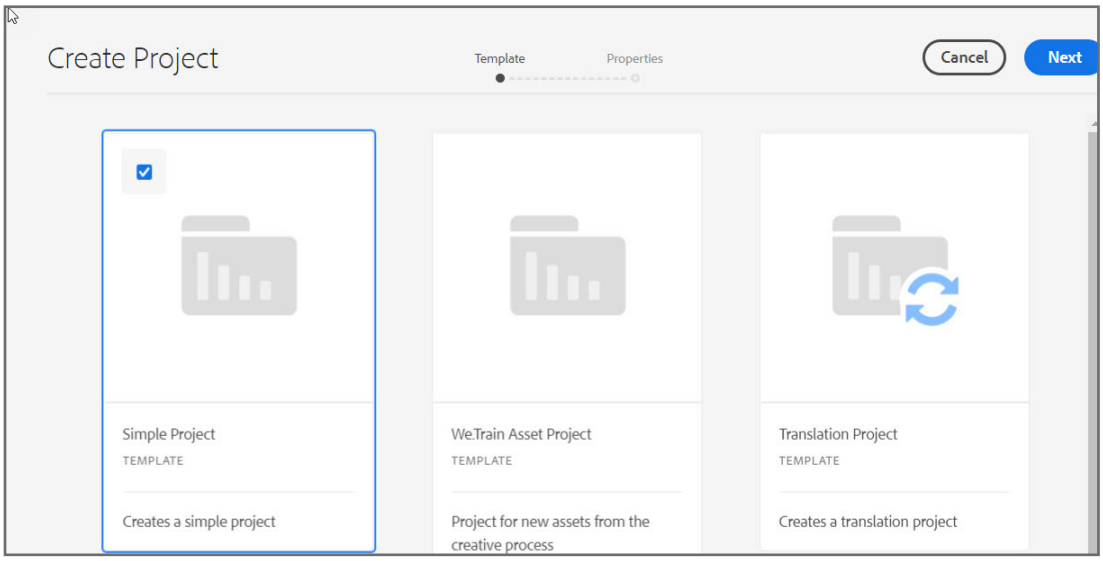
1. In AEM, go to the **Navigation** screen and click **Projects**, as shown:



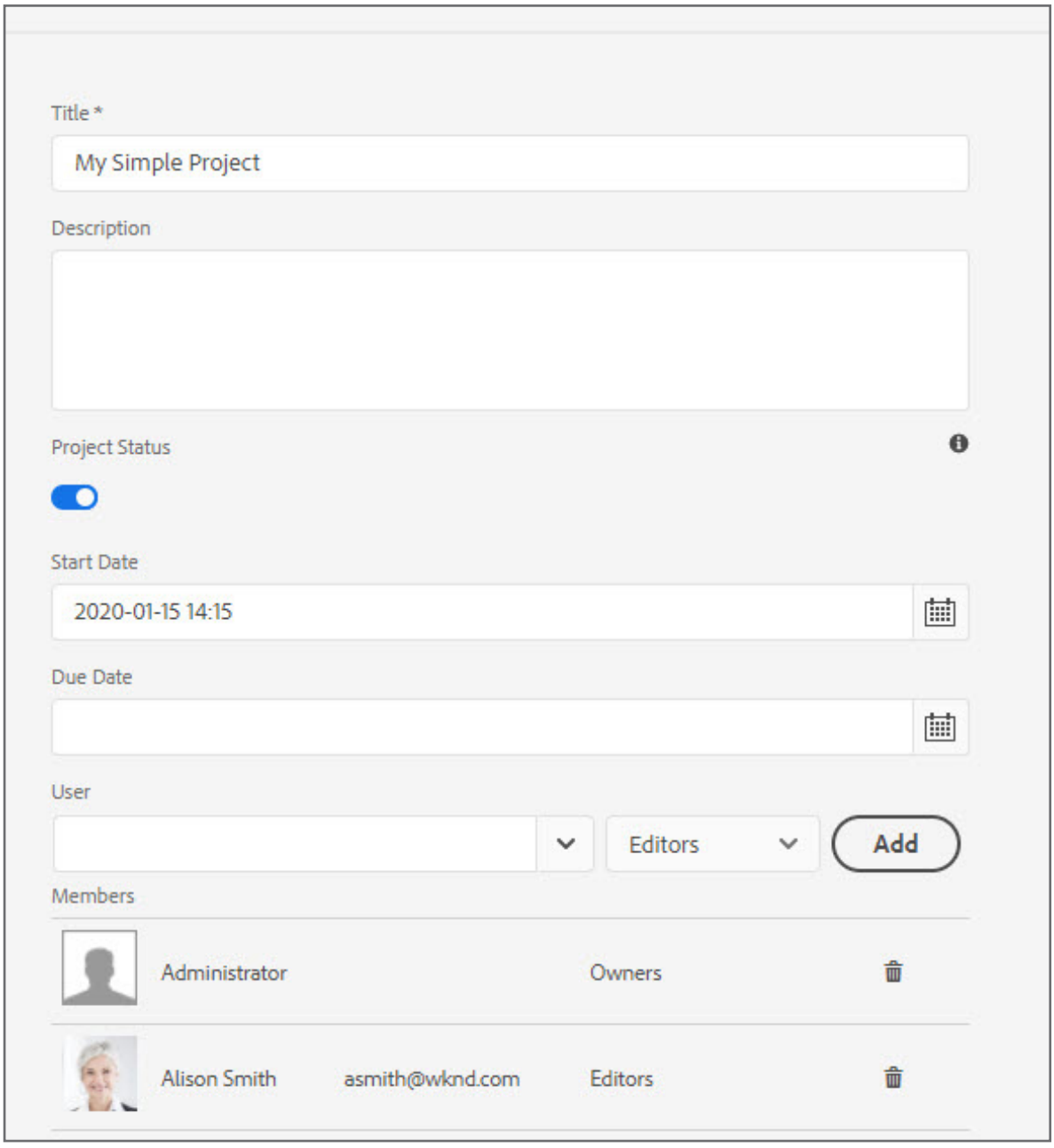
The **Projects** console opens, as shown:



2. In the Projects console, click **Create** > **Project**. The **Create Project** page opens.
3. Select **Simple Project** from the list of templates, as shown:

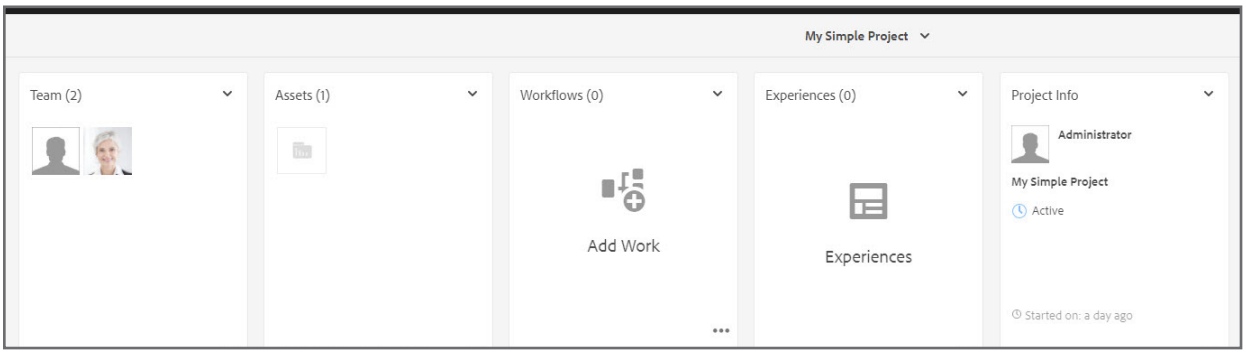


4. Click **Next**. The Properties window opens.
5. Type the following values for the simple project, as shown:
   a. Enter **My Simple Project** in the Title field.
   b. Type **Alison Smith** in the User field. Ensure the drop-down menu beside the **User** field is set to **Editors**, and click **Add**.



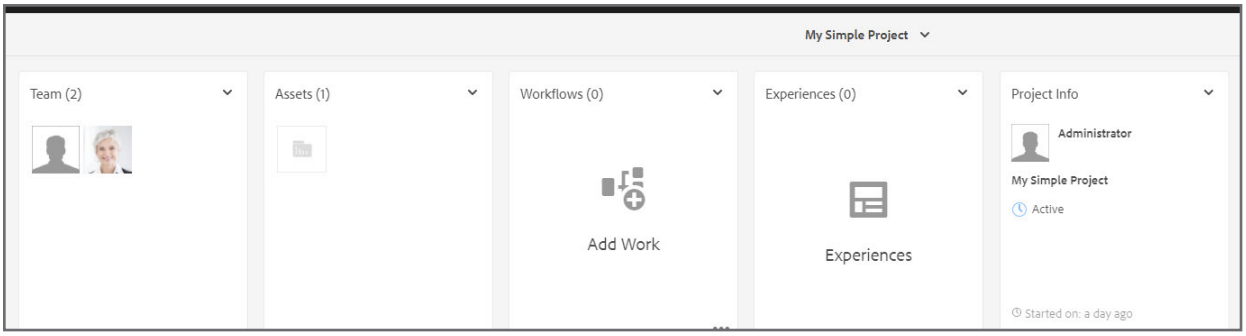
6. Click **Create** in the upper right. The Success pop-up window opens.

7. Click **Open**. The project is created, as shown:



8. Notice the 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.



Executing an Existing Workflow

AEM comes with a set of predefined workflows that perform common actions. You can customize these 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)

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.

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 must complete the corresponding task(s).

You can notify participants of their required action(s) 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 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 the item at a given path 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

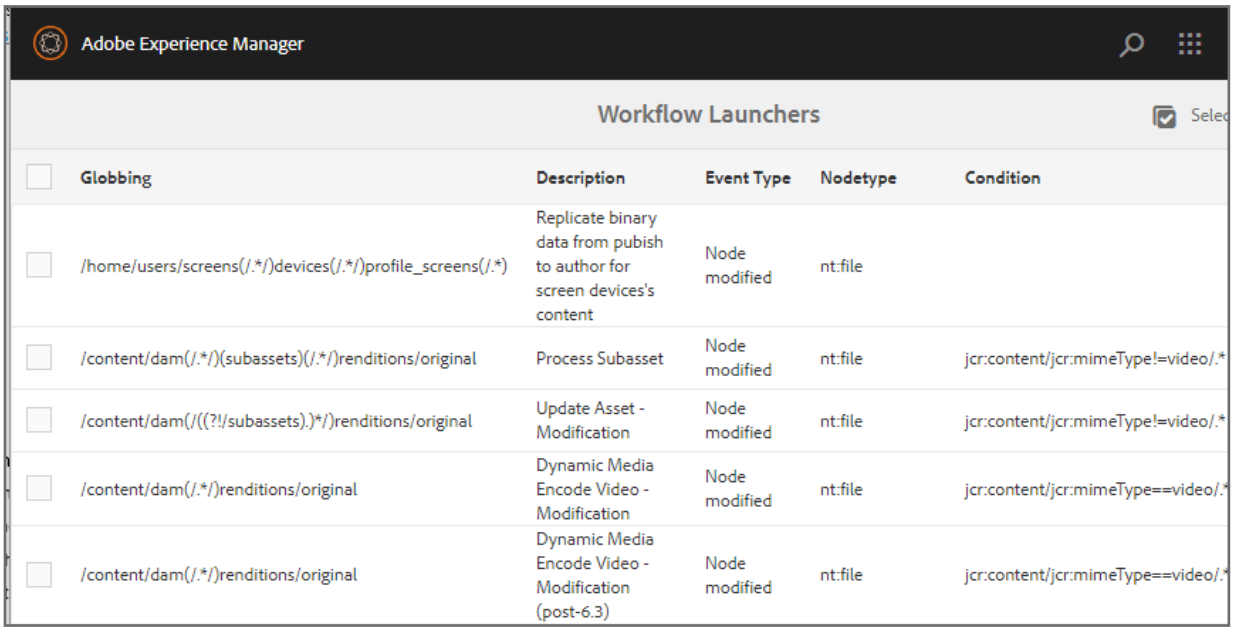
To create a workflow:

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:



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

Exercise 3: Add Java to a Custom Project

In this exercise you will create a custom workflow process. You can use the custom workflow process in standard workflows and project workflows. In this use case, we will install a custom project and project workflow and then add out custom workflow process to the project workflow.

This exercise includes four tasks:

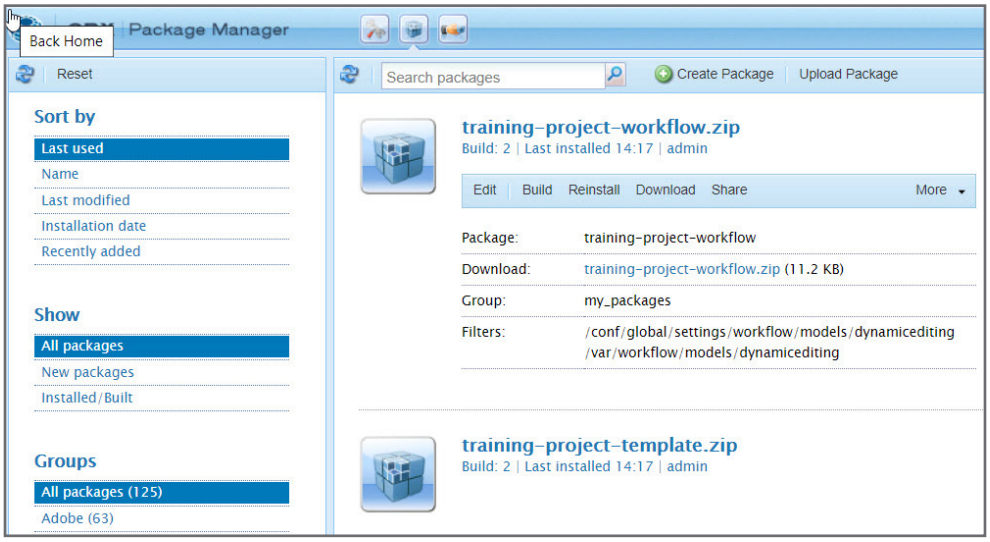
1. Install a custom AEM project
2. Create the custom workflow process
3. Add the custom workflow process to a project workflow
4. Test the workflow

Task 1: Install a custom AEM project

1. Log in to AEM and navigate to http://localhost:4502/crx/packmgr/index.jsp.
2. Install the **training-project-template.zip** content package from **Exercise_Files** under **\training-files\Dive into AEM APIs\ training-project-template.zip** with the completed project template.

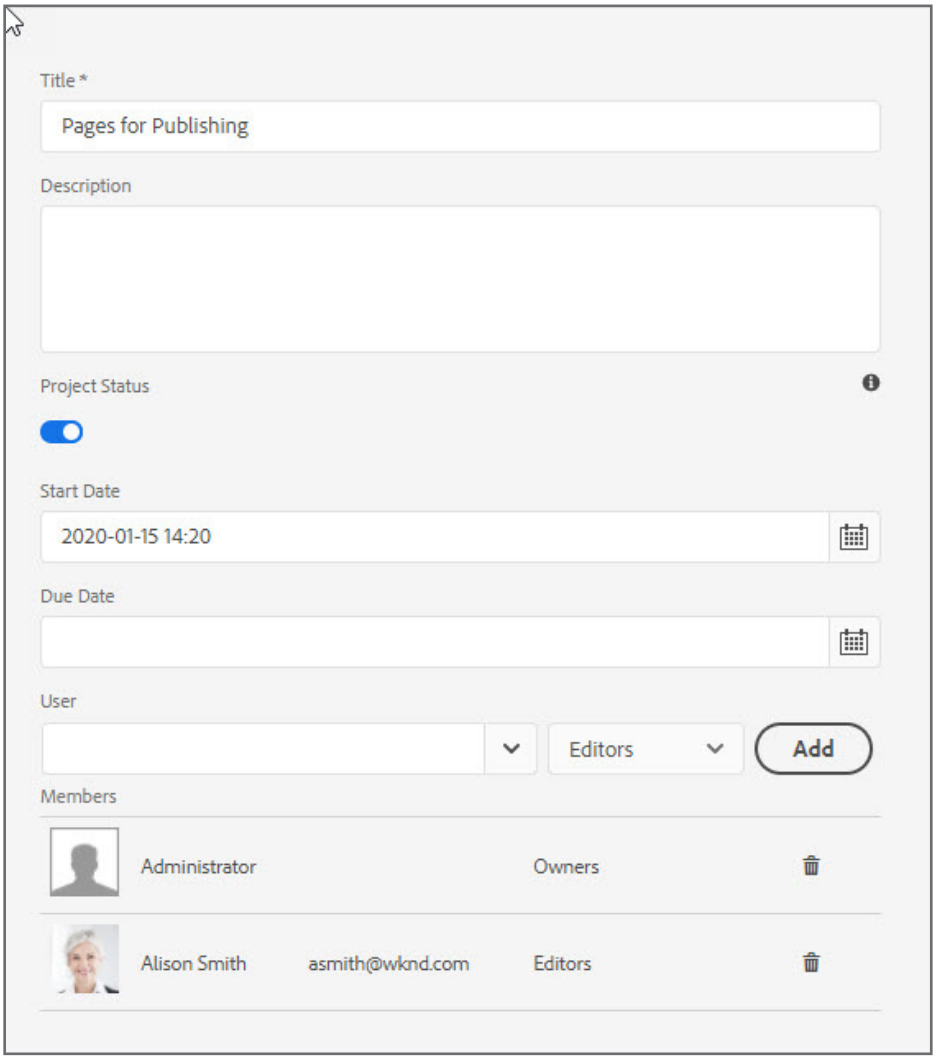
3. Install the **training-project-workflow.zip** content package from **Exercise_Files** under **\training-files\Dive into AEM APIs\ training-project-workflow.zip**.

4. Verify the content packages are successfully installed:

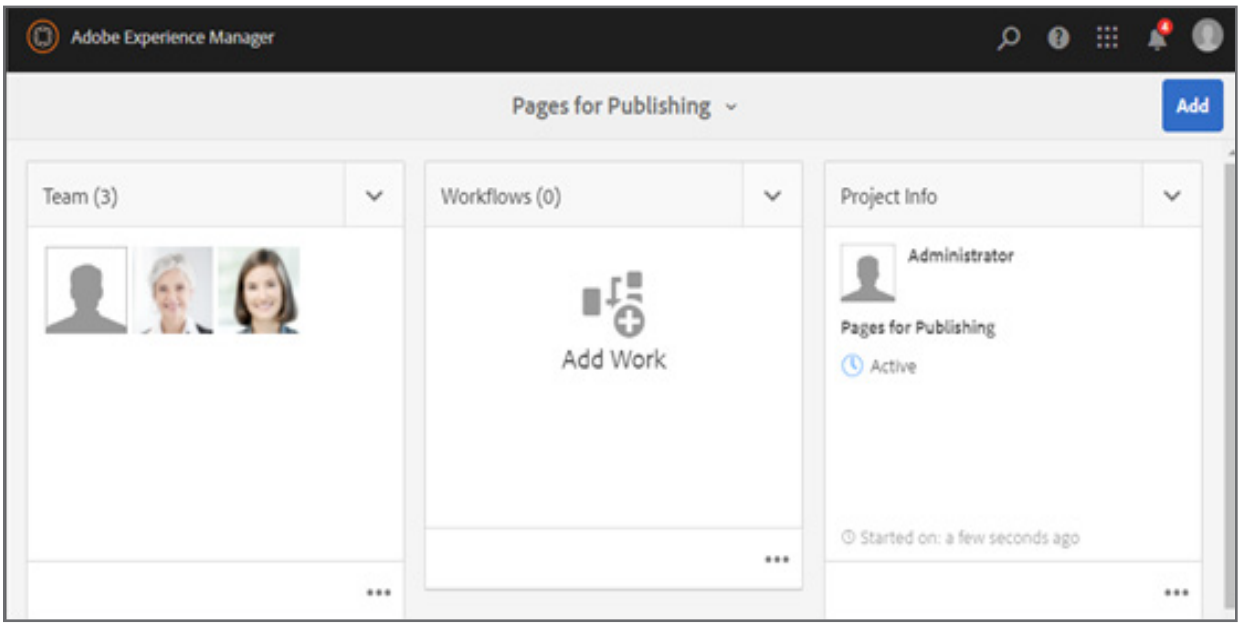


NOTE: If you want to learn more on how to create a project template, and execute and build a workflow from scratch, please refer to the "Appendix-Creating Project and Workflow" of this course.

5. To test the new project, navigate to the **AEM** > **Projects**. Select **Create** > **Project**. The Create Project wizard opens on the Template page.
6. Click **Training Publishing Project** and click **Next**. The **Create Project** window opens.
7. Provide these values for the project, as shown:
   - Type **Pages for Publishing** in the **Title** field.
   - Type **Alison Smith** in the **User** field. Ensure the drop-down menu beside the **User** field is set to **Editors**, and click **Add**.



8. Click **Create** in the upper right. The **Success** pop-up window opens.
9. Click **Open** in the pop-up window. Observe the changes of the custom project, as shown.



Task 2: Create the custom workflow process

1. Launch **Eclipse** and in **Project Explorer**, navigate to **training.core** > **src/main/java**.
2. Copy the file **ApprovalStatusWriter.java** from **Exercise_Files** under **/core/src/main/java/com/adobe/training/core/**.

NOTE: The code is provided as part of the Exercise_Files under **/core/src/main/java/com/adobe/training/core/**. 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.

3. In Eclipse, right-click **com.adobe.training.core** and paste the file. The **ApprovalStatusWriter.java** class is created.
4. The **ApprovalStatusWriter.java** file opens in the editor.

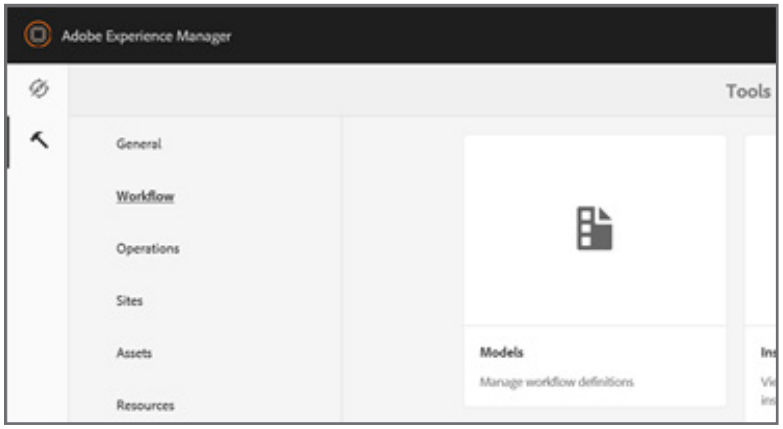
```
1.  package com.adobe.training.core;
2.
3.  import com.adobe.granite.workflow.WorkflowSession;
4.  import com.adobe.granite.workflow.WorkflowException;
5.  import com.adobe.granite.workflow.exec.WorkItem;
6.  import com.adobe.granite.workflow.exec.WorkflowProcess;
7.  import com.adobe.granite.workflow.exec.WorkflowData;
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. /**
18.  * This workflow process writes a property to the payload.
19.  * Where the property is payload-node/jcr:content/approved = true | false
20.  *
21.  */
22. @Component(service = WorkflowProcess.class,
23.         property = {Constants.SERVICE_DESCRIPTION + "=Workflow process to set status to approved",
24.                 Constants.SERVICE_VENDOR + "=Adobe",
25.                 "process.label=Approval Status Writer"
26.         })
27. public class ApprovalStatusWriter implements WorkflowProcess {
28.
29.     private static final String TYPE_JCR_PATH = "JCR_PATH";
30.
31.     @Override
32. public
    void execute(WorkItem item, WorkflowSession workflowSession, MetaDataMap args) throws WorkflowException {
33.         WorkflowData workflowData = item.getWorkflowData();
34.         if (workflowData.getPayloadType().equals(TYPE_JCR_PATH)) {
35.             String path = workflowData.getPayload().toString() + "/jcr:content";
36.             try (ResourceResolver rr = workflowSession.adaptTo(ResourceResolver.class)){
37.                 Resource resource = rr.getResource(path);
38.                 resource.adaptTo(ModifiableValueMap.class).put("approved", readArgument(args));
39.                 rr.commit();
40.             }
41.             catch (PersistenceException e ) {
42.                 throw new WorkflowException(e.getMessage(), e);
43.             }
44.         }
45.     }
46.
47.     private static boolean readArgument(MetaDataMap args) {
48.         //setting default to true if no args exist as this step won't
49.         //be reached unless Approval has been granted.
50.         String argument = args.get("PROCESS_ARGS", "true");
51.         return argument.equalsIgnoreCase("true");
52.     }
53. }
```

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

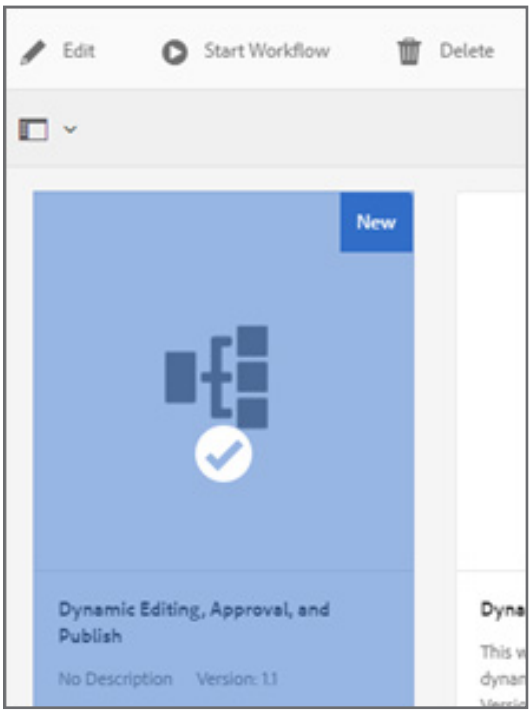
6. Verify the bundle installed successfully, as shown:

Task 3: Add the custom workflow process to a project workflow

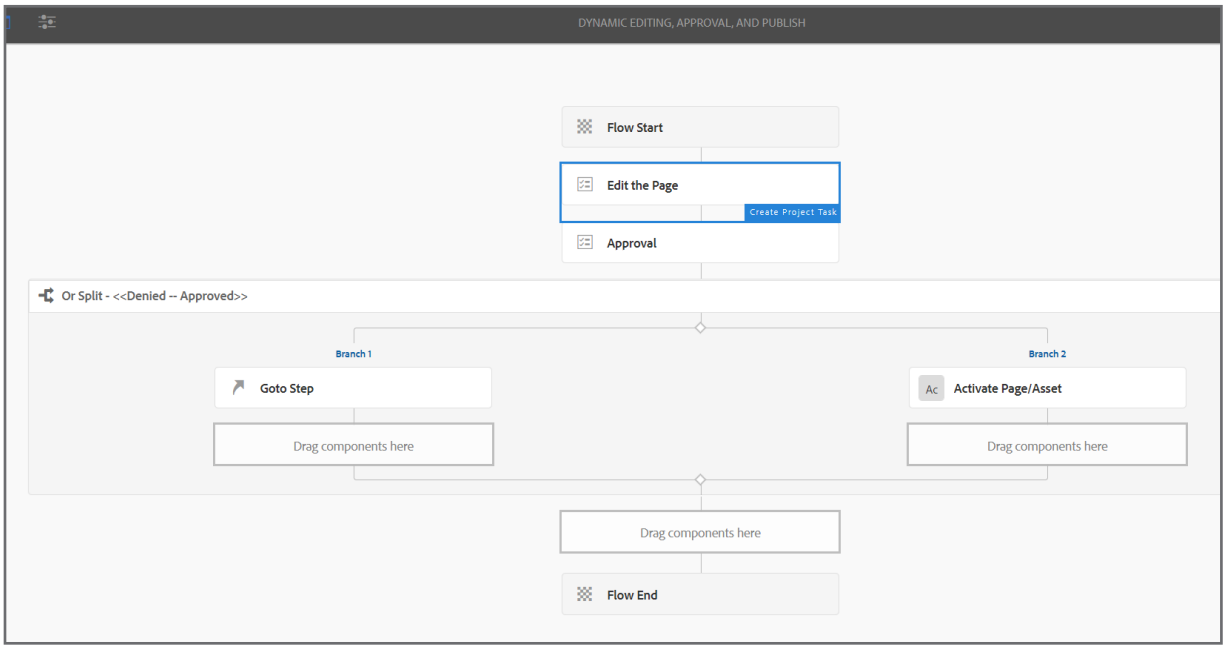
1. In AEM, navigate to **Tools** > **Workflow** > **Models**, as shown:



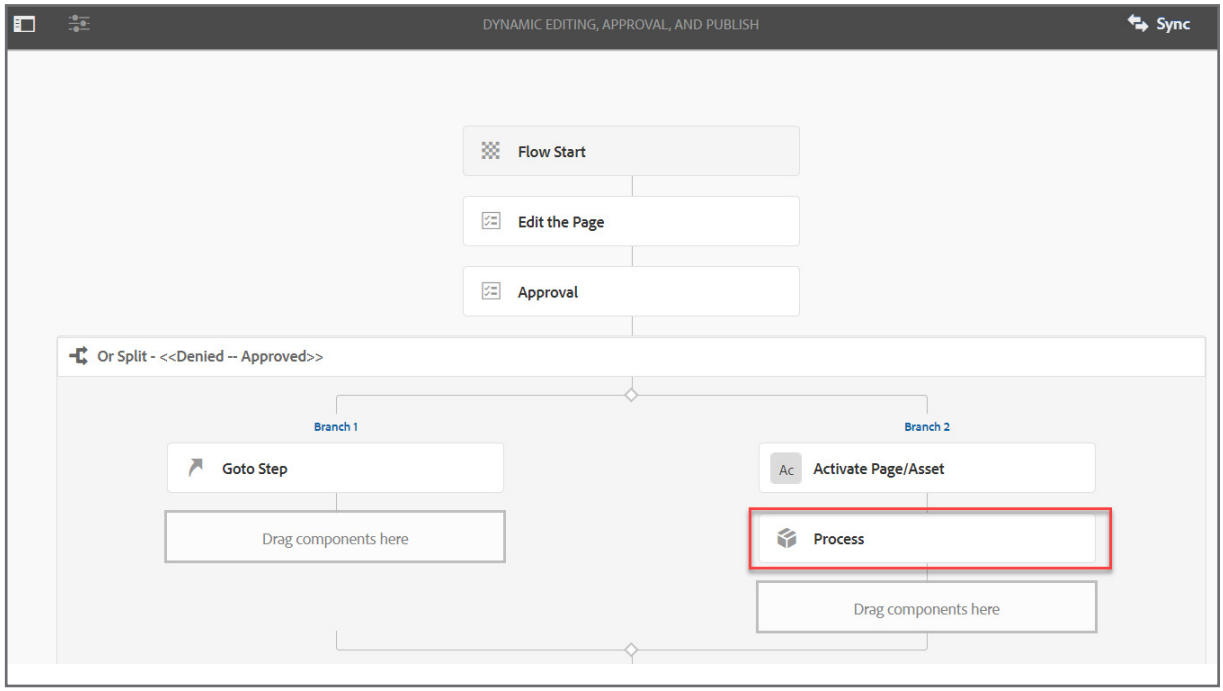
2. In the **Workflow Models** window, select **Dynamic Editing, Approval, and Publish** and click **Edit**, as shown:



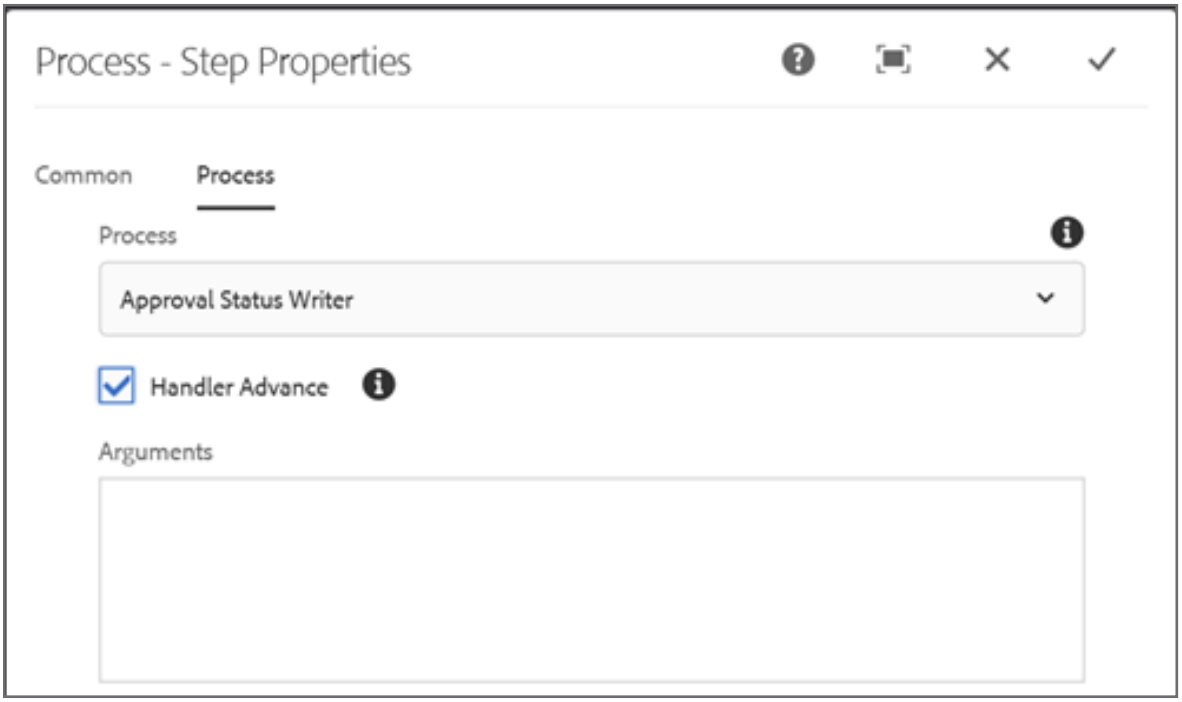
The **DYNAMIC EDITING, APPROVAL, and PUBLISH** Workflow Model opens for editing in a new tab in your browser, as shown:



3. Drag the **Process step** component from the left-hand side onto the **Drag components here** box, as shown:



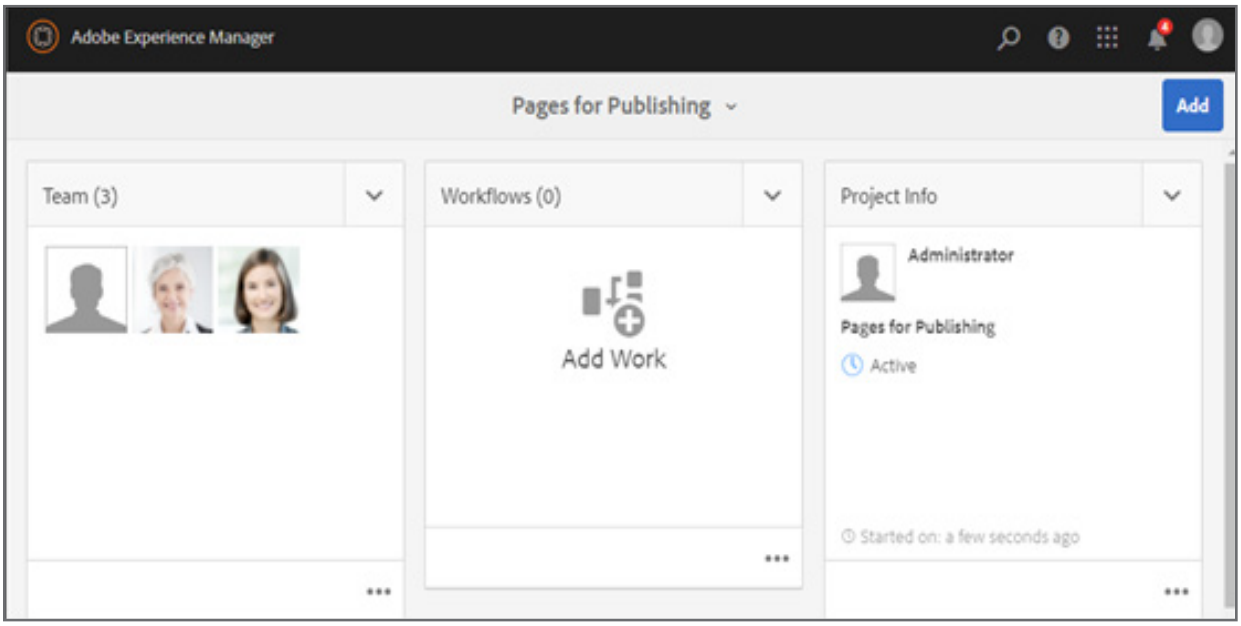
4. Click the **Process** step and click the **Configure** icon (wrench) to open the **Process** dialog box. By default, the dialog box opens on the **Common** tab.
5. In the **Title** field, type: **Approval Status Writer**
6. Click the **Process** tab and provide the following details, as shown:
   - › From the Process drop-down menu, select **Approval Status Writer**.
   - › Select the **Handler Advance** checkbox.



7. Click the Done icon to save the changes.
8. At this point the workflow is complete. In the upper right click **Sync**, as shown. This will write the workflow model.

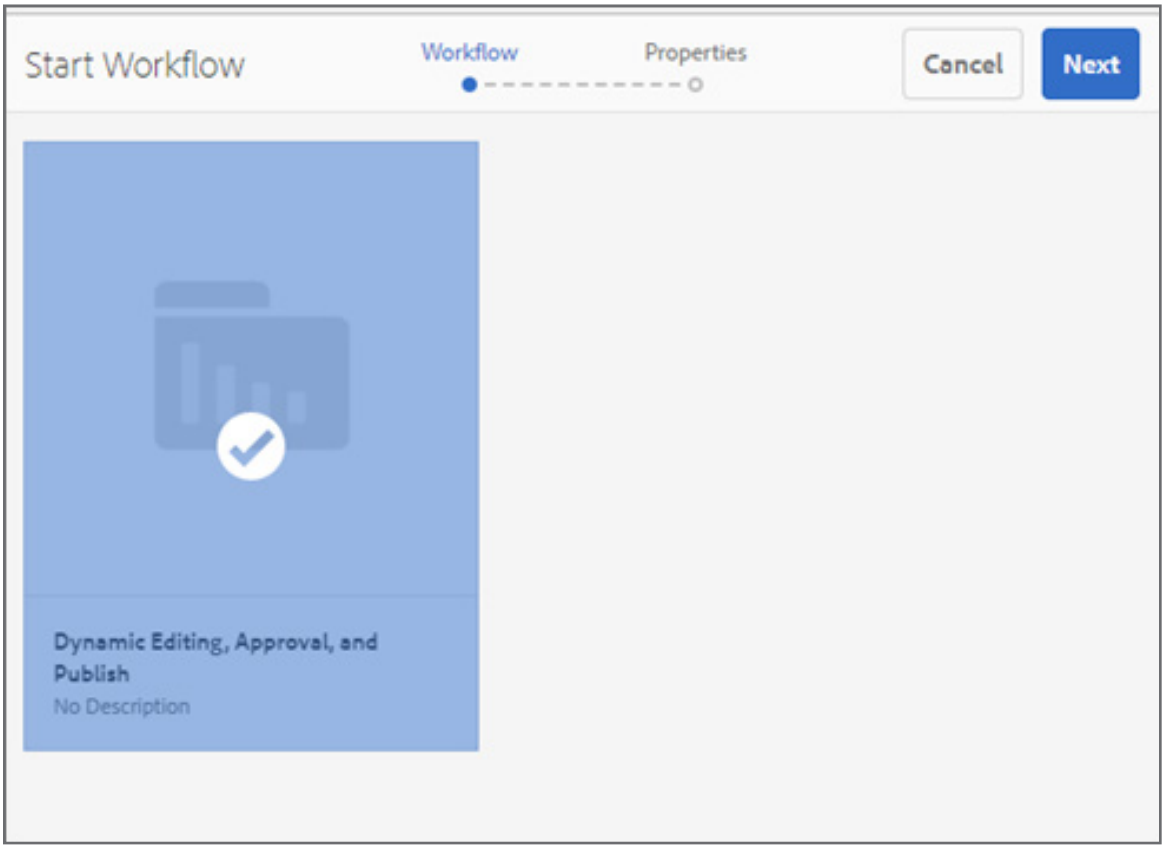
Task 4: Test the Workflow

1. Open up the custom AEM project that you created in Task 1 (http://localhost:4502/projects.html/content/project and then click **Pages for Publishing**). The page displays, as shown:



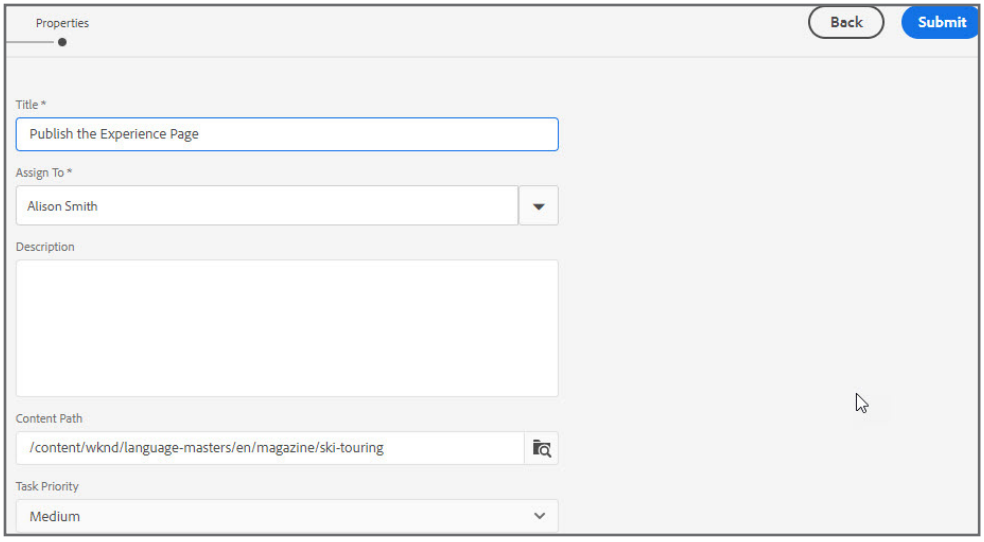
2. Click the drop-down arrow on the **Workflows** tile and select **Start Workflow**. The **Start Workflow** page opens, as shown:

3. Select the **Dynamic Editing, Approval and Publish** workflow and click **Next**, as shown: The Properties window opens.



4. Provide the following details, as shown:

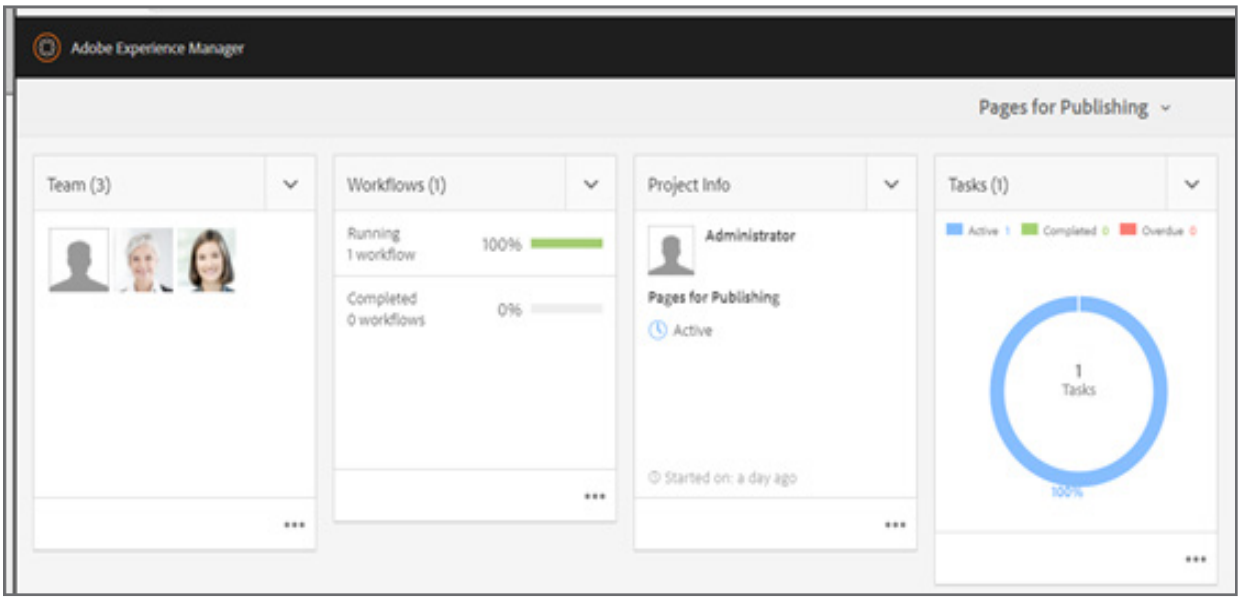
- Title: **Publish the Experience Page**
- Assign To: **Alison Smith**
- Content Path: **/content/wknd/language-masters/en/magazine/ski-touring**



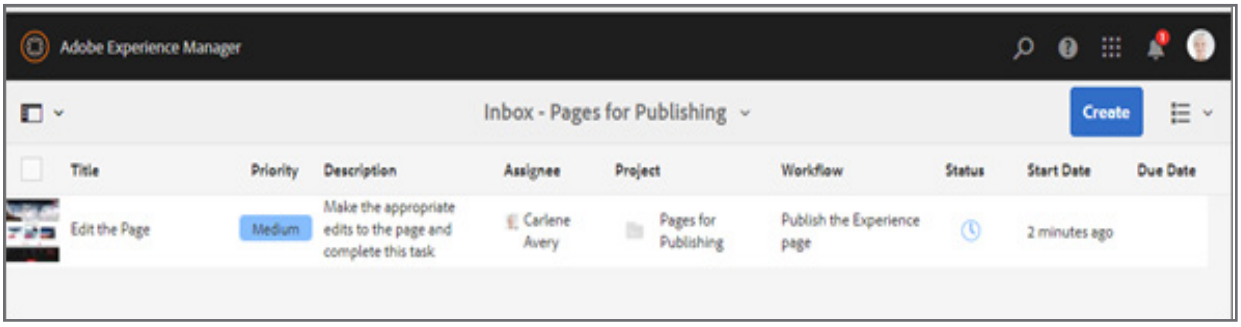
5. Leave all other fields as is.
6. Click **Submit** in the upper right. The workflow is built.

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.

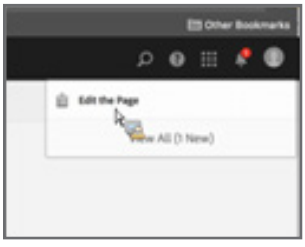
7. Refresh the browser and notice the new Tasks tile, as shown:



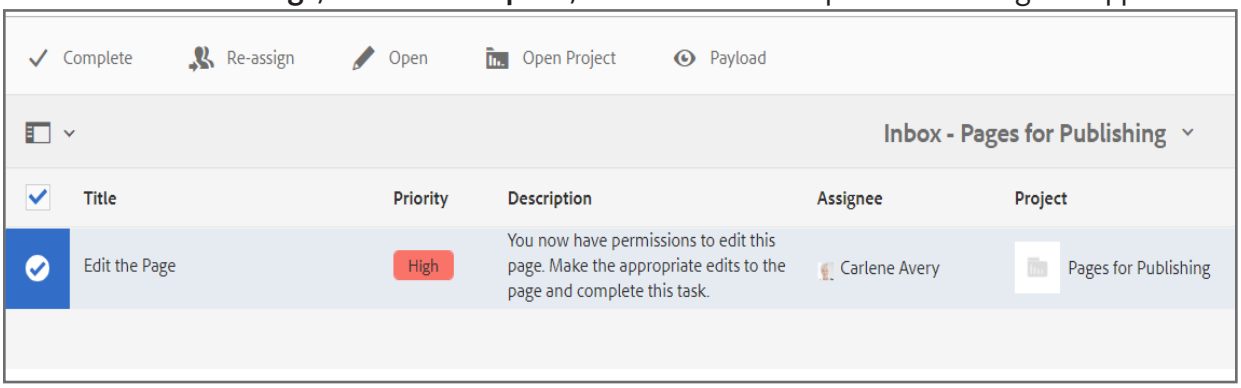
8. Click the **ellipsis** (three dots) in the lower right on the **Tasks** Tile. The **Inbox - Pages for Publishing** opens, as shown:



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.

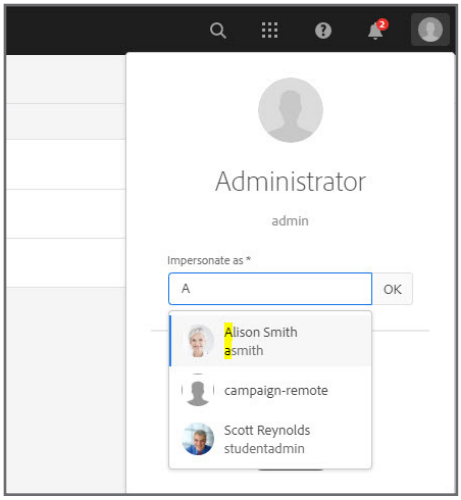


11. Type a comment and click **Complete**.

12. Notice there is now an **Approval Task**, as shown:

13. In the upper right, click the circular **User** logo.

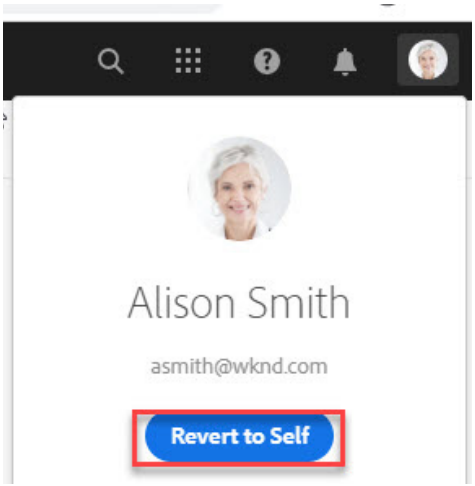
14. In the Impersonate as field, type **Alison Smith**. When her name appears in the drop-down list, click to select it, and then click **OK**, as shown:



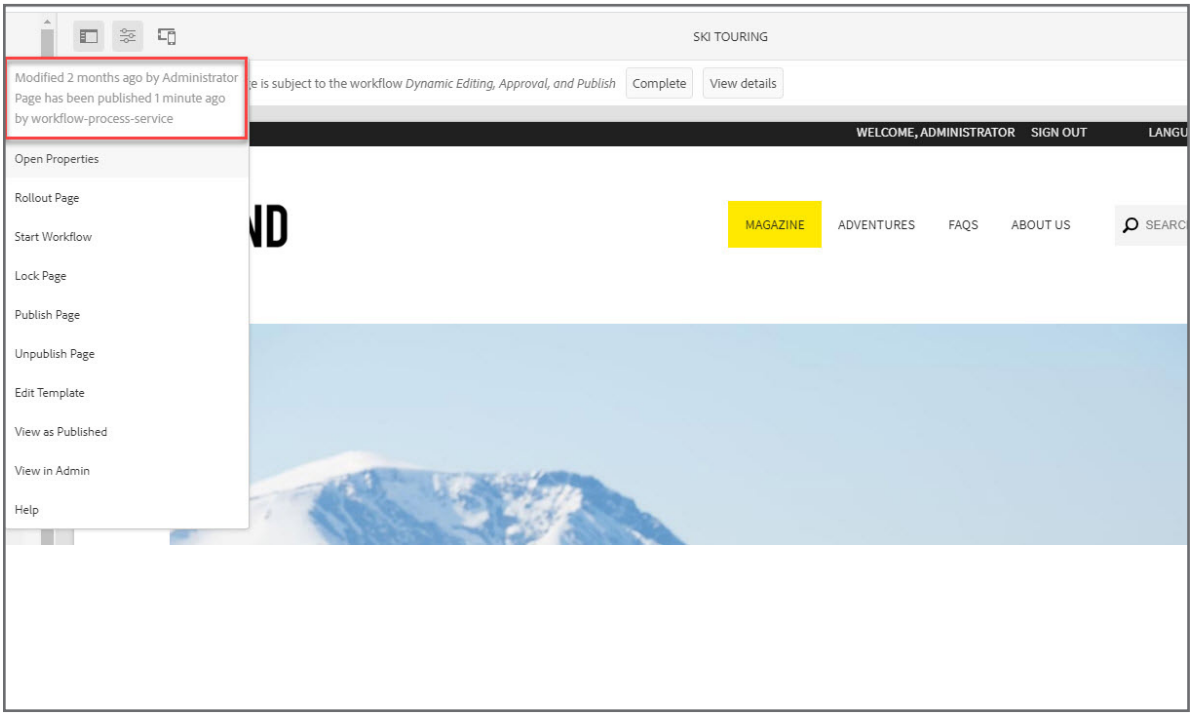
15. Select the **Approval Task** and click **Complete**. The **Complete Task** pop-up window appears. Notice there is a Select action drop-down menu with **Approved** and **Denied** as options.

Note: If you select Denied, the workflow will step back and create a new task and re-edit the page with the comments of the reviewer. If you select Approved, the workflow will move forward.

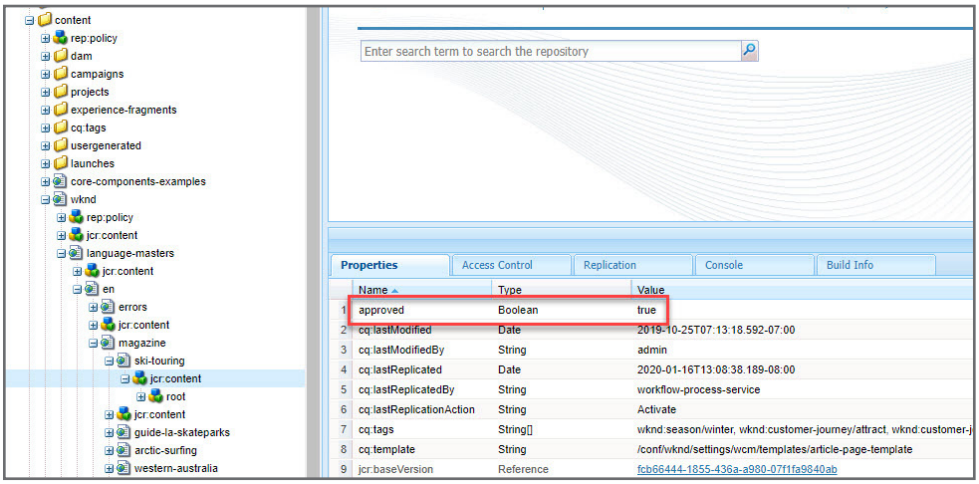
16. Complete the task by selecting **Approved** and adding a comment, **Approved**. Click **Complete**.
17. Click the User profile logo in the upper right and click **Revert to Self** (in your case, your "self" is the **admin** user,) as shown:



18. Navigate to http://localhost:4502/editor.html/content/wknd/language-masters/en/magazine/ski-touring.html and verify the page is published, as shown:



19. In **CRXDE Lite**, navigate to **/content/wknd/language-masters/en/magazine/ski-touring/jcr:content** and observe the jcr-content properties, as shown:



Note: Notice a new property named **approved** is added to the list of properties.

References

For more information on AEM Projects, refer:
https://docs.adobe.com/content/help/en/experience-manager-cloud-service/sites/authoring/projects/overview.html
Workflows:
https://docs.adobe.com/help/en/experience-manager-cloud-service/sites/authoring/workflows/overview.html

Users, Groups, and Permissions

Introduction

Every Adobe Experience Manager (AEM) implementation requires the configuration of users, groups, and permissions. You can govern access to content and functionality in AEM by configuring permissions. While an admin can perform most configurations of users, groups, and permissions in the AEM UI, some configurations may involve other tools and consoles. This module presents best practices and options for these configurations.

Objectives

After completing this module, 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
- Import permissions into the maven project

Working with Users, Groups, and Access Control Lists

You can configure and manage user authentication and authorization within AEM.

Users and Groups

Organizations generally define multiple roles, with varying skill sets, for the purpose of granting access to AEM resources and functionality. For these purposes, AEM provides multiple tools:

- Configuration Browser – managing access to Template Editor, Content Fragment Model Editor, and ContextHub Segment Editor
- Permissions tab in the Properties pane – managing access to pages
- Permissions Console – detailed management of Access Control Lists.

Permissions and ACLs

AEM uses Access Control Lists (ACLs) to determine what actions a user or a group can take, and where those actions can be performed. Permissions define who can perform which actions on a resource.

Access Control Lists (ACLs)

AEM uses Object-based Access Control. The ACLs are associated with the resource, not the user or group.

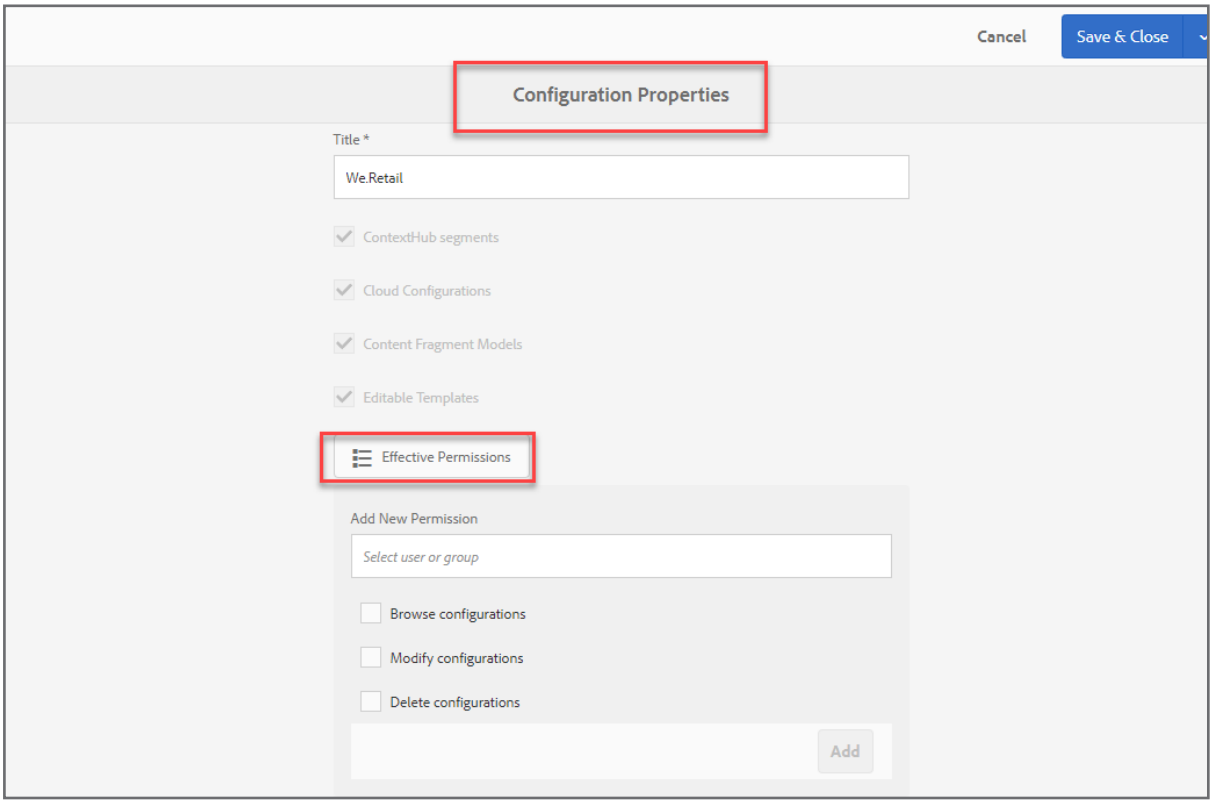
Access Control Lists (ACLs) are made up of individual permissions: Access Control Entries (ACE), and are used to determine the order in which those permissions are actually applied. A list of effective permissions is formed by assembling all of the ACEs in all of the ACLs in the parent hierarchy. This list is then scanned bottom-up until the first appropriate permission to apply to the specified page, asset, or other resource is found. and that permission is then applied.

Concurrent Permissions in 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. An administrator can modify which permission is applied by changing the order of the conflicting permissions.

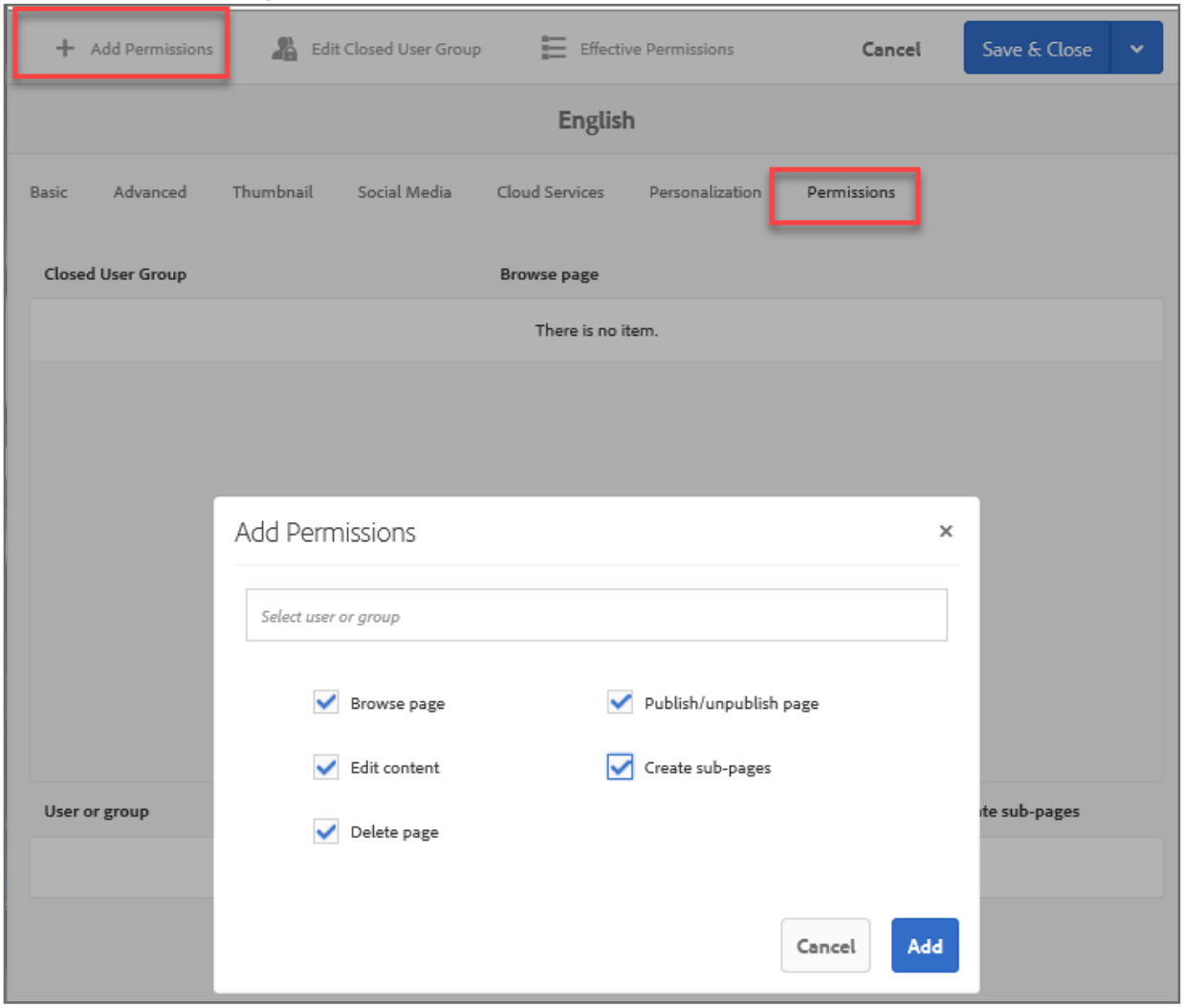
Managing Access to the Template Editor, Content Fragment Model Editor, and ContextHub Segment Editor

The Configuration Browser, found at **Tools** > **General** > **Configuration Browser** allows application administrators to manage access to "power/super user" functionality.



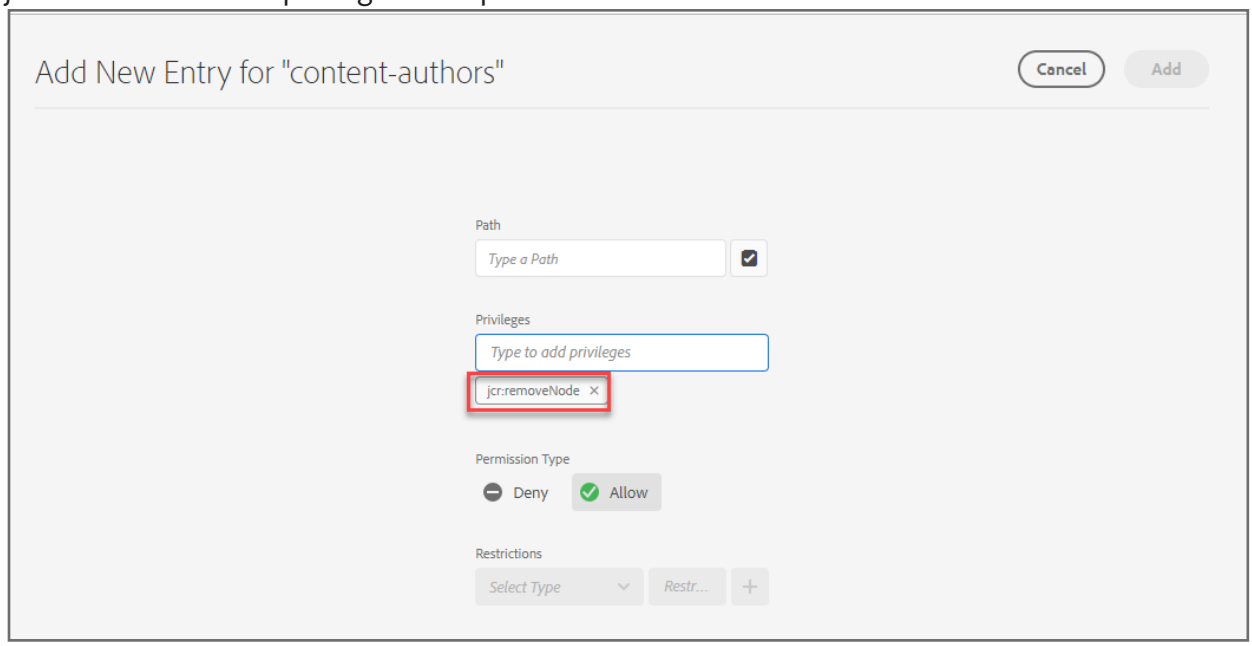
Setting Permissions on Pages

Using the Permissions tab in the Properties pane, an author with the ability to manage users can grant or deny access to a page and all of its children.



Managing ACLs directly

An administrator can manage access control policies by using the Permissions console, accessed at **Tools** > **Security** > **Permissions**. The ACEs displayed by this console contain granular JCR privileges, as defined in the Java Solution Request (JSR) 283 specification. Each privilege is either Allowed or Denied. To take any action in the repository, aggregates of privileges are typically required. For example, in order to actually remove a node requires the privilege jcr:removeNode on that node and the jcr:removeChildNodes privilege on the parent node.



NOTE: If you are going to be using the Permissions Console to define ACLs, you should read and understand Chapters 9 Permissions and Capabilities and 16 Access Control Management in the JCR v2.0 Specification (known as JSR 283). https://jcp.org/en/jsr/detail?id=283 and https://docs.adobe.com/docs/en/spec/jcr/2.0/index.html.

Exercise 1: Create and configure users and groups in AEM

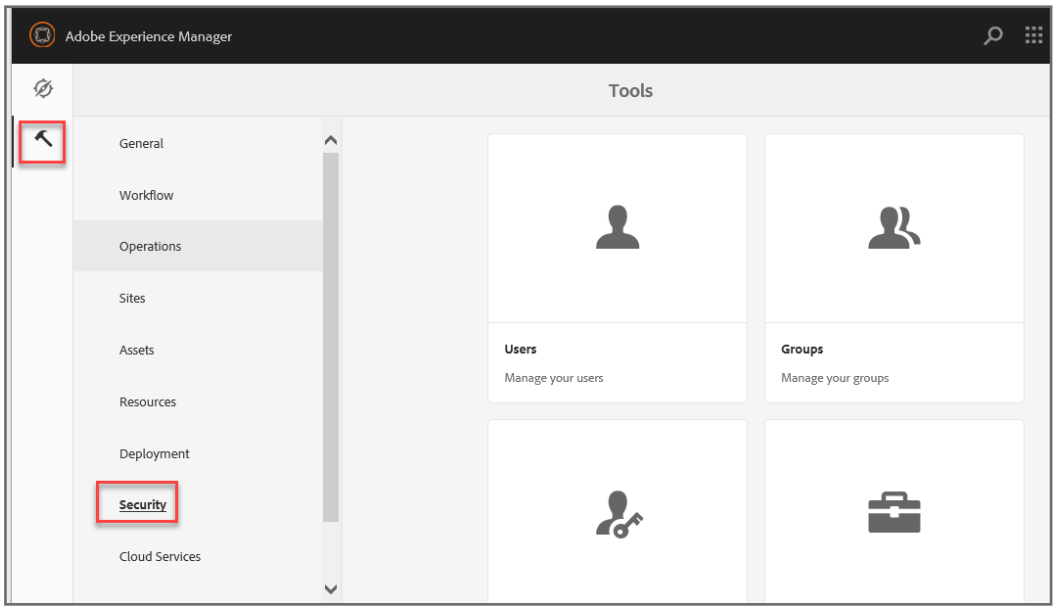
In this hands-on exercise, you will create users and groups completely in AEM as an administrator without using any other development or administrative tools.
This exercise includes three tasks:

1. Create a new user
2. Create a new group
3. Add a group to a permissioned group
4. Provide specific permissions

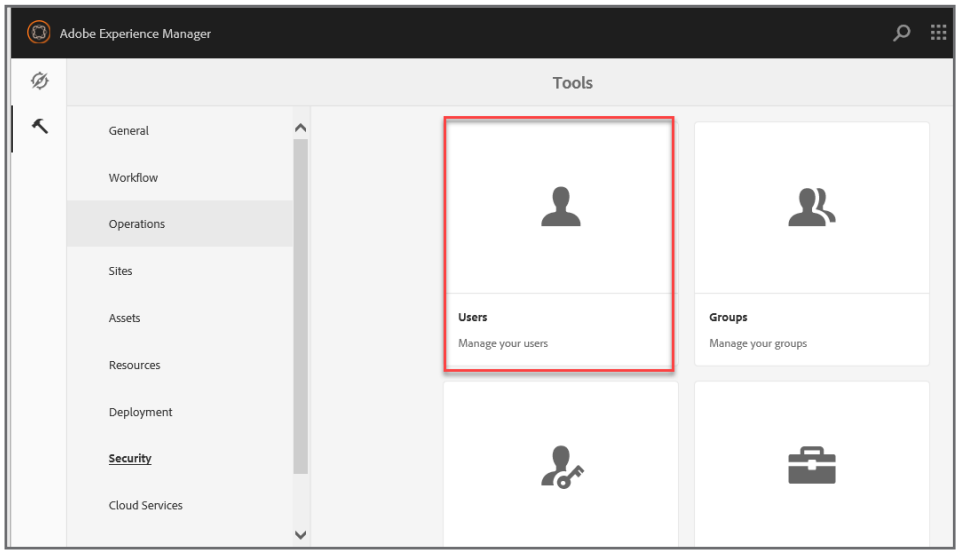
Task 1: Create a new user

Create Chuck Grant, a new user in AEM.

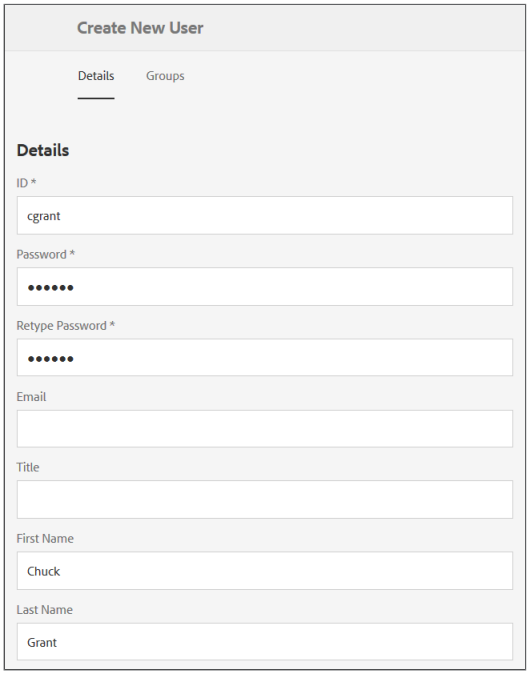
1. In AEM, navigate to **Tools** > **Security**.



2. Click **Users**. The **User Management** page is displayed.



3. Click **Create** in the upper right. The **Create New User** page is displayed.
4. Provide the following new user details in the **Create New User** page. Ensure all required fields are populated.
   a. **ID**: **cgrant**
   b. **Password**: **cgrant**
   c. **First Name**: **Chuck**
   d. **Last Name**: **Grant**



5. Click **Save & Close** in the upper right. Your new user (Chuck Grant) is now created. A success message displays on the **User Management** page.
6. Scroll down on the **User Management** page to locate your newly created user.

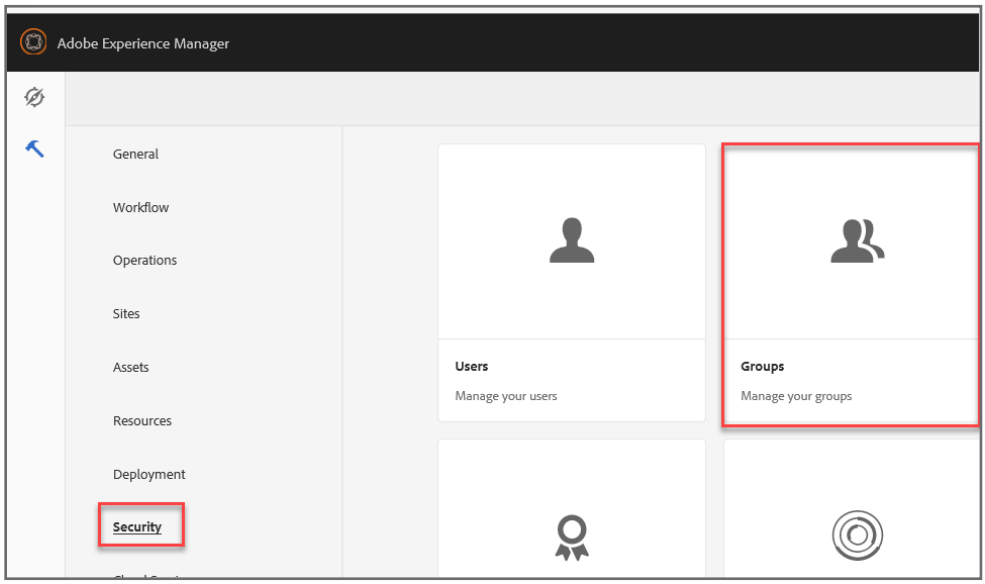
Note: By default, all users are displayed alphabetically by ID (Login Name).

Tip: To locate a user (or group), use the built-in search feature. For example, if you want to locate Virginia Armstrong's user account, type "/" to search within AEM and type "Virginia". This will retrieve the corresponding user account, which may be easier than scrolling to the bottom of the User Management page.

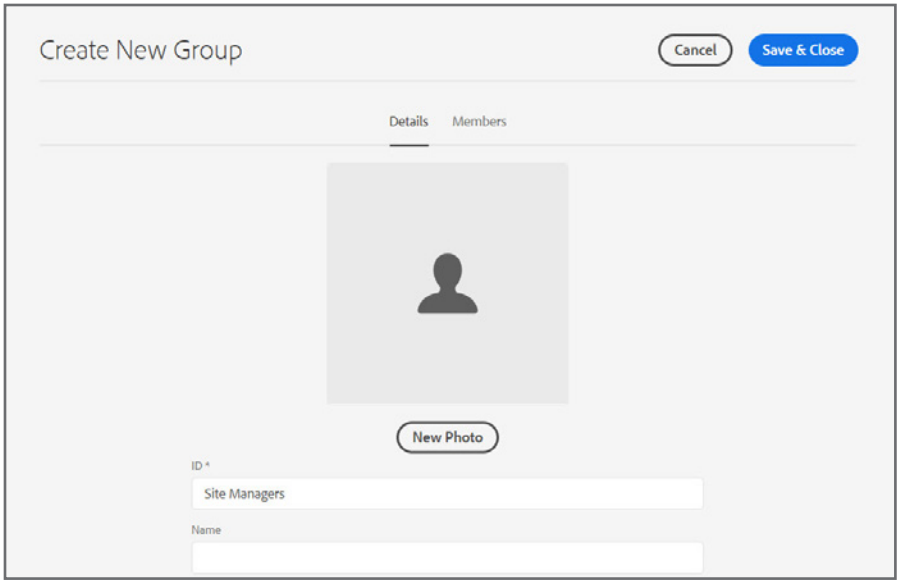
Task 2: Create a group and add a user to the group

Create a user group called, **Site Managers**, in AEM and add **Chuck Grant** to this group.

1. Navigate to **Tools** > **Security** > **Groups**. The **Group Management** page is displayed.

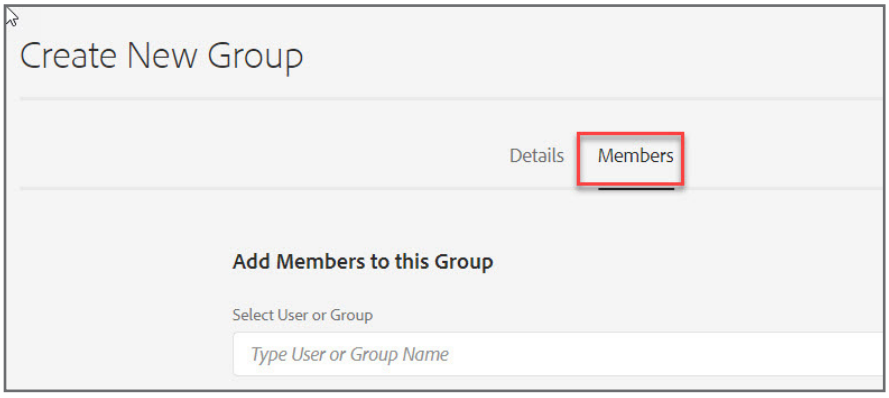


2. Click **Create** in the upper right. The **Create New Group** page is displayed.
3. In the **ID** field, type **Site Managers**.

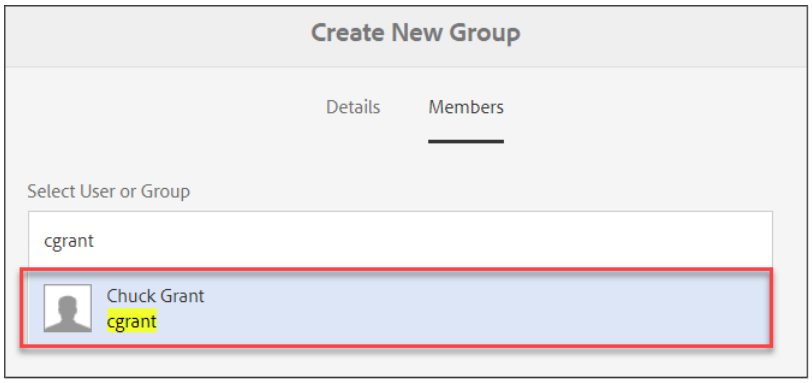


Note: DO NOT click **Save & Close** just yet.

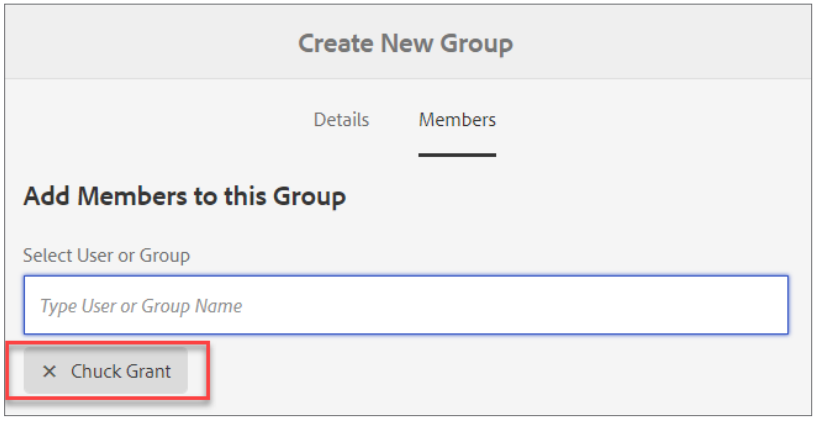
4. Click the **Members** tab.



5. Type **cgrant** into the **Select User or Group** field. This will retrieve the user you just created by auto-type in AEM. Select the **cgrant** user.

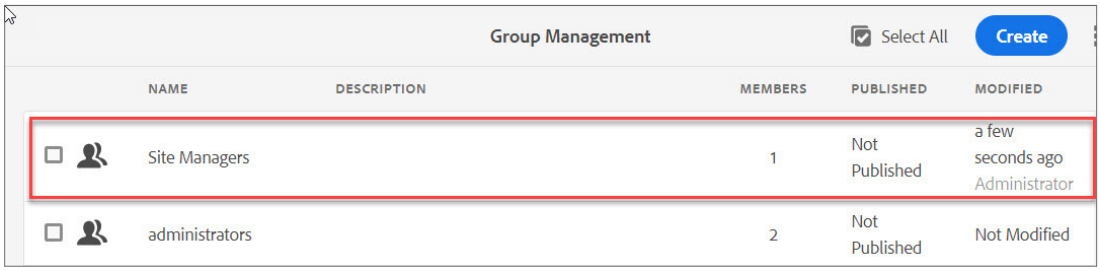


You should see **Chuck Grant's** user account now added to the group, as displayed.



6. Click **Save & Close** in the upper right. Your group is now created, and **Chuck Grant** is added as a group member. A success message displays at the bottom of the **Group Management** page.

7. In the **Group Management** page, locate your newly created group.

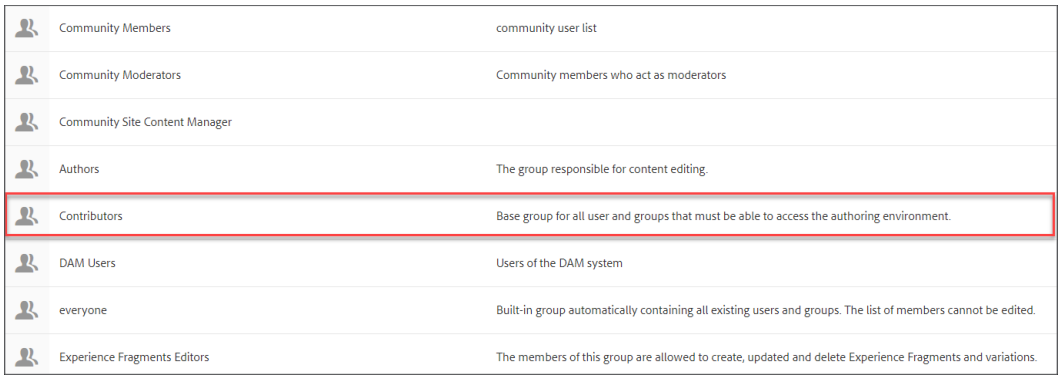


Note: So far, you have created a new user and a new group, and added your new user to your new group. However, Chuck Grant cannot successfully sign in to AEM yet. You need to add the group to which Chuck belongs (**Site Managers**) to the built-in **contributors** group in AEM. This ensures this user has the basic authorization to navigate the consoles in AEM. Technically, the **contributors** group provides read-only access to everything. This allows a user to navigate in AEM but without performing any actions on the repository (such as editing a page).

Task 3: Add a group to a permissioned group

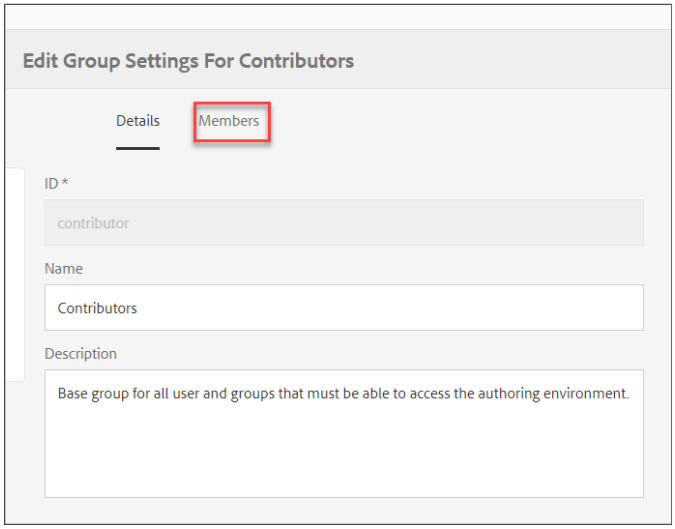
Add the **Site Managers** group to the contributors group.

1. On the **Group Management** page, locate and click on the **Contributors** group.

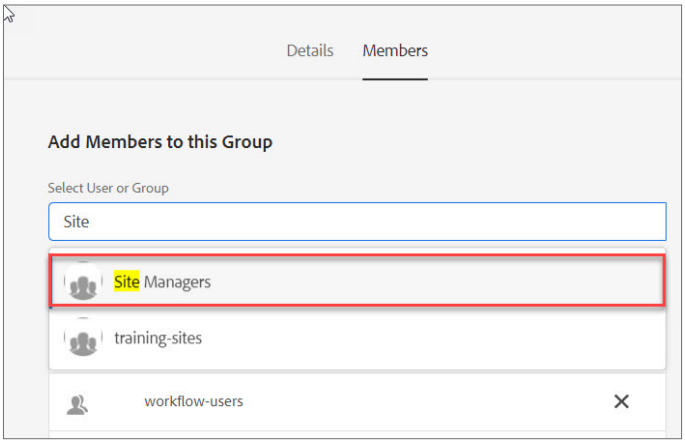


The group settings page for **Contributors** is displayed

2. In the **Group Settings** page, click the **Members** tab.



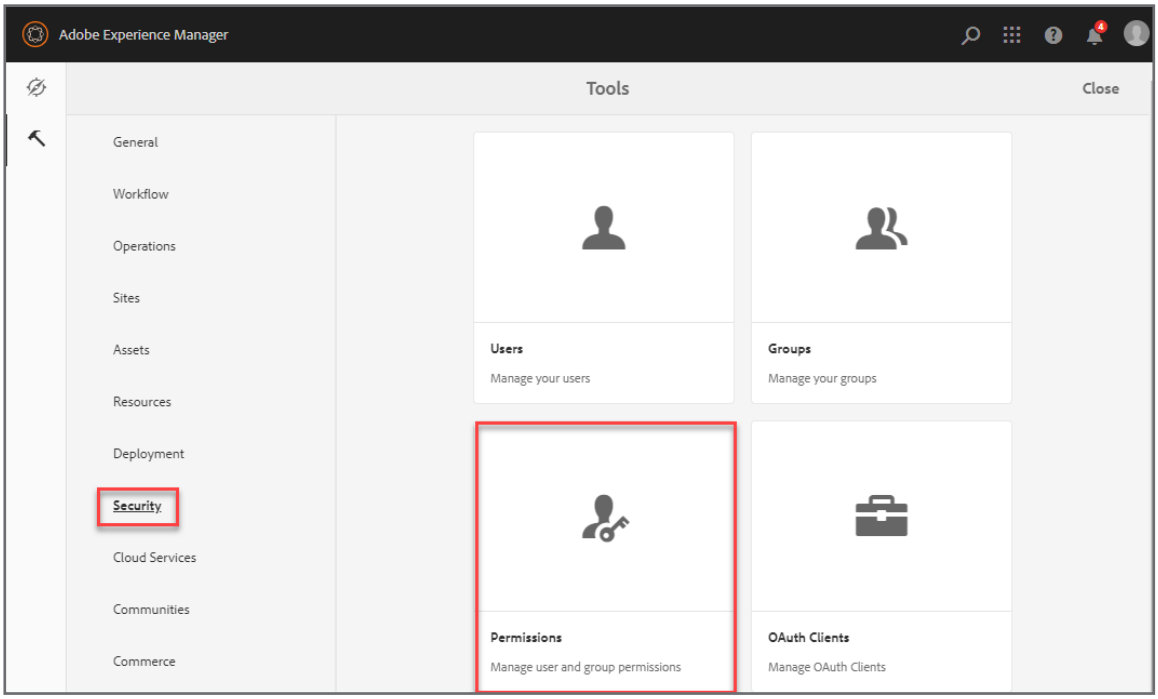
3. In the **Add Members to this Group**, type **Site Managers** in the **Select User or Group** field. Select **Site Managers** in the drop-down list to add the **Site Managers** group to the **Contributors** group.



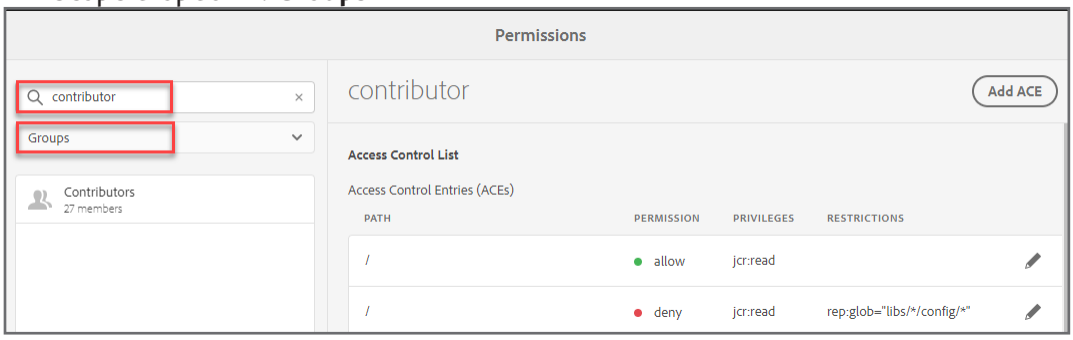
4. Click **Save & Close** in the upper right.

Note: Chuck Grant now has the appropriate minimum permissions to sign in and navigate AEM consoles and features. Because Chuck Grant is a part of the **Site Managers** group, Chuck will also have the same access as the contributors group now.

5. Navigate to **Tools** > **Security** > **Permissions,** as shown. The **Permissions** window opens.



6. In the left pane, type the following information in the corresponding fields listed:
   - Search field: **contributor**
   - Scope dropdown: **Groups**



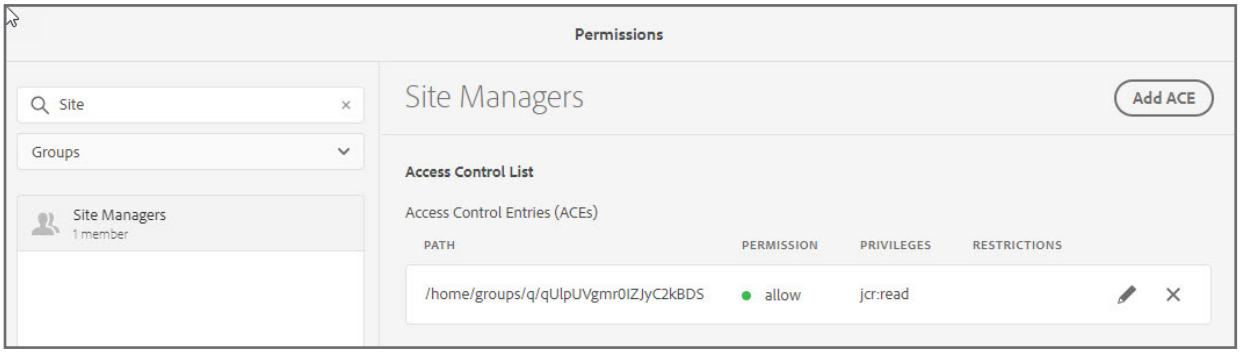
7. Click the Contributors group that appears, as a result of your search in the left pane. The corresponding contributor page opens to the right.

8. Notice that the Contributors group has **jcr:read permission** at the repository root. Since Access Control policies cascade down the tree, this means that the Contributors group has read permissions at **/apps, /libs**, **/content** and so on. Read access to **/libs** and the children of **/libs** gives sufficient permissions to login and read content in the repository.

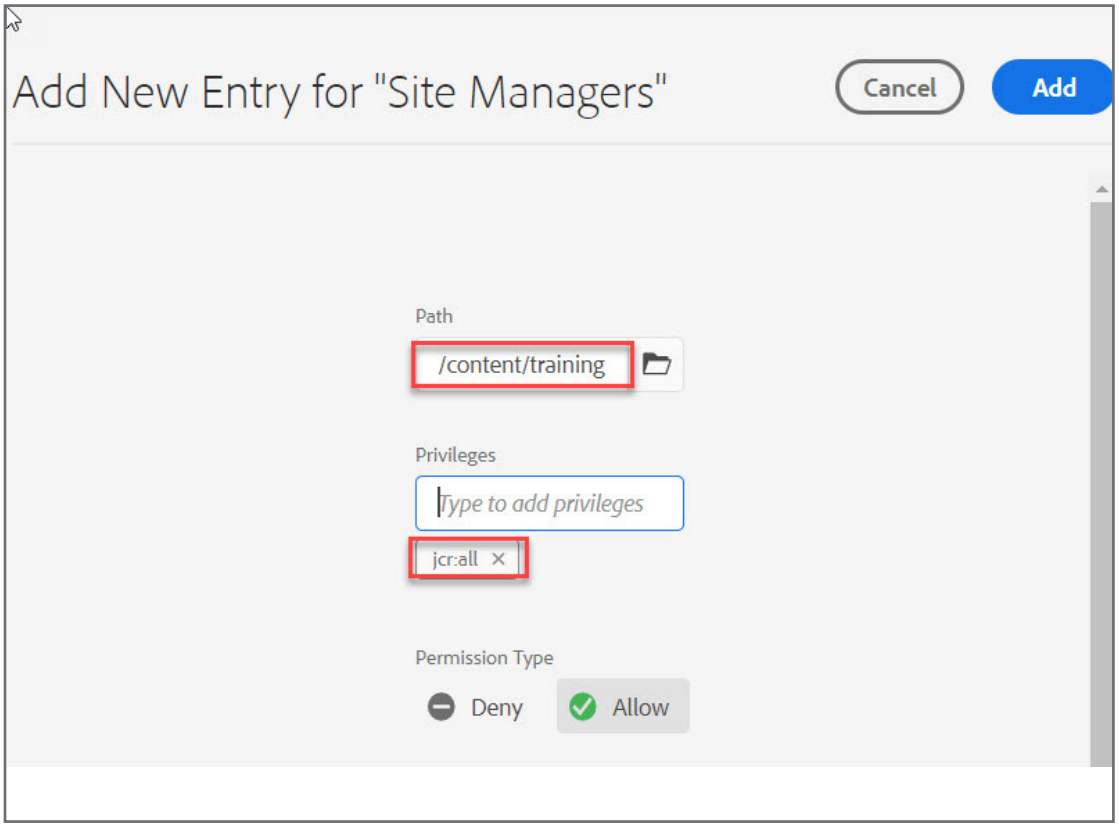
Since Chuck Grant is a member of the **Site Managers** group and the **Site Managers** group is a member of the **Contributors** group, Chuck Grant now has sufficient privileges to login and read content in the repository.

Task 4: Provide specific permissions to the group

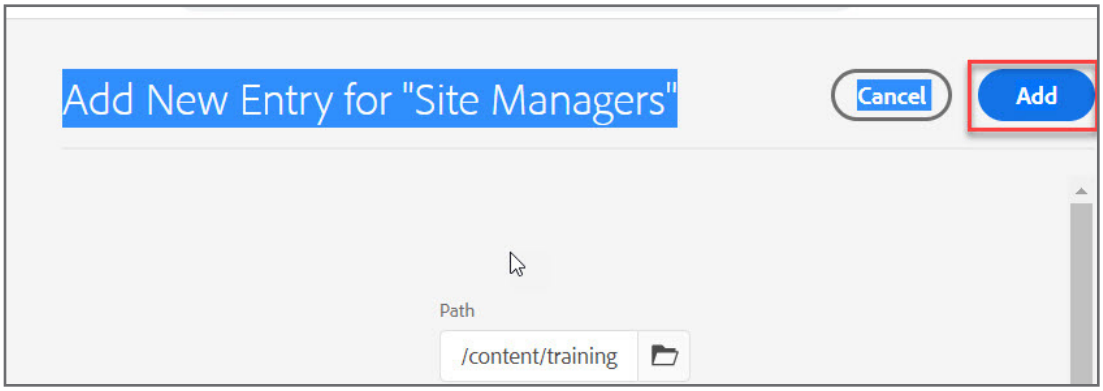
1. Navigate to **Tools** > **Security** > **Permissions**.
2. Type the following values for the **Search field**. The Site Managers group is displayed, as shown:
   - › Search field: **Site**
   - › Scope dropdown: **Groups**.



3. Click **Add ACE** button located at the top right. The **Add New Entry for "Site Manager"** screen is displayed.
4. Enter the value for **Path** and select **Privileges** from the drop-down list, as shown:
   - › Path: **/content/training** (Click **Open Selection Dialog** and select the path)
   - › Privileges: **jcr:all**



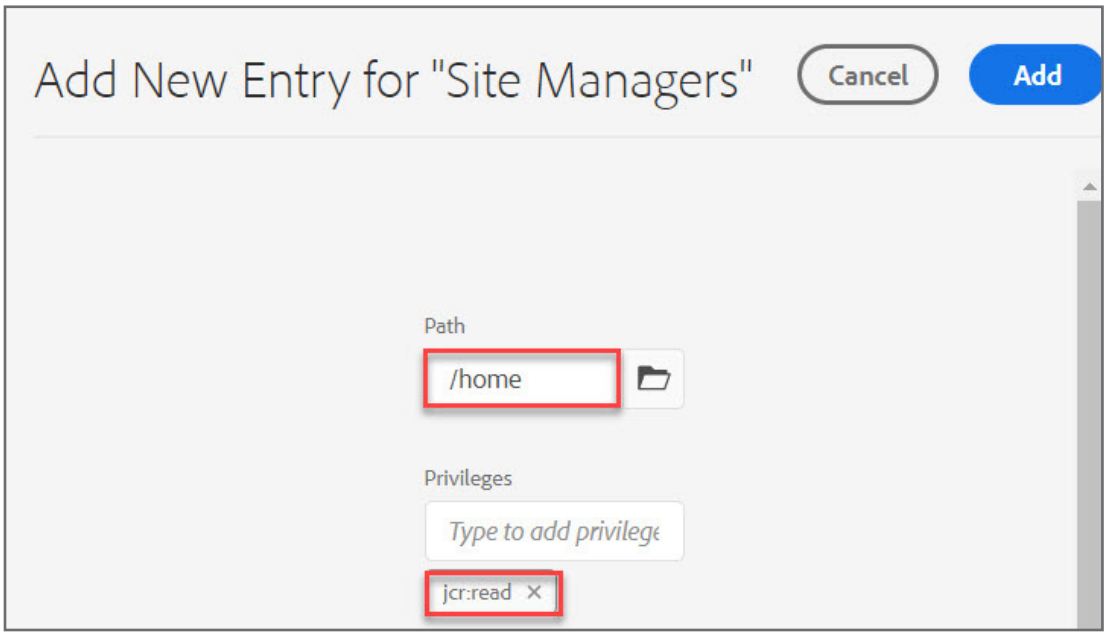
5. Click the **Add** button at the top, as shown:



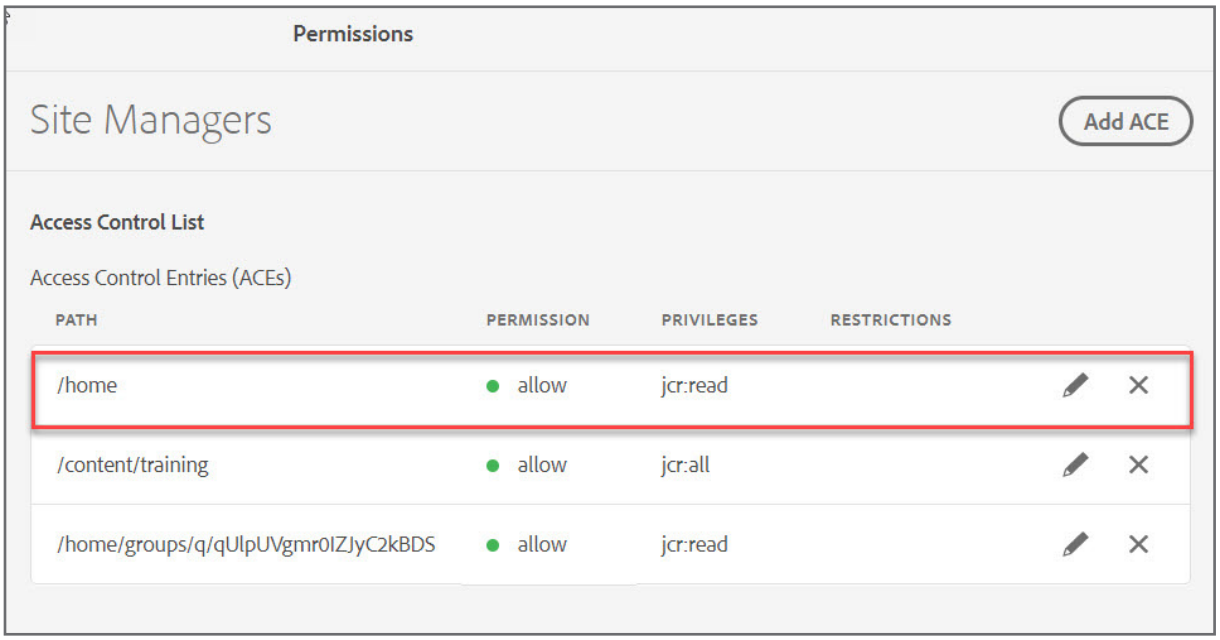
The Permissions are added, as shown:

6. Click **Add ACE** to give more permissions to **Site Managers.** The **Add New Entry for "Site Manager"** screen is displayed.

7. Enter the value for **Path** and select **Privileges** from the drop-down list, as shown:
   - Path: **/home** (Click **Open Selection Dialog** and select the path)
   - Privileges: **jcr:read**



8. Click the **Add** button at the top. The Permissions screen is displayed, as shown:

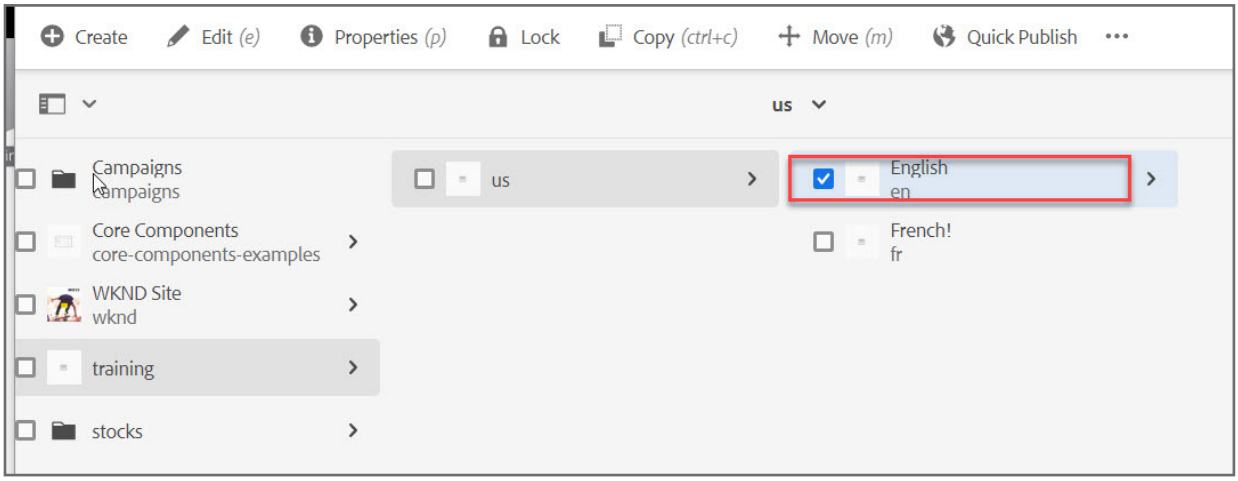


Note: We have now given the Site Managers group the access to give permission control to other users. This means, the user we created, Chuck Grant will have the permission to browse the website, edit the website or publish pages.

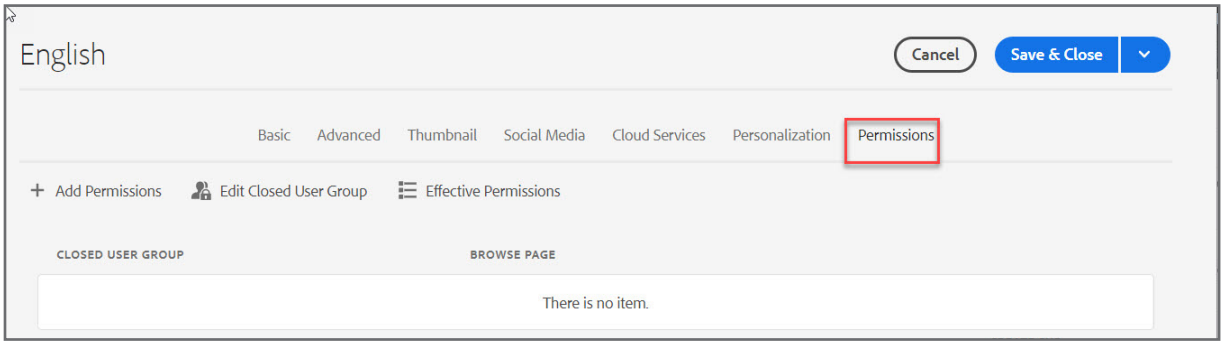
9. Log out as **admin**.

10. Login to **AEM** as **Chuck Grant**

11. Open http://localhost:4502/sites.html and select **training** > **us** > **en**, as shown:

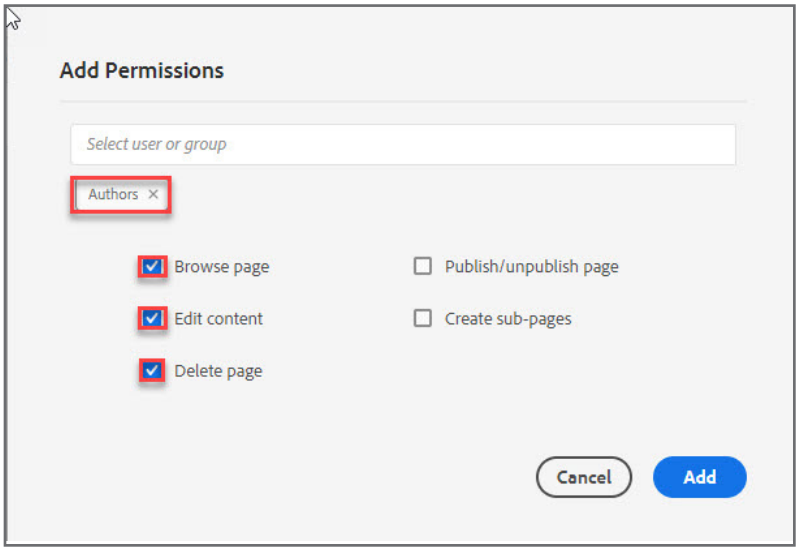


12. Click **Properties** and click the **Permissions** tab, as shown:

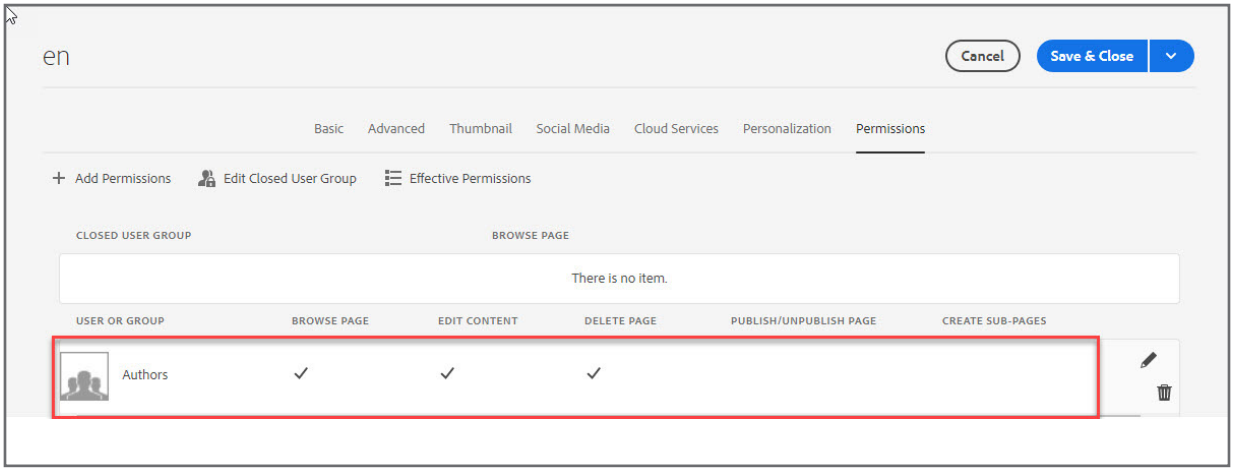


13. Click **Add Permissions**. The **Add Permissions** window opens.

14. In the Select user or group field, search for and select group **Authors** and select the following permissions for the group, as shown:



15. Click **Add**. The Permissions gets added, as shown:



16. Click **Save&Close**.

Note: Now any user that is added to the **Authors** group, will be able to browse, edit and delete from this website en.

17. Optionally, add the user **Alison Smith** to the **Authors** group and then login as **Alison Smith** to the site and verify the permissions.

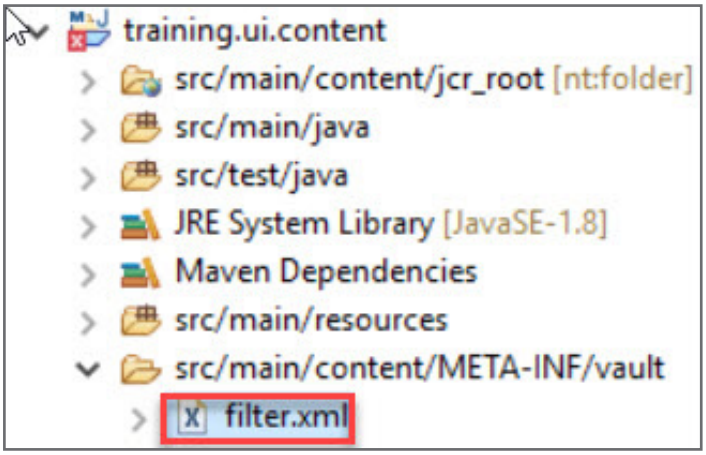
Exercise 2: Import permissions into the Maven project

In the previous exercise you setup users, groups and permissions within your local AEM instance. Depending on the requirements of your project you may need to add the users, groups, or permissions back to your maven project so they can be deployed elsewhere. In this exercise you will learn how to find the nodes needed to import back into your maven project.

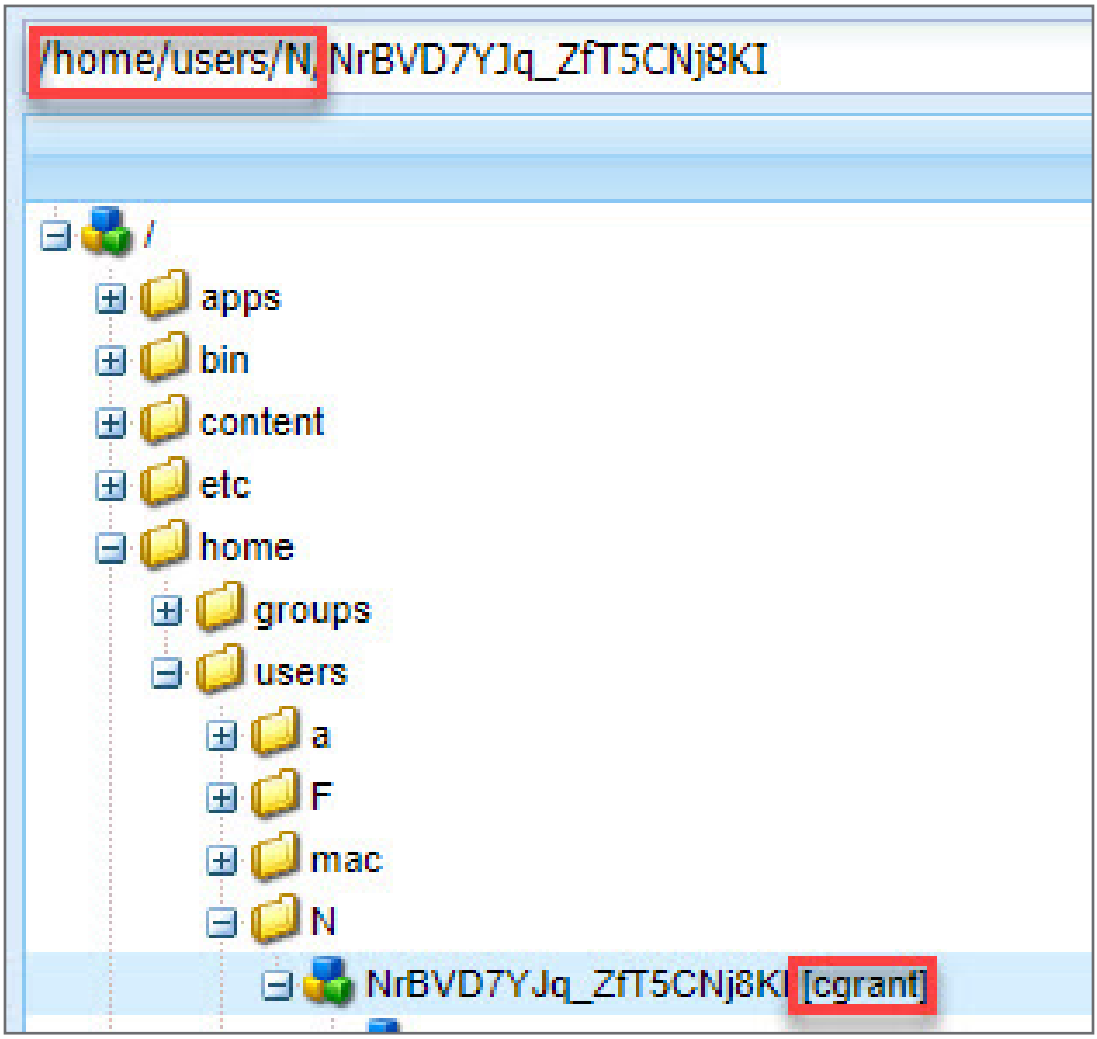
.

Note: Users and groups fall under mutable content.

1. Open **Eclipse**.
2. Navigate to: **training.ui.content** > **src/main/content/META-INF/vault** > **filter.xml**



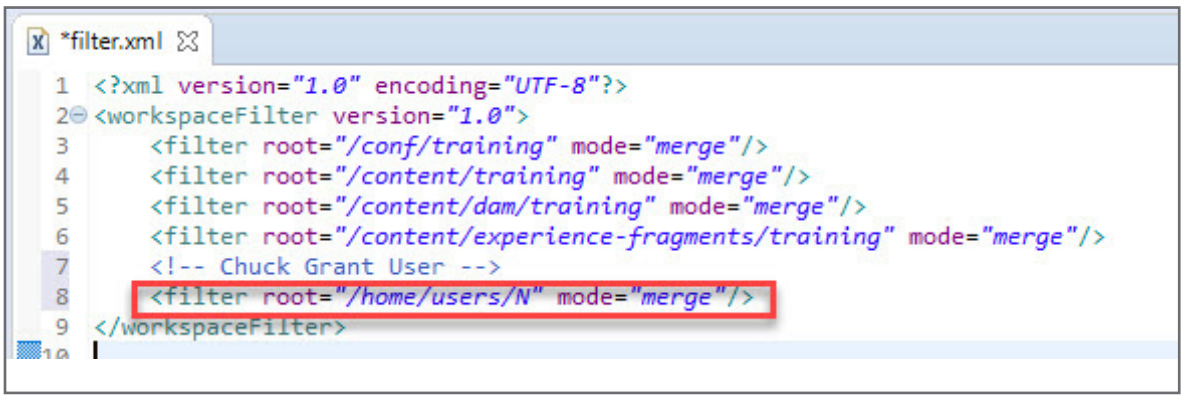
3. Double-click **filter.xml** to open it and view the current filter that will be synchronized with this module. Keep the file open.
4. Go to **CRXDE Lite** and navigate to **/home/users/**.
5. Find the user **Cgrant**, as shown:



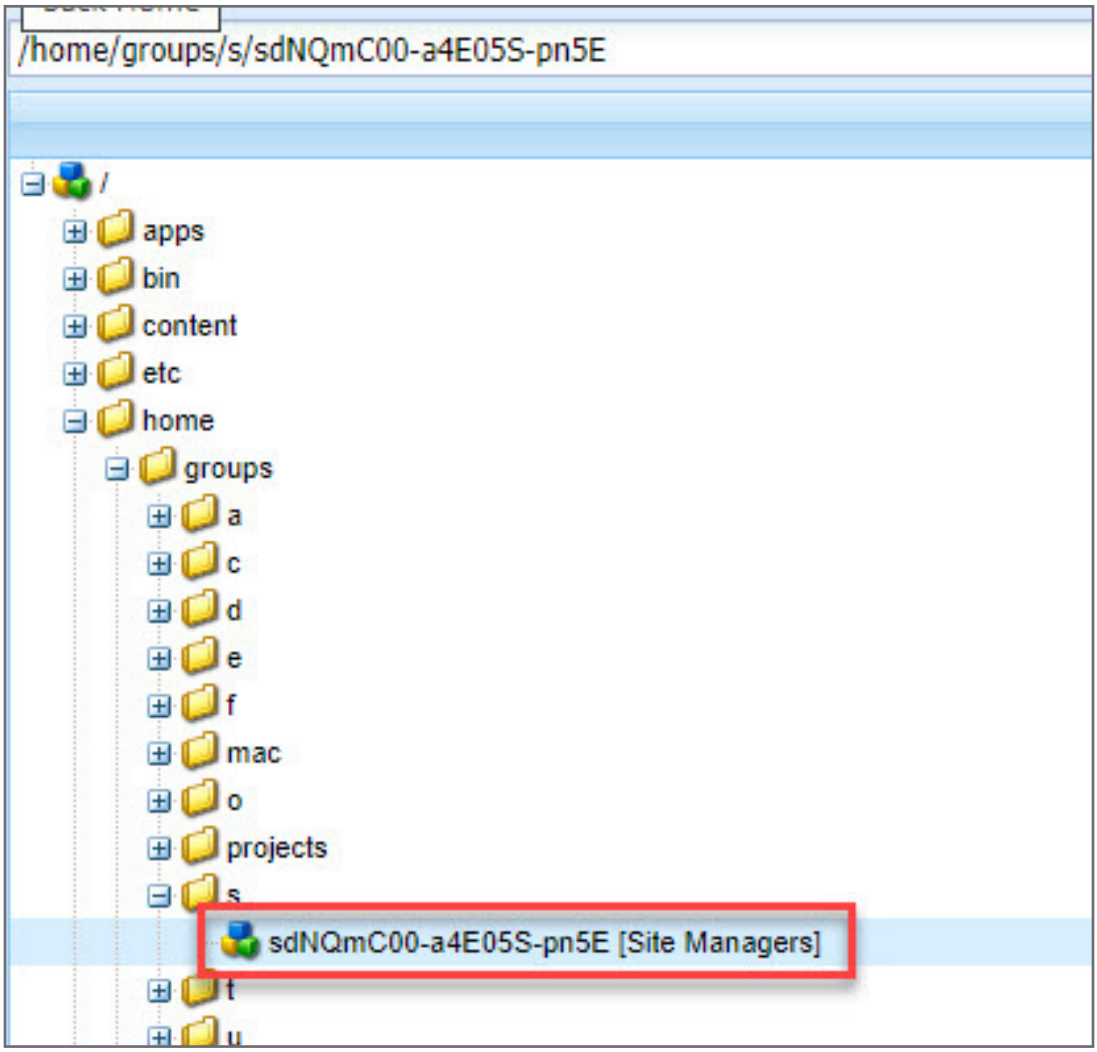
Note: When users are created, the node name is a hash. More than likely your user node name will not be the same as the screenshot. Just keep looking in the /home/users subfolders to find **cgrant**.

6. Copy the parent path of the user which is **/home/users/N** (The user node might be different for you).

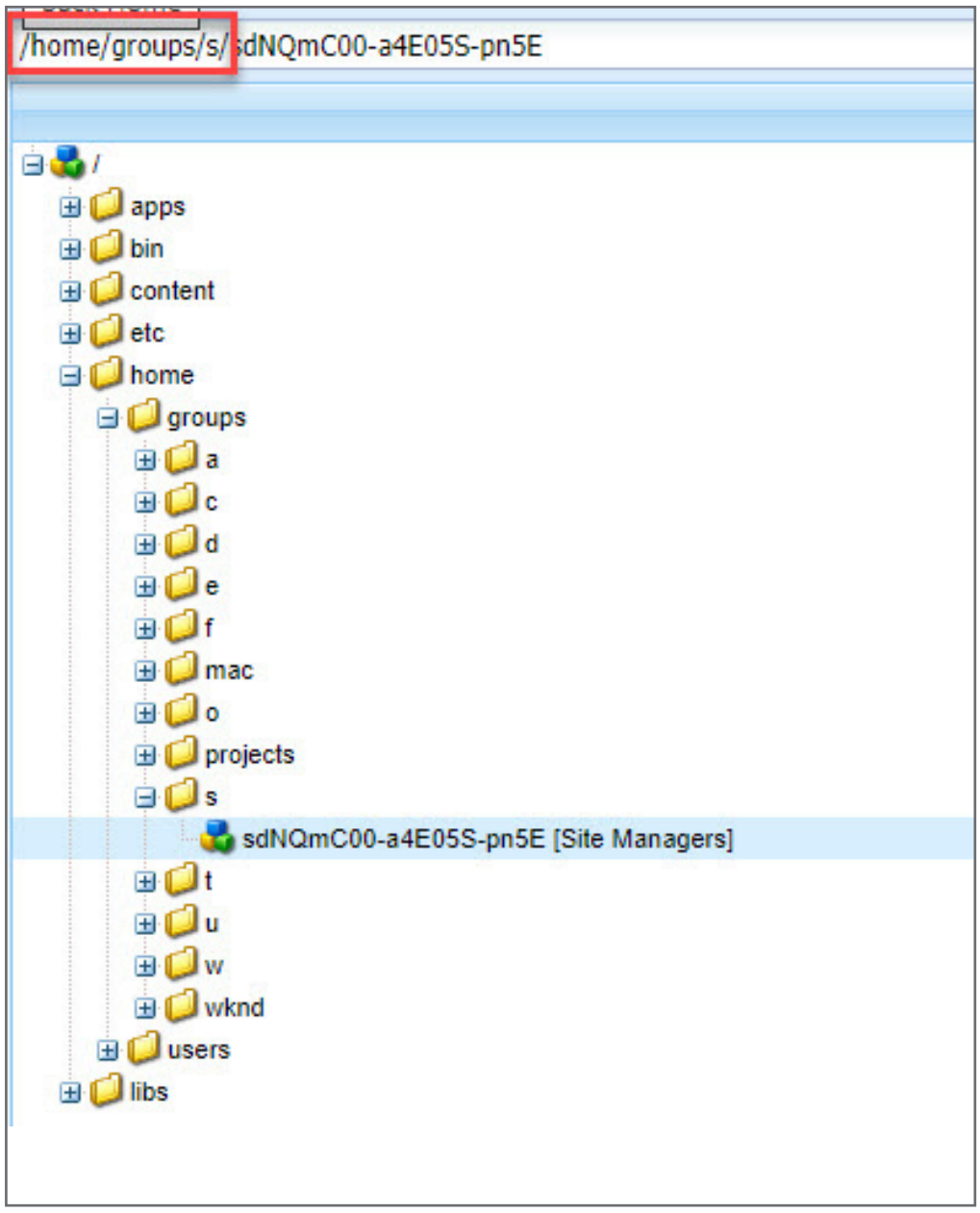
7. Back in Eclipse, paste the copied path to the **filter.xml**, as shown:



8. Save the changes.
9. Keep the file open.
10. Go to **CRXDE Lite** and navigate to **/home/groups/**.
11. Find the group **Site Managers** , as shown:

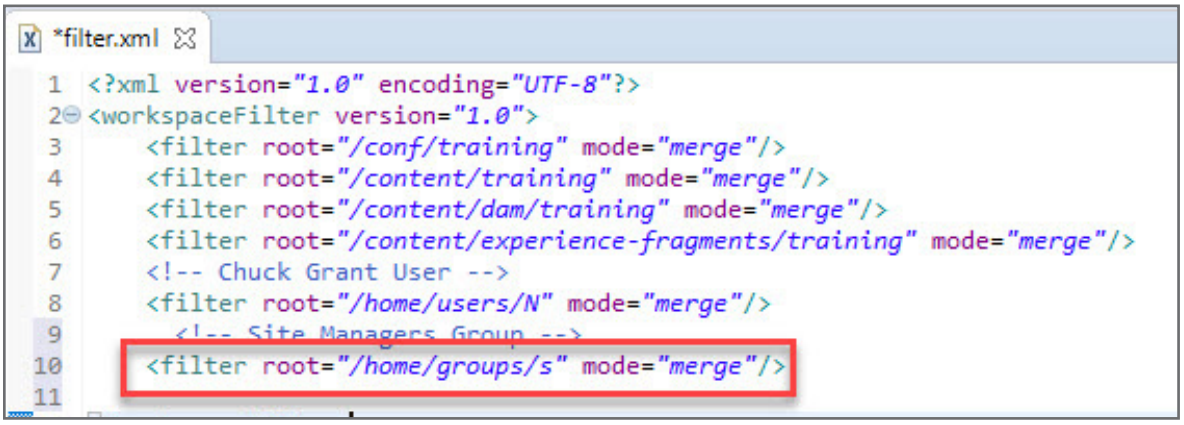


12. Copy the parent path of the group:



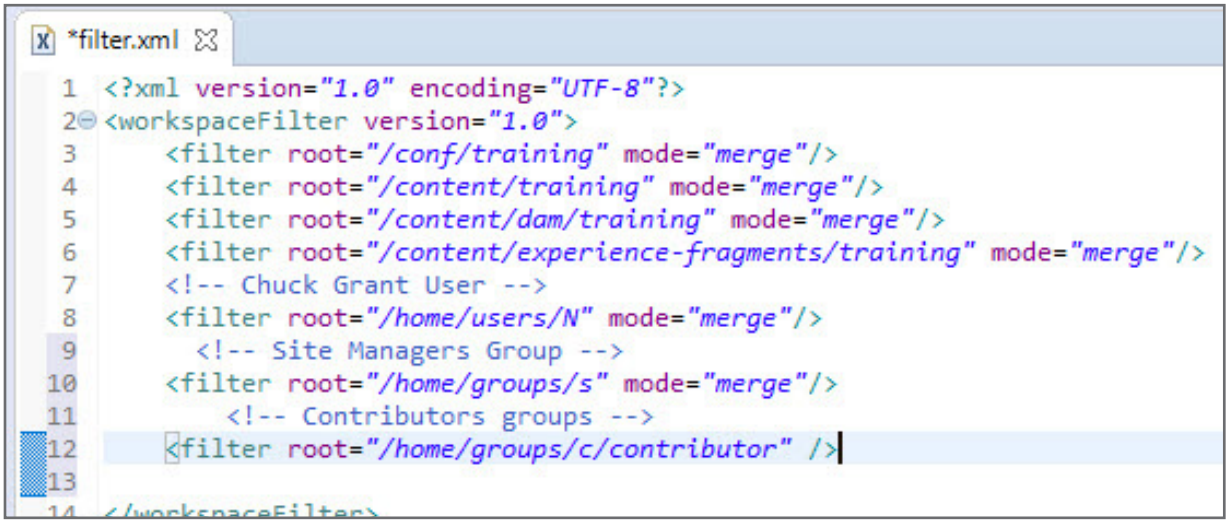
Note: When groups are created, the node name is a hash. More than likely your group node name will not be the same as the screenshot. Just keep looking in the /home/groups subfolders to find Site Managers.

13. Back in **Eclipse**, paste the copied path to the **filter.xml**, as shown:



14. Similarly, find the **contributor** group in CRXDE Lite.

15. Copy the path and paste it to **filter.xml** in **Eclipse**, as shown:

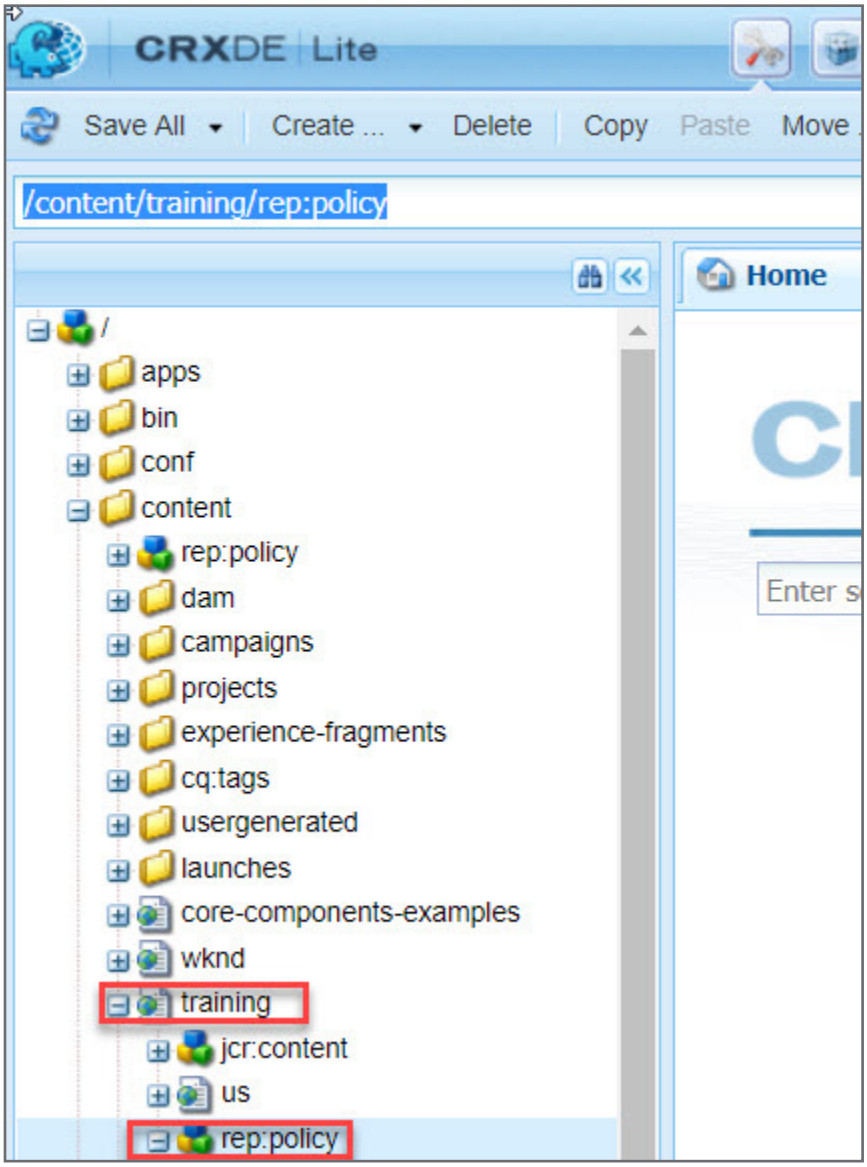


16. Save the changes.

To add Permissions given to Site Managers group:

17. Recall the permissions that you added in Exercise 1 - Task 4 for the group **Site Managers:**

18. Go to **CRXDE Lite**, find the **rep:policy** for the **training** site, as shown:



19. Copy the path and paste it to **filter.xml**, as shown:

```
<?xml version="1.0" encoding="UTF-8"?>
<workspaceFilter version="1.0">
    <filter root="/conf/training" mode="merge"/>
    <filter root="/content/training" mode="merge"/>
    <filter root="/content/dam/training" mode="merge"/>
    <filter root="/content/experience-fragments/training" mode="merge"/>
    <!-- Chuck Grant User -->
    <filter root="/home/users/N" mode="merge"/>
      <!-- Site Managers Group -->
    <filter root="/home/groups/s" mode="merge"/>
        <!-- Contributors groups -->
    <filter root="/home/groups/c/contributor" />
      <!-- add permissions that were manually given to site-managers -->
    <filter root="/content/training/rep:policy" />

</workspaceFilter>
```

20. Similarly, find the **rep:policy** for **home** in **CRXDE Lite** which is **/home/rep:policy** and copy the path**.**

21. In Eclipse, paste the copied path to **filter.xml**, as shown:

```
<?xml version="1.0" encoding="UTF-8"?>
<workspaceFilter version="1.0">
    <filter root="/conf/training" mode="merge"/>
    <filter root="/content/training" mode="merge"/>
    <filter root="/content/dam/training" mode="merge"/>
    <filter root="/content/experience-fragments/training" mode="merge"/>
    <!-- Chuck Grant User -->
    <filter root="/home/users/N" mode="merge"/>
      <!-- Site Managers Group -->
    <filter root="/home/groups/s" mode="merge"/>
        <!-- Contributors groups -->
    <filter root="/home/groups/c/contributor" />
      <!-- add permissions that were manually given to site-managers -->
    <filter root="/content/training/rep:policy" />
        <filter root="/home/rep:policy" />
</workspaceFilter>
```

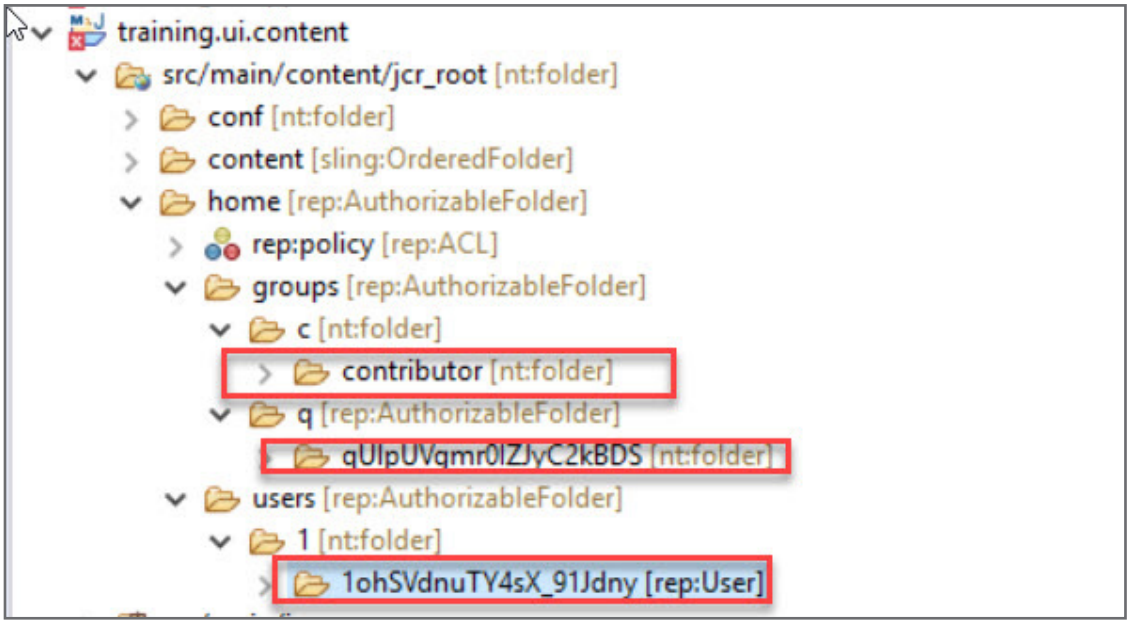
22. Save the changes.

23. Right-click on **training.ui.content > src/main/content/jcr_root** and select **Import from server.**

24. Click **Finish.**

Note: If you had any changes in AEM (based on **filter.xml)**, those nodes and properties are now imported to your project.

25. In **Eclipse**, navigate to **training.ui.content** > **src/main/content/jcr_root[nt:folder]** > **home** and verify the imported groups and users, as shown:



(Optional) Exercise 3: Import the System User: training-user

Recall from a previous module you created a system used within AEM called training-user. This user currently only resides in your local AEM instance. If you want this user to be a part of your code, ensuring it will run properly when deployed, you need to add the system user to your maven project. Use the previous exercise as a guide to accomplish this exercise.

Pseudo Steps:

1. Find the system user in /home/users/system and add it to the filter.xml under ui.content.
2. Find the group(s) the user is added to in /home/groups and add it to the filter.xml under ui.content.
3. Find the permissions attached to the user and add them to the ui.content filter.xml
4. Run import from the server in Eclipse.

Writing Tests

Introduction

In Adobe Experience Manager (AEM), the testing frameworks, such as Maven, Mockito, and Sling JUnit 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 the 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 replicates 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 an 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 for the creation of custom argument matchers or using existing hamcrest matchers

Performing Unit Tests

In this module, 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.

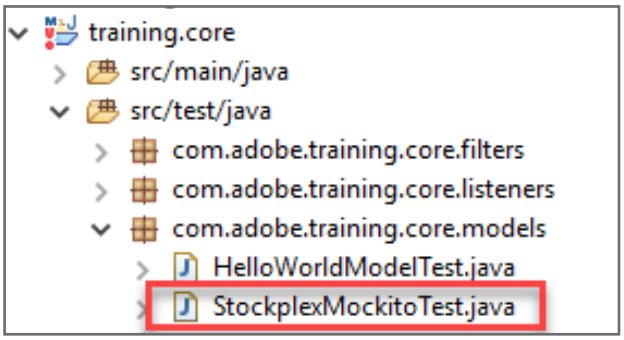
Exercise 1: Create unit tests using Mockito

In this lab exercise, you will create a unit test and test the Stockplex java class you created earlier. This is a basic unit Test outside the server. If you do not have the Stockplex.java class implemented, then you will need to go back and perform those exercises first.

1. Launch **Eclipse**. The Eclipse Workspace opens.
2. In Project Explorer, navigate to **training.core** > **src/test/java**.
3. Copy the file **StockplexMockitoTest.java** from **Exercise_Files** under **core/src/test/java/com/adobe/training/core/models/**.

Note: The code is provided as part of **Exercise_Files** under **core/src/test/java/com/adobe/training/core/models/**.. Copy and paste the file from the exercise files referenced to StockplexMockitoTest.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. In Eclipse, right-click **com.adobe.training.core.models** and paste the file. The **StockplexMockitoTest.java** class is created, as shown:



5. The **StockplexMockitoTest.java** file opens in the editor.

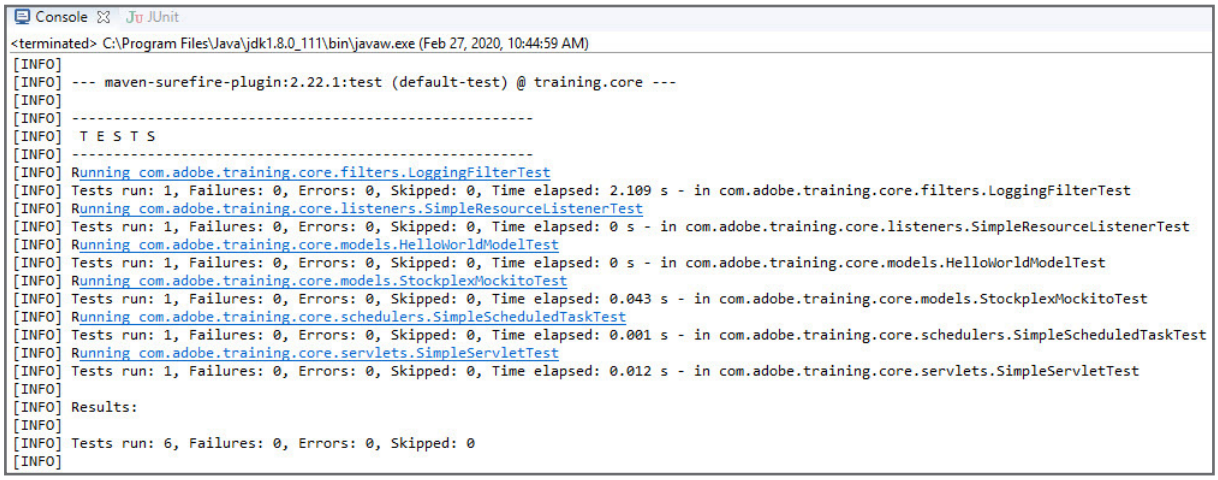
```
1.  package com.adobe.training.core.models;
2.
3.  import org.junit.jupiter.api.BeforeEach;
4.  import org.junit.jupiter.api.Test;
5.  import org.junit.jupiter.api.extension.ExtendWith;
6.  import org.mockito.junit.jupiter.MockitoExtension;
7.
8.  import java.util.Random;
9.
10. import static org.junit.jupiter.api.Assertions.assertNotNull;
11. import static org.junit.jupiter.api.Assertions.assertTrue;
12. import static org.junit.jupiter.api.Assertions.assertFalse;
13. import static org.mockito.Mockito.*;
14.
15. /**
16.  * Tests the Stockplex model using the Mockito testing framework.
17.  * This is good for simple test cases, where the stubbing is fairly simple like in this example.
18.  * For more complex mocking, use the Sling Mock API or the AEM Mocks when AEM objects need to be mocked.
19.  *
20.  * Note that the testable class is under /src/main/java:
21.  * com.adobe.training.core.models.Stockplex.java
22.  *
23.  *  To correctly use this testing class:
24.  *  -put this file under training.core/src/test/java in the package com.adobe.training.core.models
25.  *
26.  */
27. @ExtendWith(MockitoExtension.class)
28. public class StockplexMockitoTest {
29.
30.     private Stockplex stock;
31.
32.     @BeforeEach
33.     public void setup() throws Exception {
34.
35.         //Adapt the Resource if needed
36.         Stockplex STOCKMODEL_MOCK = mock(Stockplex.class);
37.
38.         stock = STOCKMODEL_MOCK;
39.
40.         //Setup stock symbol
41.         final String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
42.         final int N = alphabet.length();
43.         Random r = new Random();
44.         String str = "";
45.         for (int i = 0; i < 4; i++) {
46.             str = str + alphabet.charAt(r.nextInt(N));
47.         }
48.         when(STOCKMODEL_MOCK.getSymbol()).thenReturn(str);
49.
50.         //Setup current trade price
51.         Random rand = new Random();
52.         double n = Math.round(100*(rand.nextInt(150) + 100)+rand.nextDouble())/100; //
    random value between 100.00 and 150.00
53.         when(STOCKMODEL_MOCK.getCurrentPrice()).thenReturn(n);
54.
55.     }
56.
57.     @Test
58.     void testGetLastTradeValue() throws Exception{
59.         assertNotNull(stock, "lastTradeModel is null»);
60.         assertTrue(stock.getCurrentPrice() > 100, "current value is incorrect");
61.         assertFalse(stock.getSymbol().isEmpty(), "stock symbol is incorrect");
62.
63.         // Verify that only those two methods were called on the mocked object
64.         verify(stock).getCurrentPrice();
65.         verify(stock).getSymbol();
66.
67.         // Verify that no other interaction occurred on the mocked object
68.         verifyNoMoreInteractions(stock);
69.             // one below is when we want to make sure that a specific method was never invoked
70.         verify(stock, never()).getSummary();
71.     }

72. }
```

6. Examine the code.

Note: This class will test the Java class Stockplex.java.

7. Right-click **training.core** and select **Run As** > **Maven test**. The Junit Test starts.

8. Verify the test ran successfully, as shown:



Sling Mocks

Sling Mocks are a part of the AEM archetype.

The Maven dependency for Sling Mocks are as follows:

Junit 5:

```
<dependency>
  <groupId>org.apache.sling</groupId>
  <artifactId>org.apache.sling.testing.sling-mock.junit5</artifactId>
</dependency>
```

The mock implementation supports:

- ResourceResolver implementation for reading and writing resource data using the Sling Resource API Backed by a mocked or real Jackrabbit JCR implementation
  - › Uses the productive Sling JCR resource provider implementation internally to do the Resource-JCR mapping.
  - › Alternatively, the non-JCR mock implementation provided by the Sling resourceresolver-mock implementation can be used.
- AdapterManager implementation for registering adapter factories and resolving adaptions
  - › The implementation is thread-safe, so it can be used in parallel running unit tests.
- SlingScriptHelper implementation providing access to mocked request/response objects and supports getting OSGi services from the mocked OSGi environment.
- Implementations of the servlet-related Sling API classes like SlingHttpServletRequest and SlingHttpServletRequest
  - › It is possible to set request data to simulate a certain Sling HTTP request.
- Support for Sling Models (Sling Models API 1.1 and Impl 1.1 or higher required), all relevant Sling Models services are registered by default.
- Additional services: MimeTypeService
- Context Plugins

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. Add the sling-mock dependency
2. Create a unit test with sample data
3. Run the test

Task 1: Add the Sling-mock dependency

1. In Project Explorer, navigate to **training** > **pom.xml**.
2. Double-click **pom.xml** to open it in the editor. The pom.xml file opens.
3. Add the dependency **org.apache.sling.testing.sling-mock.junit5**, as shown:

```
685            <!-- TRAINING DEPENDENCIES -->
686
687            <!-- Dependency for StockplexSlingMockTest.java -->
688            <dependency>
689                <groupId>org.apache.sling</groupId>
690                <artifactId>org.apache.sling.testing.sling-mock.junit5</artifactId>
691                <version>2.3.4</version>
692                <scope>test</scope>
693            </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.
The code is available as part of the parent-pom.xml under **Exercise_Files/training-files/Writing-Tests/**.

4. In the **Exercise_Files**, navigate to **/training-files/Writing-Tests/.** Open **core-pom.xml**.
5. Copy the **junit5** dependency at the bottom:

```
    <!-- Dependency for StockplexSlingMockTest.java -->
    <dependency>
        <groupId>org.apache.sling</groupId>
        <artifactId>org.apache.sling.testing.sling-mock.junit5</artifactId>
    </dependency>
```

6. In Project Explorer, navigate to **training** > **core** > **pom.xml**.
7. Double-click **pom.xml** to open it in the editor. The **pom.xml** file opens.
8. Add the **org.apache.sling.testing.sling-mock.junit5** dependency, as shown:

```
<!-- Dependency for StockplexSlingMockTest.java -->
        <dependency>
            <groupId>org.apache.sling</groupId>
            <artifactId>org.apache.sling.testing.sling-mock.junit5</artifactId>
        </dependency>
```

Note: The code is available as part of the parent-pom.xml under **Exercise_Files/backend-training-dev/training-files/Writing-Tests/**.

9. Save the changes.

Task 2: Create a unit test with sample data

1. In Project Explorer, navigate to **training.core** > **src/test/java**.
2. Copy the file **StockplexSlingMockTest.java** from **Exercise_Files** under **core/src/test/java/com/adobe/training/core/models/**.

Note: The code is provided as part of **Exercise_Files** under **core/src/test/java/com/adobe/training/core/models/**. Copy and paste the file from the exercise files referenced to StockplexSlingMockTest.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. In Eclipse, right-click **com.adobe.training.core.models** and paste the file. The **StockplexSlingMockTest.java** class is created.
4. The **StockplexSlingMockTest.java** file opens in the editor, as shown:

```
1.  package com.adobe.training.core.models;
2.
3.  import org.apache.sling.api.resource.Resource;
4.  import org.apache.sling.models.factory.ModelFactory;
5.  import org.apache.sling.testing.mock.sling.junit5.SlingContextExtension;
6.  import org.apache.sling.testing.mock.sling.junit5.SlingContext;
7.  import org.junit.jupiter.api.BeforeEach;
8.  import org.junit.jupiter.api.Test;
9.  import org.junit.jupiter.api.extension.ExtendWith;
10.
11. import com.google.common.collect.ImmutableMap;
12.
13. import java.util.Map;
14.
15. import static org.junit.jupiter.api.Assertions.assertEquals;
16. import static org.junit.jupiter.api.Assertions.assertNotEquals;
17.
18. /**
19.  * Tests the Stockplex model using the Sling Mock implementation for Sling APIs
20.  *
21.  * A Sling context (the mock environment) needs to be created to test the classes.
22.  * The implementation used in that example is the ResourceResolver mock,
23.  * to allow in-memory reading and writing resource data using the Sling Resource API.
24.  *
25.  * It also provides support for Sling Models (Sling Models API 1.1 and Impl 1.1 or higher required)
26.  *
27.  * Note that the testable class is under /src/main/java:
28.  * com.adobe.training.core.models.Stockplex.java
29.  *
30.  *  To correctly use this testing class:
31.  *  -put this file under training.core/src/test/java in the package com.adobe.training.core.models
32.  *
33.  * In order test with SlingContext, in parent/pom.xml add the dependency:
34.             <dependency>
35.                 <groupId>org.apache.sling</groupId>
36.                 <artifactId>org.apache.sling.testing.sling-mock.junit5</artifactId>
37.                 <version>2.3.4</version>
38.                 <scope>test</scope>
39.             </dependency>
40.  *
41.  * And in the core/pom.xml add:
42.         <dependency>
43.             <groupId>org.apache.sling</groupId>
44.             <artifactId>org.apache.sling.testing.sling-mock.junit5</artifactId>
45.         </dependency>
46.  *
47.  */
48. @ExtendWith(SlingContextExtension.class)
49. public class StockplexSlingMockTest {
50.
51.     private SlingContext context = new SlingContext();
52.
53.     private String noDataString = "No stock data imported for this stock symbol";
```

```
54.
55.     @BeforeEach
56.     void setup() throws Exception {
57.         //be required to register the models of this project manually depending on your project/IDE setup
58.         context.addModelsForClasses(Stockplex.class);
59.         //Load dummy data for imported stock data
60.         context.load().json("/imported-stock-data.json","/content/stocks/");
61.     }
62.
63.     /**
64.      * Loads a page into the context via a JSON file and then tests the stock model can successfully return values
65.      */
66.     @Test
67.     void testLoadedContent() throws Exception{
68.         //Load a resource from JSON
69.         Resource pageResource = context.load().json("/testpage-content.json", "/content/training/us/en/testpage");
70.         Resource stockplexResource = pageResource.getChild("jcr:content/root/responsivegrid/stockplex");
71.         context.request().setResource(stockplexResource);
72.
73.         Stockplex model = context.getService(ModelFactory.class).createModel(context.request(), Stockplex.class);
74.
75.         assertEquals("ADBE", model.getSymbol());
76.         assertEquals("Adobe's stock price today", model.getSummary());
77.         assertNotEquals("MSFT", model.getSymbol());
78.     }
79.
80.     /**
81.      * Load a resource into the context with the create().resource method.
82.      */
83.     @Test
84.     void testCreatedContent() {
85.         //Create a resource programmatically
86.         Resource stockplexResource = context.create().resource("/content/mypage/jcr:content/root/responsivegrid/
    stockplex", ImmutableMap.<String,Object>builder()
87.                 .put("jcr:primaryType", "nt:unstructured")
88.                 .put("jcr:createdBy", "admin")
89.                 .put("summary", "Adobe's stock price today")
90.                 .put("jcr:lastModifiedBy", "admin")
91.                 .put("symbol", "ADBE")
92.                 .put("jcr:created", "Thu Mar 21 2019 12:23:17 GMT-0400")
93.                 .put("showStockDetails", "true")
94.                 .put("jcr:lastModified", "Thu Mar 21 2019 12:23:34 GMT-0400")
95.                 .put("sling:resourceType", "training/components/content/stockplex")
96.                 .build());
97.         context.request().setResource(stockplexResource);
98.
99.         Stockplex model = context.getService(ModelFactory.class).createModel(context.request(), Stockplex.class);
100.
101.        assertEquals("ADBE", model.getSymbol());
102.        assertNotEquals("", model.getSummary());
103.        assertNotEquals("MSFT",model.getSymbol());
104.    }
105.
106.    /**
107.     * In this test we create a resource that uses the stockplex model. This model then looks for imported stock data
108.     * under /content/stocks/ADBE. Since ADBE stock was loaded, the expected outcome is the price being returned.
109.     */
110.    @Test
111.    void testImportedStockData() {
112.        String stockSymbol = "ADBE";
113.        Resource stockplexResource = context.create().resource("/content/mypage/jcr:content/root/responsivegrid/
    stockplex", ImmutableMap.<String,Object>builder()
114.                .put("jcr:primaryType", "nt:unstructured")
115.                .put("symbol", stockSymbol)
116.                .put("sling:resourceType", "training/components/content/stockplex")
117.                .build());
118.        context.request().setResource(stockplexResource);
119.        Stockplex model = context.getService(ModelFactory.class).createModel(context.request(), Stockplex.class);
120.
121.        assertNotEquals(0, model.getCurrentPrice()); //Validate there is stockprice
122.
123.        Map<String, Object> data = ImmutableMap.<String,Object>builder().put(stockSymbol,noDataString).build();
124.        assertNotEquals(data, model.getData()); //validate the data returned is stock the stock data
125.    }
126.
127.    /**
```

```
128.      * In this test we create a resource that uses the stockplex model. This model then looks for imported stock data
129.      * under /content/stocks/
APPL. Since APPL stock was not loaded, the expected outcome is 0 and data to be the ‹noDataString›
130.      */
131.     @Test
132.     void testNoneExistantStockData() {
133.         String stockSymbol = "APPL";
134.         Resource stockplexResource = context.create().resource("/content/mypage/jcr:content/root/responsivegrid/
stockplex", ImmutableMap.<String,Object>builder()
135.                 .put("jcr:primaryType", "nt:unstructured")
136.                 .put("symbol", stockSymbol)
137.                 .put("sling:resourceType", "training/components/content/stockplex")
138.                 .build());
139.         context.request().setResource(stockplexResource);
140.         Stockplex model = context.getService(ModelFactory.class).createModel(context.request(), Stockplex.class);
141.
142.         assertEquals(Double.valueOf("0"), model.getCurrentPrice());
143.
144.         Map<String, Object> data = ImmutableMap.<String,Object>builder().put(stockSymbol,noDataString).build();
145.         assertEquals(data, model.getData());
146.     }
147. }
```

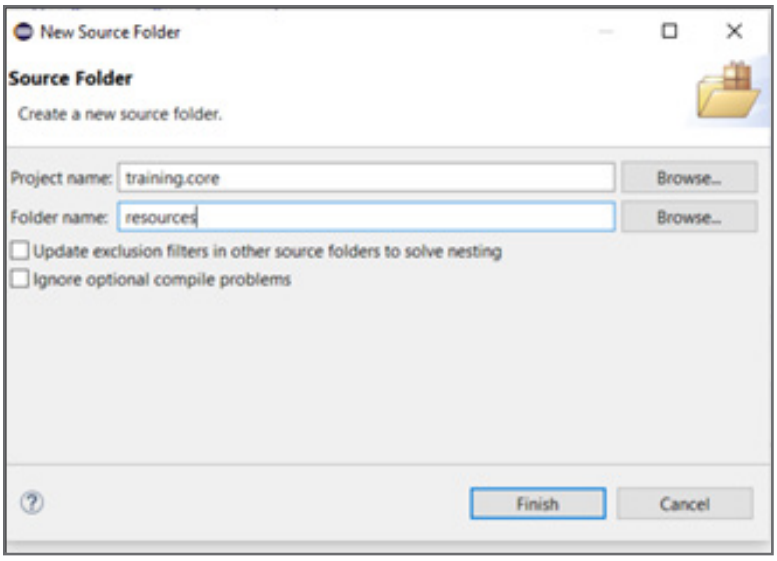
5. Examine the code. Observe how the mock test loads a json file named imported-stock-data.json. This code is used to simulate stock data already imported to the JCR.

The **StockplexSlingMock.java** essentially has two types of page setups:
1. The page resource is imported via json (testpage-content.json): a.testLoadedContent()
2. The page resource is manually created using the following methods:
a. testCreatedContent()
b. testImportedStockData() - This method tests that the stockplex component is pulling the actual stock data loaded
c. testNoneExistantStockData() - This method tests that the stockplex component is displaying a 'no stock data' message appropriately

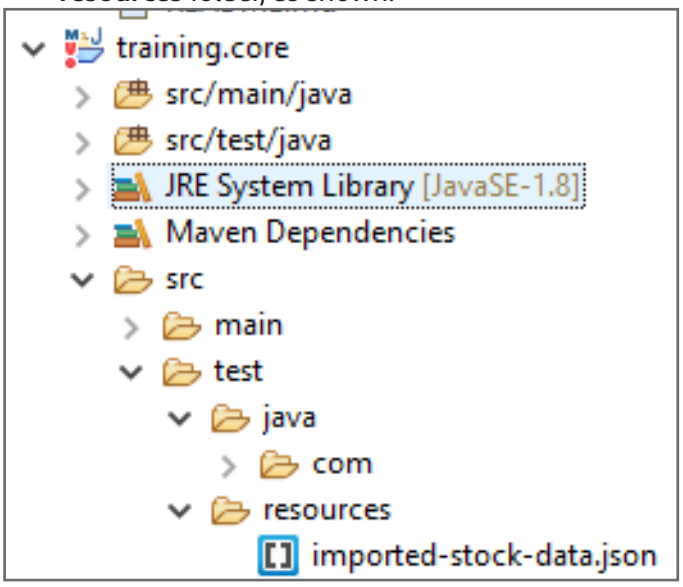
---

6. In **Project Explorer**, navigate to **training.core** > **src/test/java.**
7. Right-click **src/test/java** and select **New** > **Source Folder**. The **New Source Folder** screen opens.

8. Provide the value to create the folder, as shown:
   - Folder name: **resources**



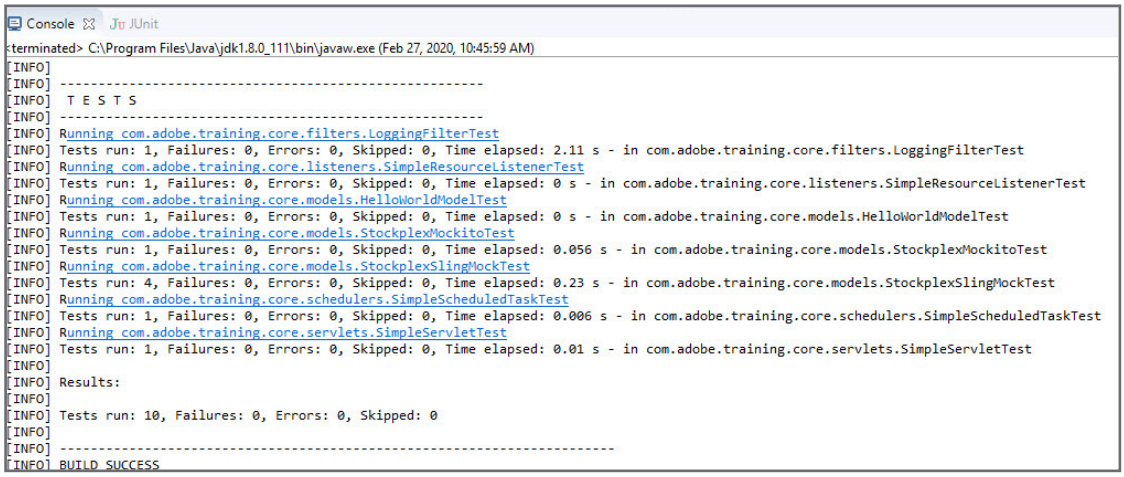
9. Click **Finish**. The folder is created.
10. Copy the file **imported-stock-data.json** from **/Exercise_Files/core/src/test/resources/.**
11. Right-click **resources** and select **Paste.** The **imported-stock-data.json** is copied to the **resources** folder, as shown:



12. Similarly, copy the file **testpage-content.json** from **/Exercise_Files/core/src/test/resources/.**
13. Right-click **resources** and select **Paste**.

Task 3: Run the test

1. To run all the tests within the bundle, right-click **training.core** and select **Run As** > **Maven test**.
2. Verify the tests ran successfully, as shown:



AEM Mocks

AEM Mocks are inclusive of JCR, OSGi, and Sling Mocks. It also adds extra functionality on top for pages, assets, and other AEM objects.
Usage:
The AemContext object provides access to mock implementations of:

- OSGi Component Context
- OSGi Bundle Context
- Sling Resource Resolver
- Sling Request
- Sling Response
- Sling Script Helper

Additionally, it supports:

- Registering OSGi services
- Registering adapter factories
- Accessing JSON Importer

The AEM mock context can be injected into a JUnit test using a custom JUnit rule named AemContext. This rule takes care of all initialization and cleanup tasks required to make sure all unit tests can run independently. Example:

```
public class ExampleTest {

  @Rule
  public final AemContext context = new AemContext();

  @Test
  public void testSomething() {
    Resource resource = context.resourceResolver().getResource("/content/sample/en");
    Page page = resource.adaptTo(Page.class);
    // further testing
  }

}
```

It is possible to combine such a unit test with an @RunWith annotation. The AEM mock context supports different resource resolver types (provided by the Sling Mocks implementation).

Exercise 3: Create unit tests using AEM Mocks

In this lab exercise, you will create unit tests using AEM mocking framework.
This exercise includes three tasks:

1. Add the AEM-mock dependency
2. Create a unit test with sample data

Task 1: Add 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. Add the dependency **com.fasterxml.jackson.core**, as shown:

```
694             <!-- Dependency for PageCreatorAEMMockTest.java -->
695             <dependency>
696                 <groupId>com.fasterxml.jackson.core</groupId>
697                 <artifactId>jackson-core</artifactId>
698                 <version>2.9.5</version>
699                 <scope>test</scope>
700             </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.
The code is available as part of the parent-pom.xml in **Exercise_Files** under **/training-files/Writing-Tests/**.

4. In the **Exercise_Files**, navigate to **/training-files/Writing-Tests/.** Open **core-pom.xml**.
5. Copy the **jackson-core** dependency at the bottom:

```
    <!-- Dependency for PageCreatorAEMMockTest.java -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
    </dependency>
```

6. In **Project Explorer**, navigate to **training** > **core** > **pom.xml**.
7. Double-click **pom.xml** to open it in the editor. The **pom.xml** file opens.
8. Add the **com.fasterxml.jackson.core** dependency, as shown:

```
199         <!-- Dependency for StockplexSlingMockTest.java -->
200         <dependency>
201             <groupId>org.apache.sling</groupId>
202             <artifactId>org.apache.sling.testing.sling-mock.junit5</artifactId>
203         </dependency>
204         <!-- Dependency for PageCreatorAEMMockTest.java -->
205         <dependency>
206             <groupId>com.fasterxml.jackson.core</groupId>
207             <artifactId>jackson-core</artifactId>
208         </dependency>
```

9. Save the changes.

Task 2: Create a unit test with sample data

1. In **Project Explorer**, navigate to **training.core** > **src/test/java**.
2. Copy the file **PageCreatorAEMMockTest.java** from **Exercise_Files** under **core/src/test/java/com/adobe/training/core/servlets/**.

NOTE: The code is provided as part of Exercise_Files under **/core/src/test/java/com/adobe/training/core/servlets/**. Copy and paste the file from the exercise files referenced to **PageCreatorAEMMockTest.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. In Eclipse, right-click **com.adobe.training.core.models** and paste the file. The **PageCreatorAEMMockTest.java** class is created.

Note: You will now test the PageCreator.java class.

4. The **PageCreatorAEMMockTest.java** file opens in the editor, as shown:

```
1.  package com.adobe.training.core.servlets;
2.
3.  import com.adobe.training.core.servlets.PageCreator;
4.  import com.day.cq.tagging.TagManager;
5.  import com.fasterxml.jackson.databind.JsonNode;
6.  import com.fasterxml.jackson.databind.ObjectMapper;
7.  import com.google.common.collect.ImmutableMap;
8.  import io.wcm.testing.mock.aem.junit5.AemContext;
9.  import io.wcm.testing.mock.aem.junit5.AemContextExtension;
10. import org.junit.jupiter.api.BeforeEach;
11. import org.junit.jupiter.api.Test;
12. import org.junit.jupiter.api.extension.ExtendWith;
13.
14. import javax.servlet.http.HttpServletResponse;
15.
16. import static org.junit.jupiter.api.Assertions.assertEquals;
17. import static org.junit.jupiter.api.Assertions.assertNotNull;
18.
19. /**
20.  * This class uses the AemContext object from AEM mocks to test the PageCreator Model.
21.  *
22.  * In order to test with JSON objects, in parent/pom.xml add the dependency:
23.          <dependency>
24.             <groupId>com.fasterxml.jackson.core</groupId>
25.             <artifactId>jackson-core</artifactId>
26.             <version>2.9.5</version>
27.             <scope>test</scope>
28.          </dependency>
29.  *
30.  * And in the core/pom.xml add:
31.       <dependency>
32.          <groupId>com.fasterxml.jackson.core</groupId>
33.          <artifactId>jackson-core</artifactId>
34.       </dependency>
35.  *
36.  */
37. @ExtendWith(AemContextExtension.class)
38. public class PageCreatorAEMMockTest {
39.
40.     private AemContext context = new AemContext();
41.
42.     private PageCreator pageCreator;
43.
```

```
44.     private static String DEFAULT_TEMPLATE = "/conf/training/settings/wcm/templates/content-page-template";
45.     private static String PARENT_PAGE = "/content/training/us/en";
46.
47.     /**
48.      * The setUp method loads the AemContext with services and resources to be used in the tests below.
49.      */
50.     @BeforeEach
51.     void setUp() throws Exception {
52.         pageCreator = context.registerService(new PageCreator());
53.
54.         context.load().json("/training-conf.json", "/conf/training");
55.
56.         context.create().page(PARENT_PAGE,DEFAULT_TEMPLATE,"English");
57.         TagManager tm = context.resourceResolver().adaptTo(TagManager.class);
58.         try {
59.             tm.createTag("/content/cq:tags/training/biking", "Biking", "");
60.         } catch (Exception e) {
61.             e.printStackTrace();
62.         }
63.     }
64.
65.     /**
66.      * This test creates a POST request for the PageCreator class to validate
67.      * a correct input for page creation
68.      */
69.     @Test
70.     void testSuccessfulInput() throws Exception{
71.         //Simulate a POST request
72.         context.request().setMethod("POST");
73.         context.request().setParameterMap(ImmutableMap.of(
74.  biking")  "importer",PARENT_PAGE+"/community,Our Community,"+DEFAULT_TEMPLATE+",/content/cq:tags/training/
75.                 );
76.
77.         //Call the Servlet
78.         pageCreator.service(context.request(), context.response());
79.         assertEquals(HttpServletResponse.SC_OK, context.response().getStatus());
80.
81.         String jsonString = context.response().getOutputAsString();
82.         ObjectMapper mapper = new ObjectMapper();
83.         JsonNode actualObj = mapper.readTree(jsonString);
84.         assertEquals("Successful", actualObj.findValue("Status").asText());
85.     }
86.
87.     /**
88.      * This test creates a POST request for the PageCreator class to validate
89.      * if an error message is returned if there is no parent path given
90.      */
91.     @Test
92.     void testNoParentPath() throws Exception{
93.         //Simulate a POST request
94.         context.request().setMethod("POST");
95.         context.request().setParameterMap(ImmutableMap.of(
96.                 "importer","Our Community,"+DEFAULT_TEMPLATE+",/content/cq:tags/training/biking")
97.                 );
98.
99.         //Call the Servlet
100.        pageCreator.service(context.request(), context.response());
101.        assertEquals(HttpServletResponse.SC_OK, context.response().getStatus());
102.
103.        String jsonString = context.response().getOutputAsString();
104.        ObjectMapper mapper = new ObjectMapper();
105.        JsonNode actualObj = mapper.readTree(jsonString);
106.        assertEquals("Error", actualObj.findValue("Status").asText());
107.    }
108.
109.    /**
110.     * This test creates a POST request for the PageCreator class to validate
111.     * if the default template is used when none is given
112.     */
113.    @Test
114.    void testNoTemplate() throws Exception{
115.        //Simulate a POST request
116.        context.request().setMethod("POST");
117.        context.request().setParameterMap(ImmutableMap.of(
118.                "importer",PARENT_PAGE+"/community,Our Community,,/content/cq:tags/training/biking")
119.                );
120.
121.        //Call the Servlet
122.        pageCreator.service(context.request(), context.response());
123.        assertEquals(HttpServletResponse.SC_OK, context.response().getStatus());
```

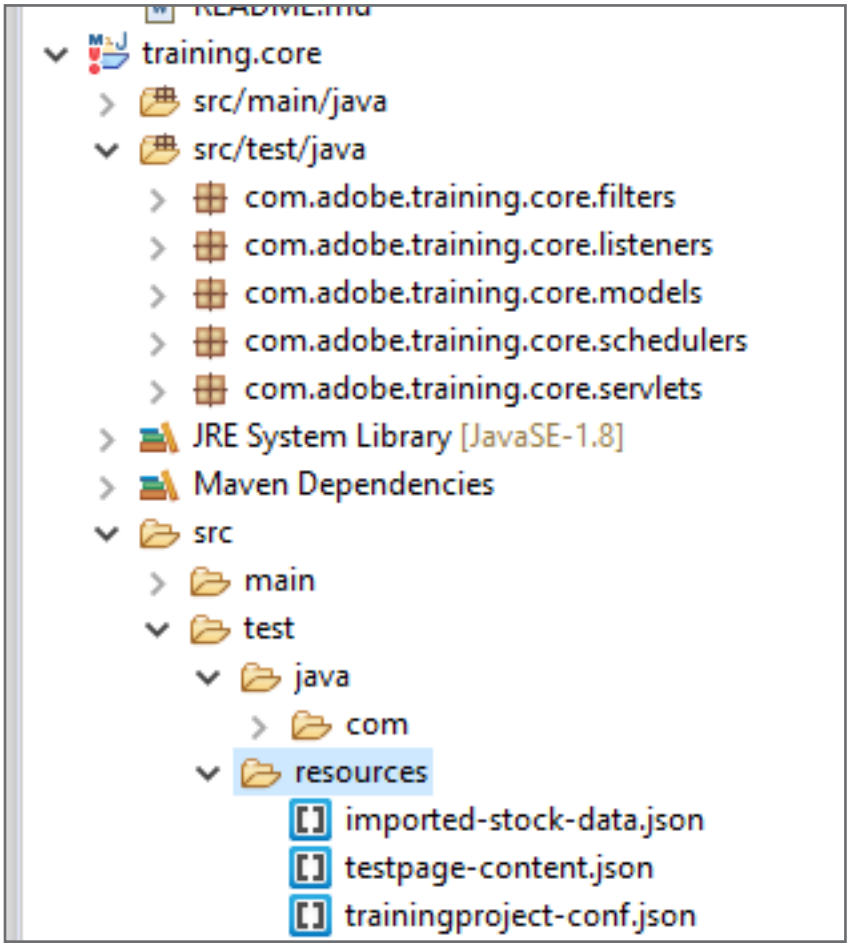
```
124.
125.        String jsonString = context.response().getOutputAsString();
126.        ObjectMapper mapper = new ObjectMapper();
127.        JsonNode actualObj = mapper.readTree(jsonString);
128.        //Assert the default template is used
129.        assertEquals(DEFAULT_TEMPLATE, actualObj.findValue("Template Used").asText());
130.    }
131.
132.    /**
133.     * This test creates a POST request for the PageCreator class to validate
134.     * if an error message is returned because the template is not a valid template in the project.
135.     */
136.    @Test
137.    void testDNETemplate() throws Exception{
138.        //Simulate a POST request
139.        context.request().setMethod("POST");
140.        context.request().setParameterMap(ImmutableMap.of(
141.                "importer",PARENT_PAGE+"/community,Our Community,/conf/training/settings/wcm/templates/
    mytemplate,/content/cq:tags/training/biking,FALSE")
142.                );
143.
144.        //Call the Servlet
145.        pageCreator.service(context.request(), context.response());
146.        assertEquals(HttpServletResponse.SC_OK, context.response().getStatus());
147.
148.        String jsonString = context.response().getOutputAsString();
149.        ObjectMapper mapper = new ObjectMapper();
150.        JsonNode actualObj = mapper.readTree(jsonString);
151.
152.        //Assert that the status is Error and a Template message is given
153.        assertEquals("Error", actualObj.findValue("Status").asText());
154.        assertNotNull(actualObj.findValue("Template"));
155.    }
156. }
```

Note: The PageCreatorAEMMockTest.java class has a few tests. The testSuccessfulInput() tests a valid input for PageCreator. The other three methods testNoParentPath(),testNoTemplate() and testDNETemplate() test error-prone inputs.

5. Examine the code. Observe how the mock test loads a json file named /**trainingproject-conf.json.**

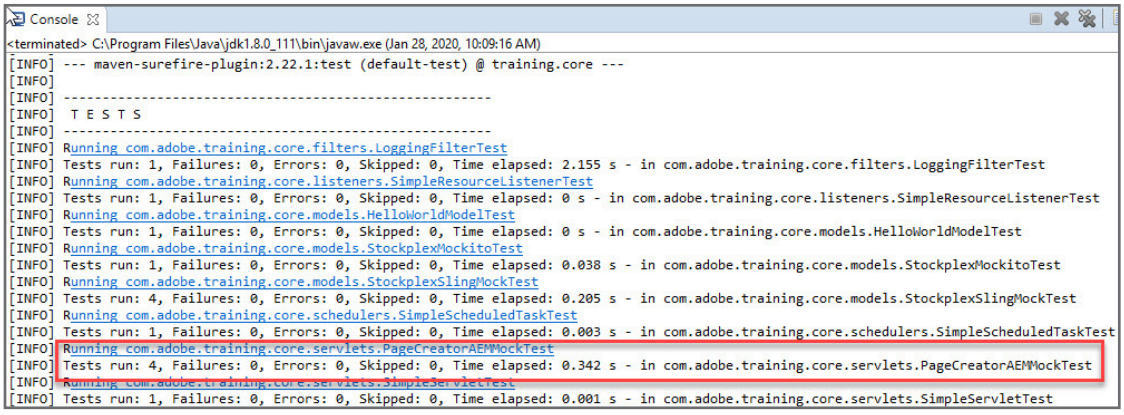
6. Copy the file **training-conf.json** from **/Exercise_Files/core/src/test/resources/.**

7. Right-click **resources** and select Paste:



8. To run all the tests within the bundle, right-click **training.core** and select **Run As** > **Maven test**.

9. Verify the tests ran successfully, as shown:



References

Sling Mocks: https://sling.apache.org/documentation/development/sling-mock.html

AEM Mocks: http://wcm.io/testing/aem-mock/usage.html

AEM Mocks in AEM projet Archetype: https://wcm-io.atlassian.net/wiki/spaces/WCMIO/pages/1046609921/How+to+use+AEM+Mocks+in+Adobe+AEM+Project+Archetype

wcm.io - Troubleshooting:
https://wcm-io.atlassian.net/wiki/spaces/WCMIO/pages/4816898/Troubleshooting

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

Appendix A: Creating Project and Workflow

Introduction

The AEM Projects feature helps manage and group all workflows and tasks associated with creating content in AEM Sites and Assets.

Objectives

After completing this course, you will be able to:

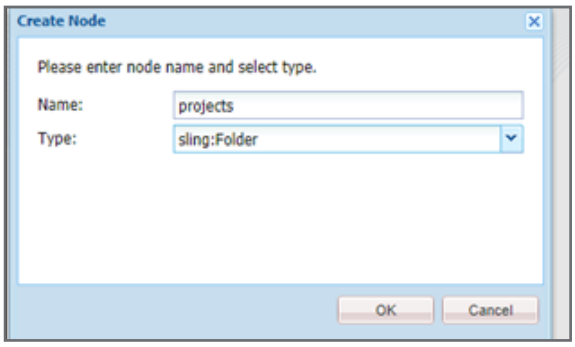
- Create a project template
- Execute a workflow
- Build and test the workflow

Exercise 1: Create a project template

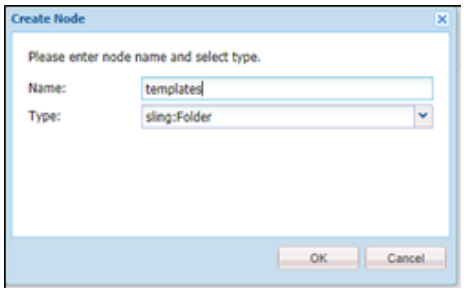
When creating custom business processes in AEM, a custom project might be required. In this exercise, you will create a project template, remove unnecessary tiles from the template, add a custom workflow, and add a new group called reviewers.

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.

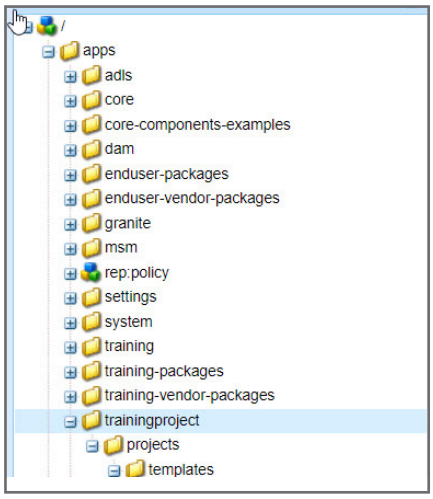
1. Ensure you are in the /apps folder, click **Create** from the actions bar, and select **Mixed Media Set** from the drop-down menu.
   - › Enter **projects** in the Name field.
   - › Select **sling:Folder** from the Type drop-down menu.



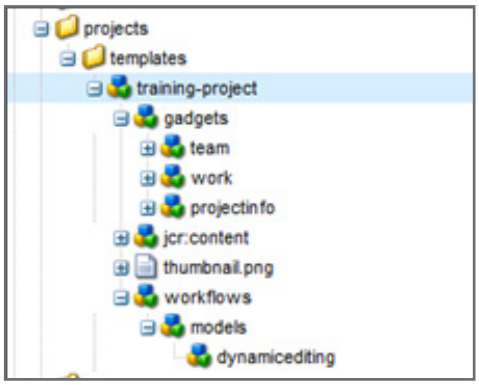
2. Click **OK**. The projects node is created.
3. Click **Save All.** Your changes are saved.
4. Right-click projects, and click **Create** > **Create node**. Enter the following values, as shown:
   - › Enter **templates** in the Name field.
   - › Select **sling:Folder** from the Type drop-down menu.



5. Click **OK**. The node is created.
6. Click **Save All**. Your changes are saved, as shown:

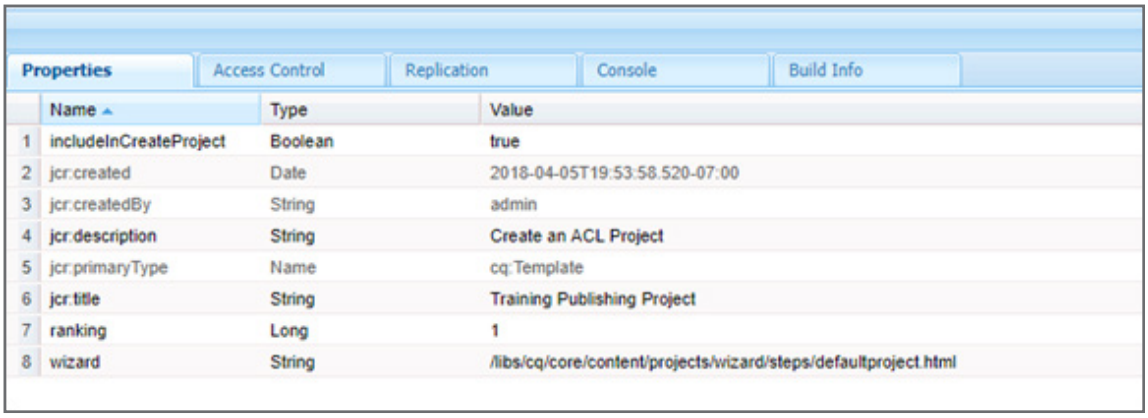


7. Copy **/libs/cq/core/content/projects/templates/default** to **/apps/trainingproject/projects/templates**.
8. Right-click **default**, and rename to **training-project,** as shown:



9. Click **Save All.** Your changes are saved.

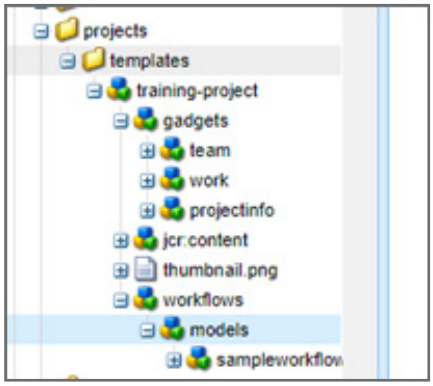
10. Edit the following properties of the node training-project with the following values, as shown:
    - jcr:title- Enter **Training Publishing Project** for the Value field.
    - jcr:description- Enter **Create an ACL Project** for the Value field.



	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

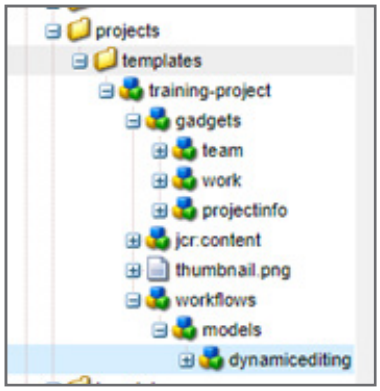
11. Click **Save All**. Your changes are saved.

12. Delete the following under training-project, :
    - **gadgets/experiences**
    - **gadgets/asset**
    - **Workflows/models/launch**
    - **Wokflow/models/landing**
    - **Workflow/models/email**

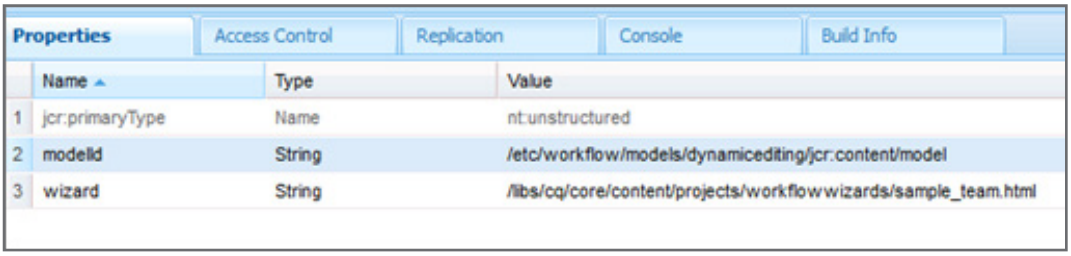


13. Click **Save All**. Your changes are saved.

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.

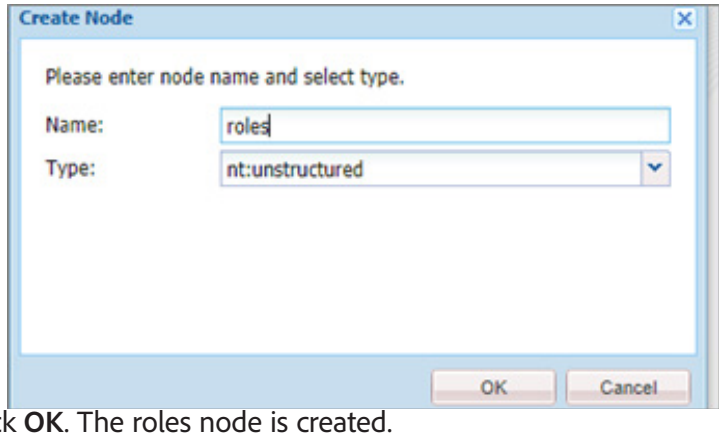


Note: This workflow named dynamicediting (the value you entered for the modelId in the current step) does not exist yet, you will create it later.

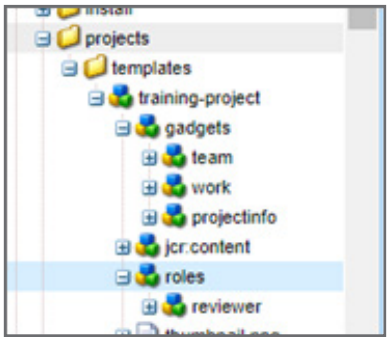
16. Click **Save All**. Your changes are saved.

17. Right-click **training-project** node, and click **Create** > **Create node**. The **Create Node** dialog box opens.

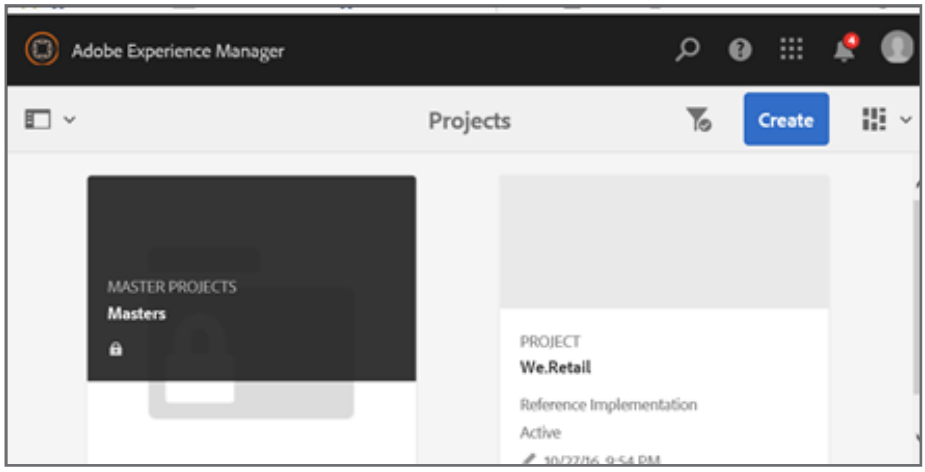
18. In the **Create Node** dialog box, enter the following values, as shown:
    - Name: **roles**
    - Select **nt:unstructured** from the Type dropdown.



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:

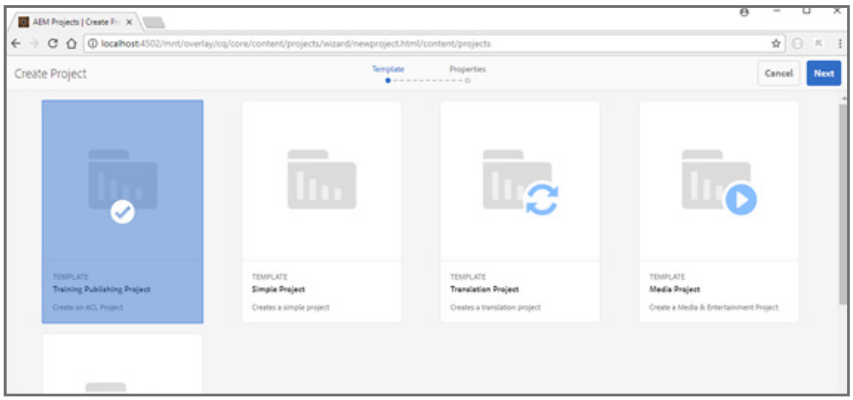


22. Click **Save All**. Your changes are saved.
23. Navigate to the **AEM** > **Projects**. The Projects Console opens, as shown:

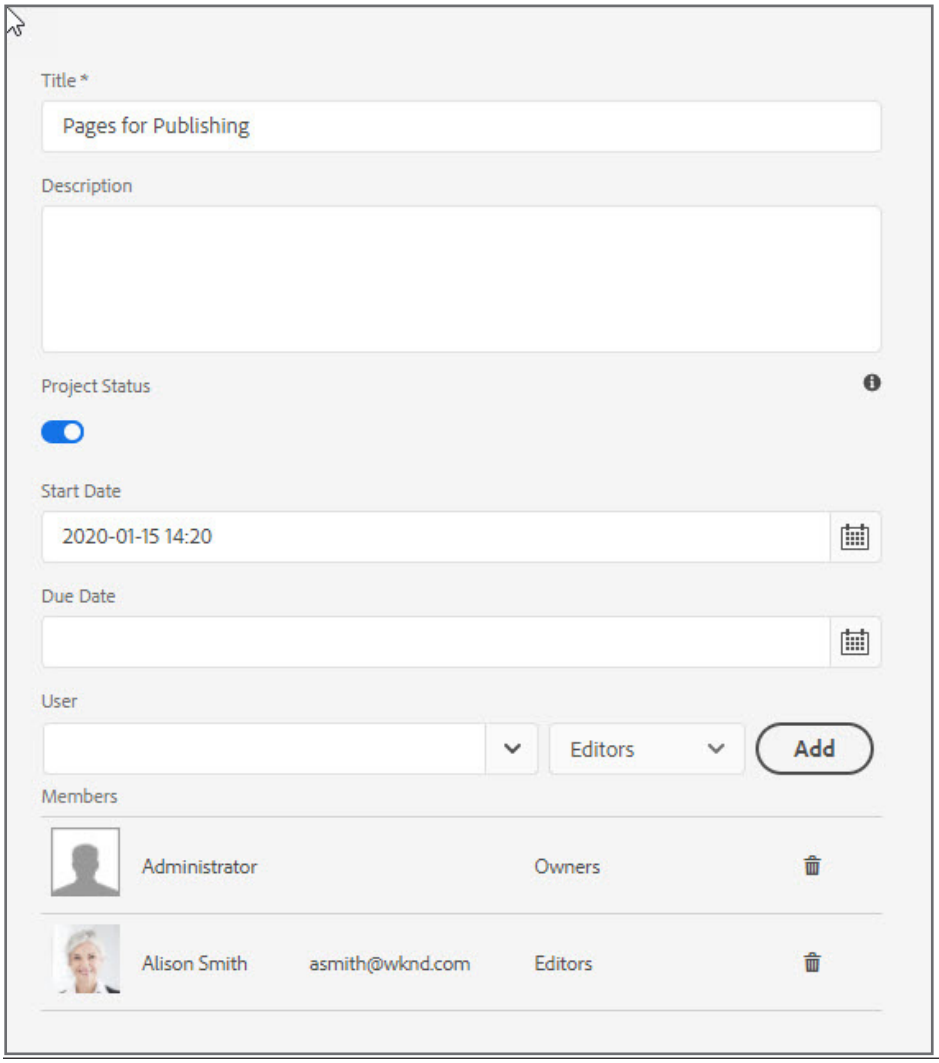


24. Select **Create** > **Project**. The template opens.

25. Select **Training Publishing Project**, and click **Next**. The **Create Project** window opens.

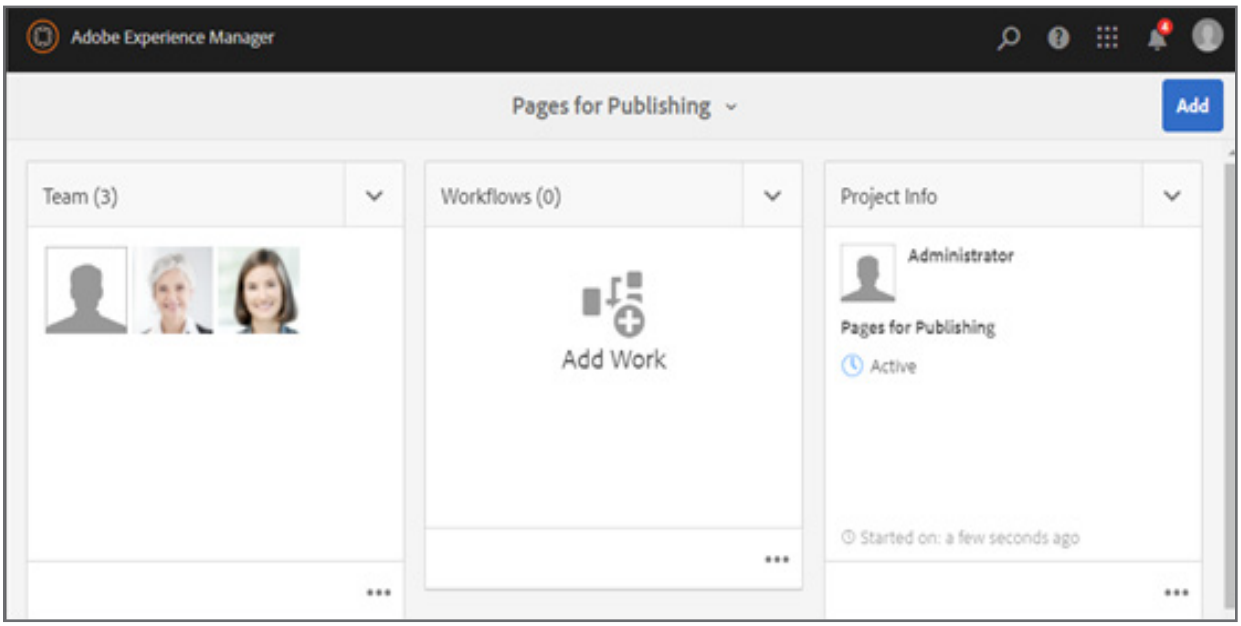


26. Provide these values for the project, as shown:
    - Type **Pages for Publishing** in the **Title** field.
    - Type **Alison Smith** in the **User** field and then select **Carlene Avery** from the drop-down menu. Ensure the drop-down menu beside the **User** field is set to **Editors**, and click **Add**.



27. Click **Create** in the top right corner. The **Success** pop-up window opens.

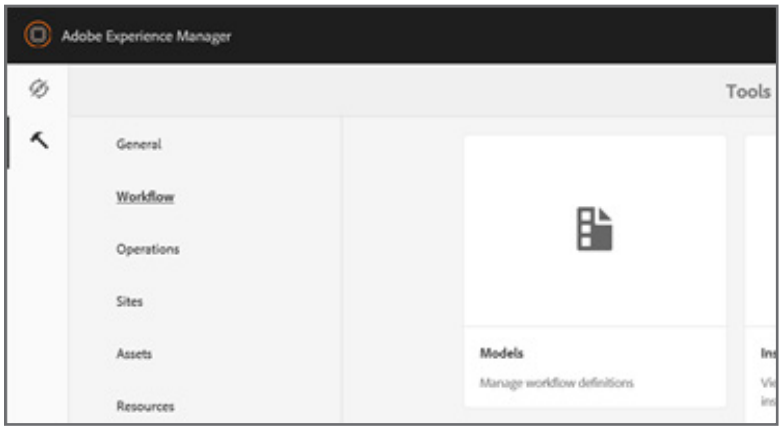
28. Click **Open** in the pop-up window. Observe the changes of the custom project, as shown.



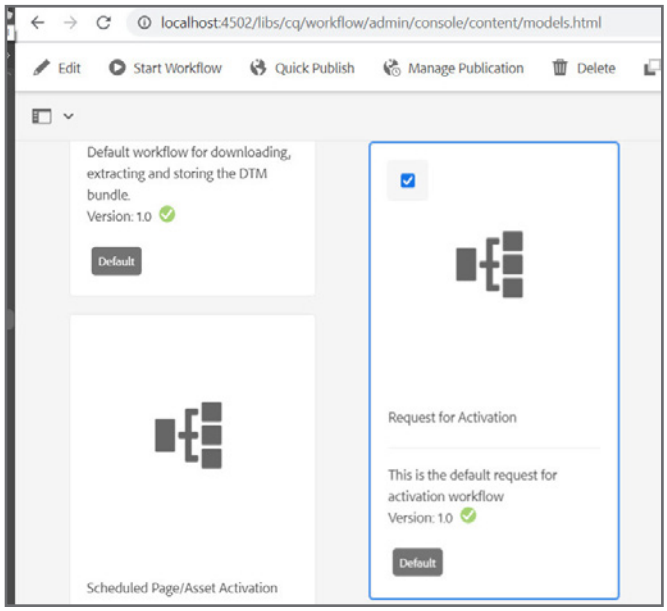
Exercise 2: Execute a workflow

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.

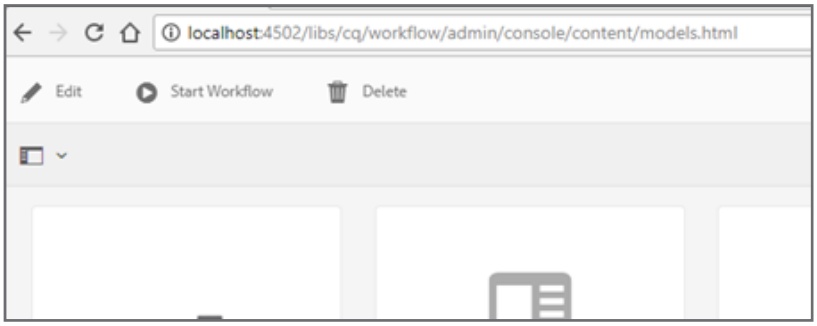
1. In AEM, navigate to **Tools** > **Workflow** > **Models**, as shown:



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.



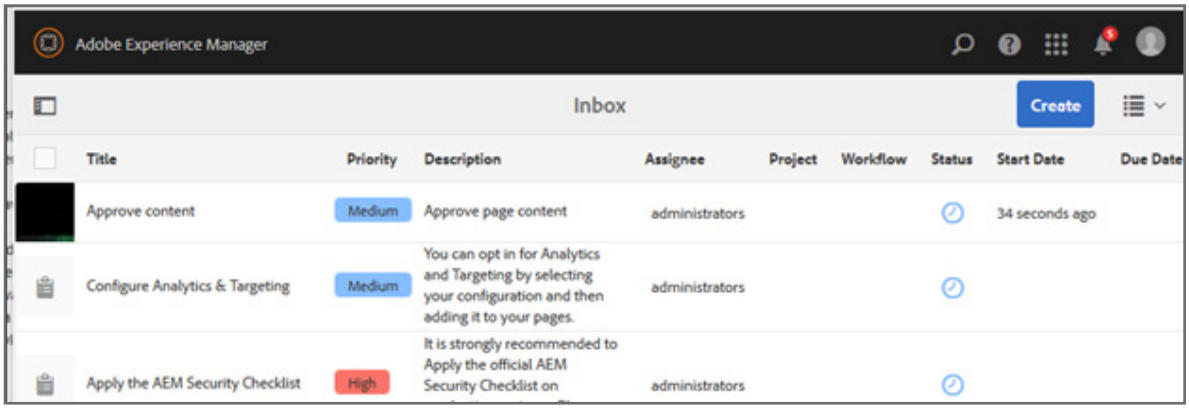
4. Click **Start Workflow** in the top left corner, as shown. The **Run Workflow** dialog box opens.



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. Frist, edit an initial version of the page. Next, publish the page. Each of these steps requires approval.

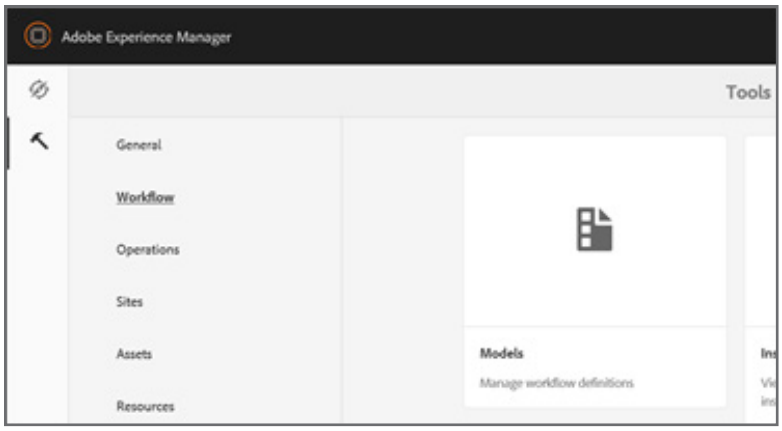
Exercise 3: Build and test a workflow

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:

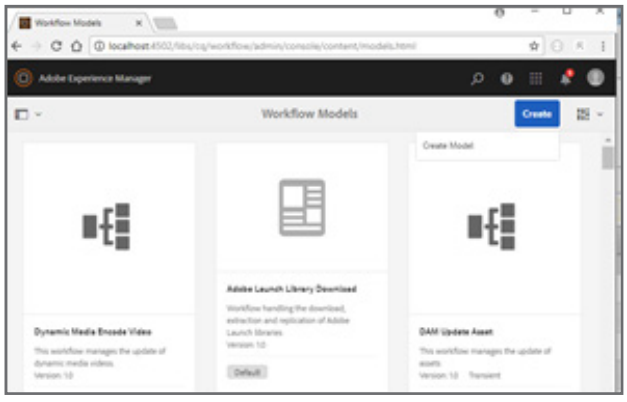
1. Build a workflow
2. Test the workflow

Task 1: Build a workflow

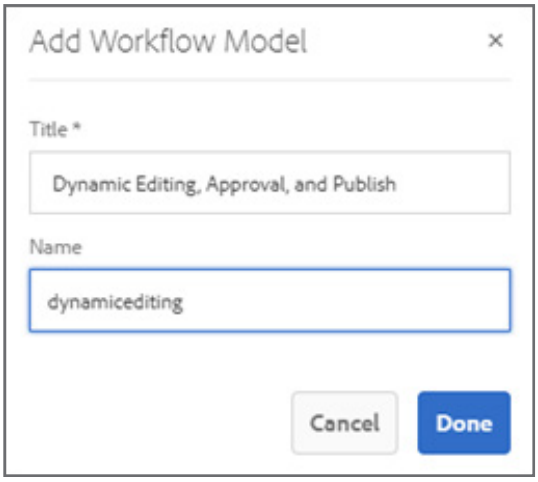
1. In AEM, navigate to **Tools** > **Workflow** > **Models**, as shown:



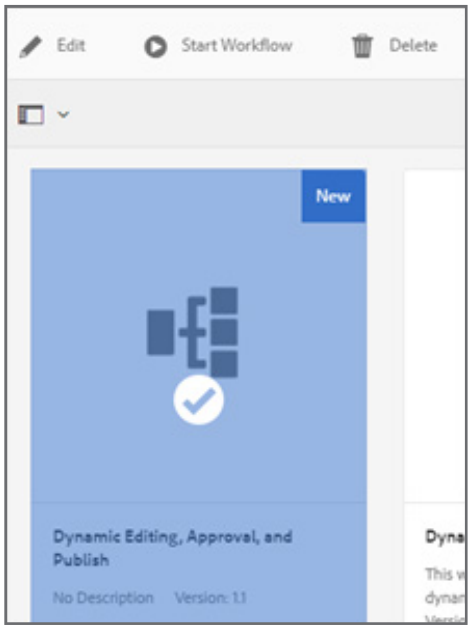
2. Select **Create** > **Create Model**, as shown:



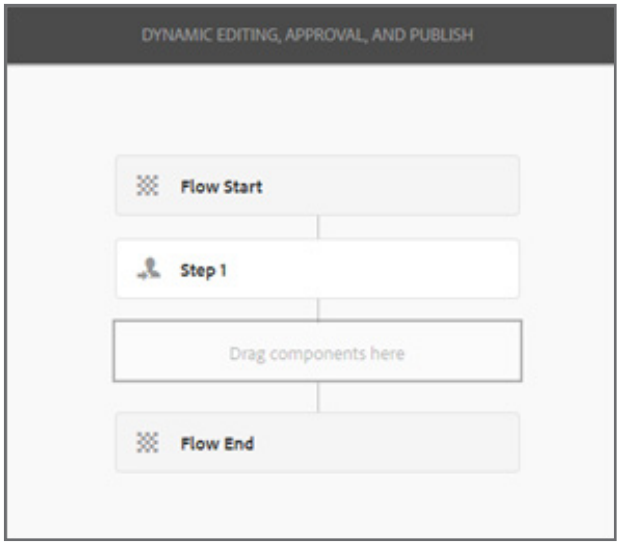
3. In the **Add Workflow Model** window, enter the values, as shown:
   - Title: **Dynamic Editing, Approval, and Publish**
   - Name: **dynamicediting**



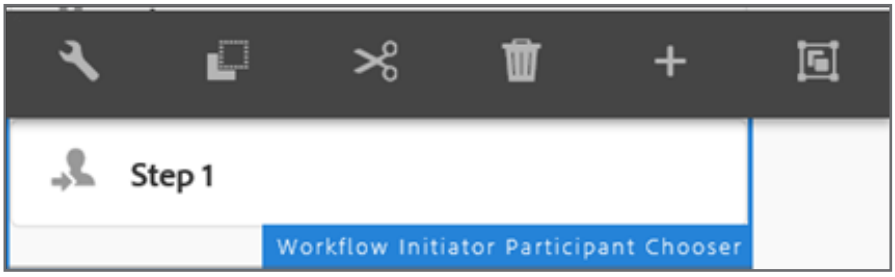
4. Click **Done**. The **Dynamic Editing, Approval, and Publish** model is created.
5. In the Workflow Models window, select the checkmark on the model **Dynamic Editing, Approval, and Publish** and click **Edit**, as shown:



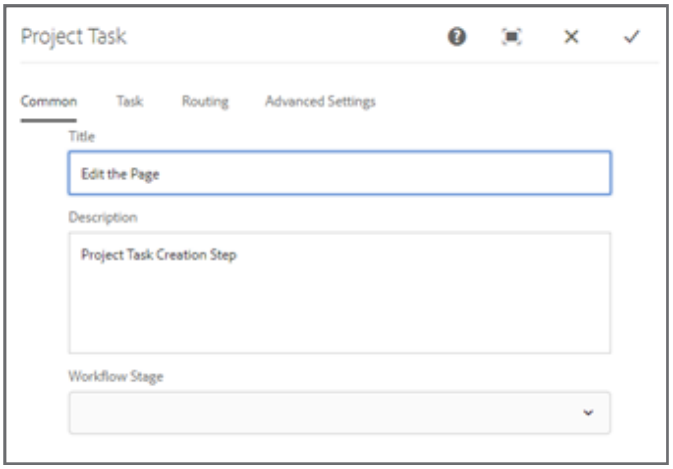
› The Workflow Model **Dynamic Editing, Approval, and Publish** opens for editing, as shown:



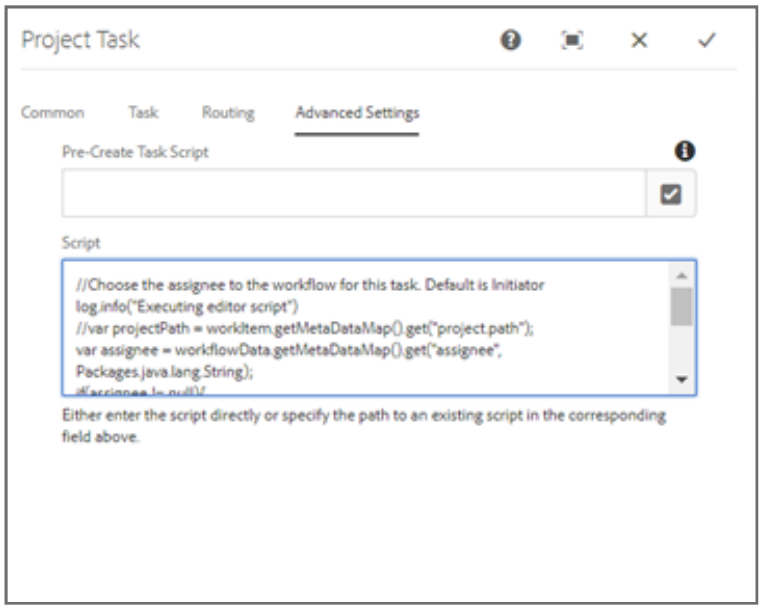
6. Delete the **Step 1** component by double-clicking it and selecting the Delete icon from the menu bar, as shown:



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:

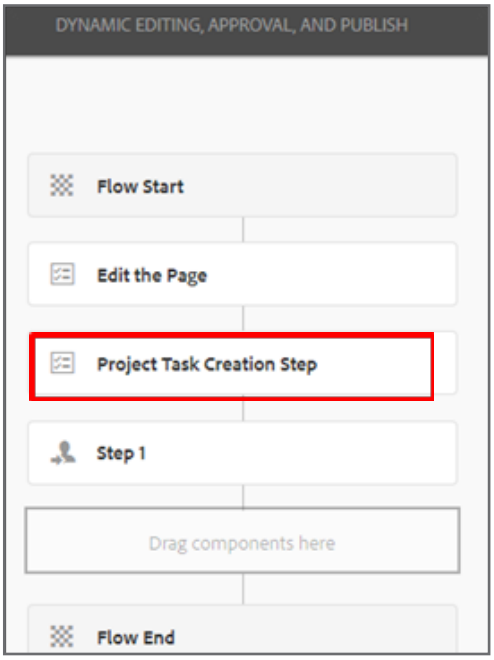


10. Click the **Task** tab and enter the following details:
    - Name: **Edit the Page**
    - Select the **Email** checkbox.
    - Description: **Make the appropriate edits to the page and complete this task**.
11. Select the **Advanced Settings** tab, and copy and paste the contents from **Exercise_Files/backend-training-dev/training-files/Deep_Dive_into_AEM_APIs/workflow-script.txt**, as shown:



12. Click the Done icon (checkmark) on the top of the dialog box to save the changes.

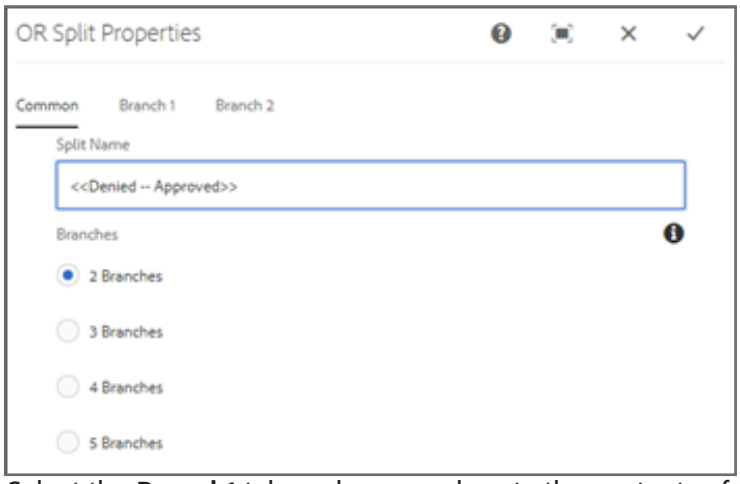
13. Drag and drop another Create **Project Task** component from the left-hand side below **Edit the Page** in the model, as shown:



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/ backend-training-dev/training-files/Deep_Dive_into_AEM_APIs/workflow-script.txt**.
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:

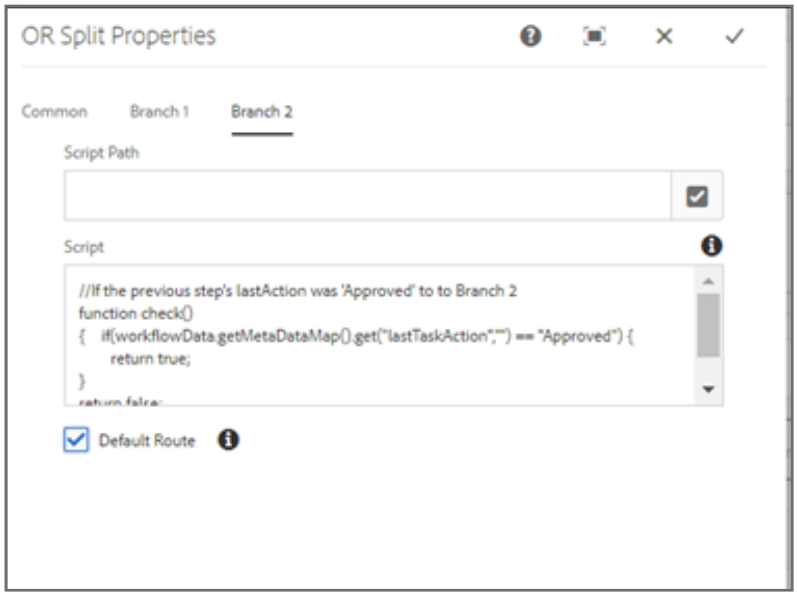
21. Double-click **OR Split**. The **OR Split** Properties dialog box opens.
22. Select the **Common** tab and enter the title as <<Denied—Approved>> and select the 2 Branches option button, as shown:



23. Select the **Branch1** tab, and copy and paste the contents of Or Split Branch 1 Scripts from **/ Exercise_Files/ backend-training-dev/training-files/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/ backend-training-dev/training-files/Deep_Dive_into_AEM_APIs/workflow-script.txt**, as shown.

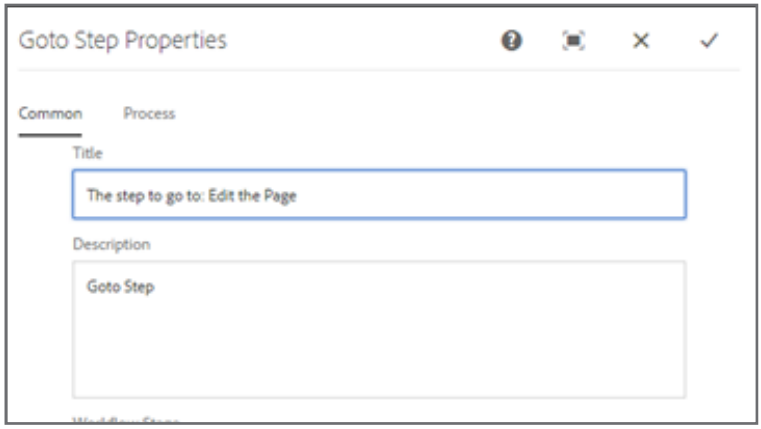
25. Select the Default Route checkbox.



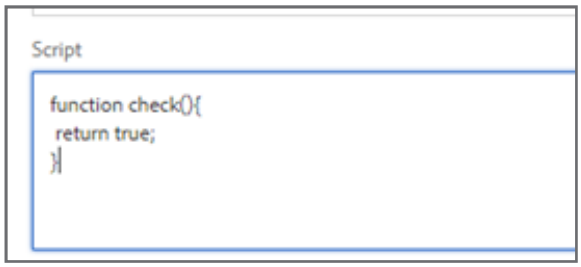
26. Click the Done icon (checkmark) on the top of the dialog box to save the changes.

27. In the left branch (Branch 1), drag the **Goto Step** component and Open the dialog box.

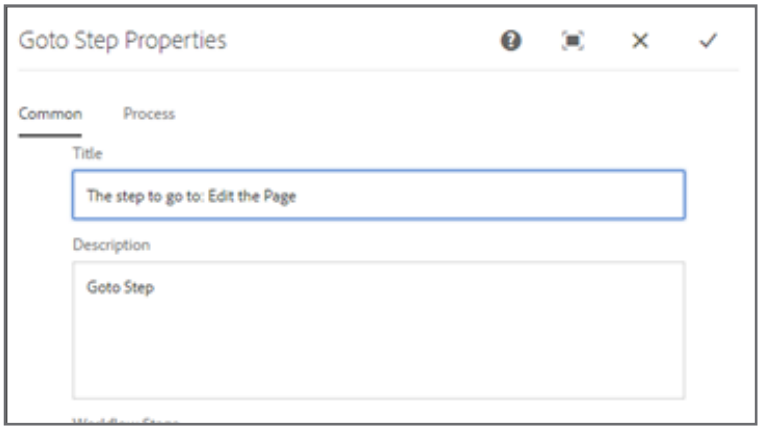
28. Select the **Process** tab, and enter the Title, as shown:



29. Script: Copy and Paste the "GOTO Script" from the workflows-script.txt, as shown:



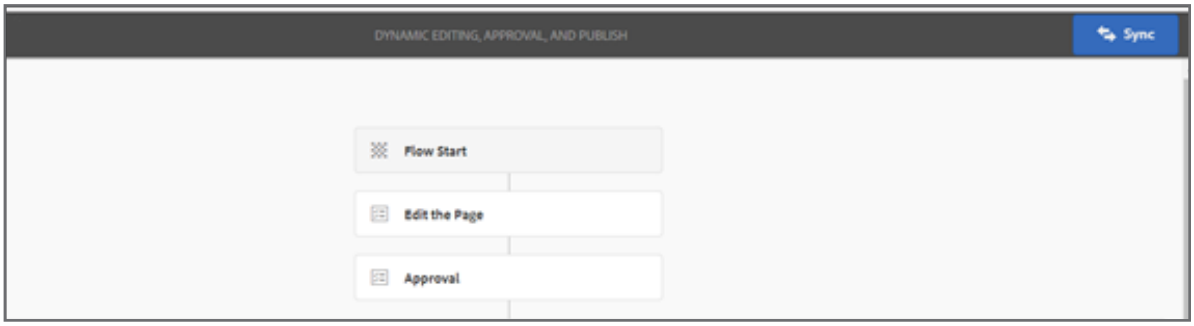
30. Select the **Common** tab, and enter the **title**, as shown:



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:

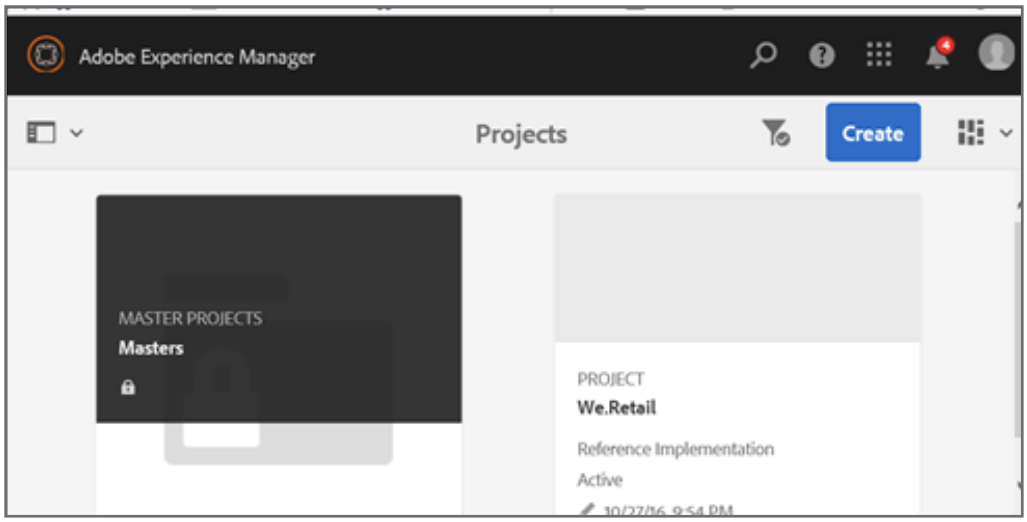
33. At this point the workflow is complete. In the top-right click **Sync**, as shown. This will write the workflow model.



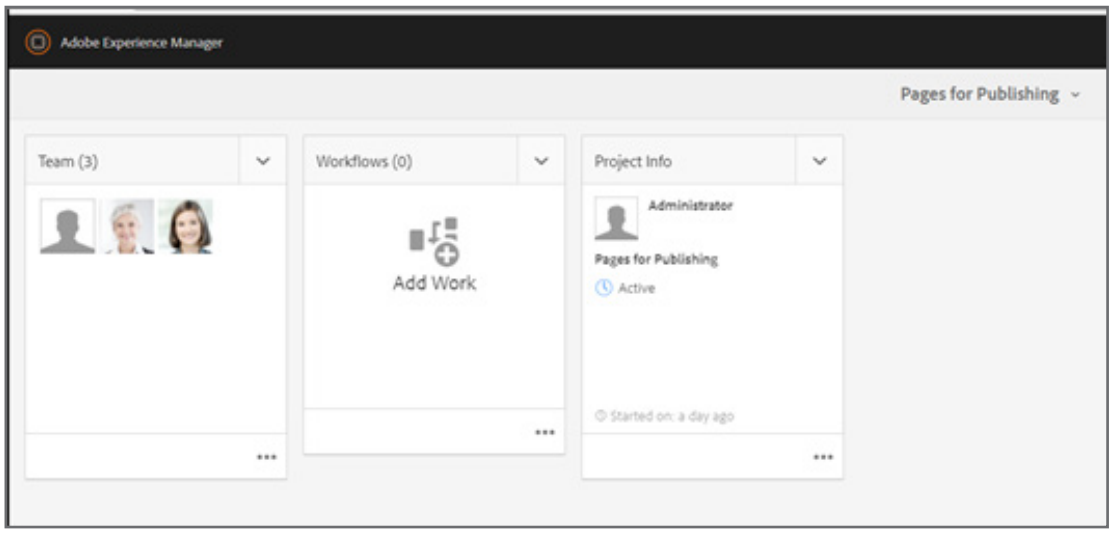
Task 2: Test the workflow

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.

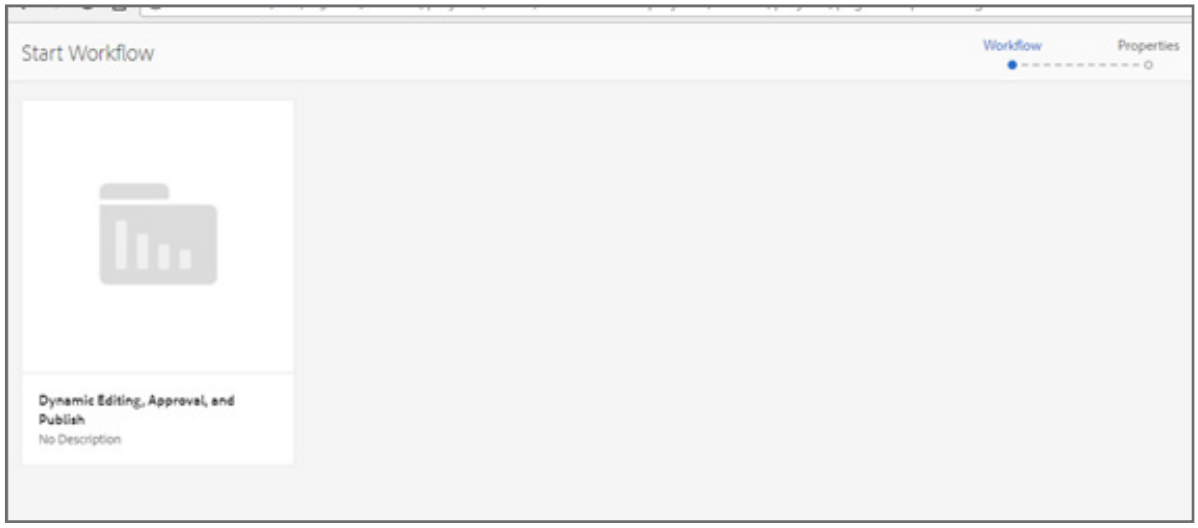
1. Navigate to **AEM > Projects.** The **Projects Console** opens, as shown:



2. Click **Pages for Publishing** Project, and click **Next**. The **Pages for Publishing** window opens, as shown:

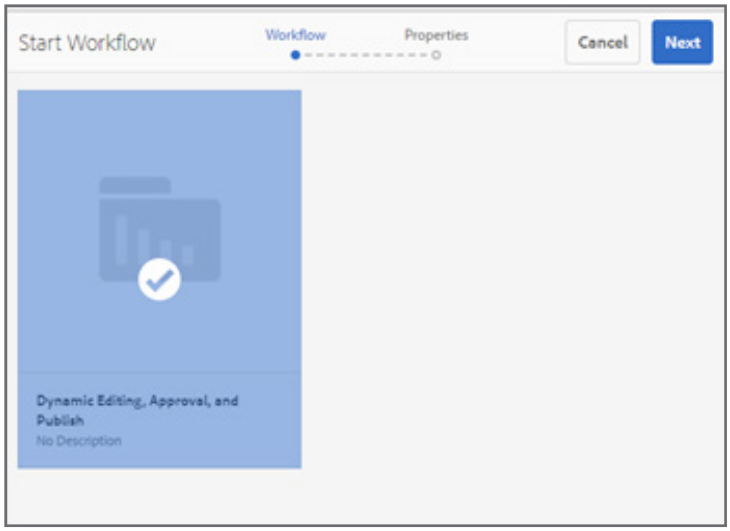


3. In the Workflow tile, click Add Work. The **Start Workflow** window opens, as shown:

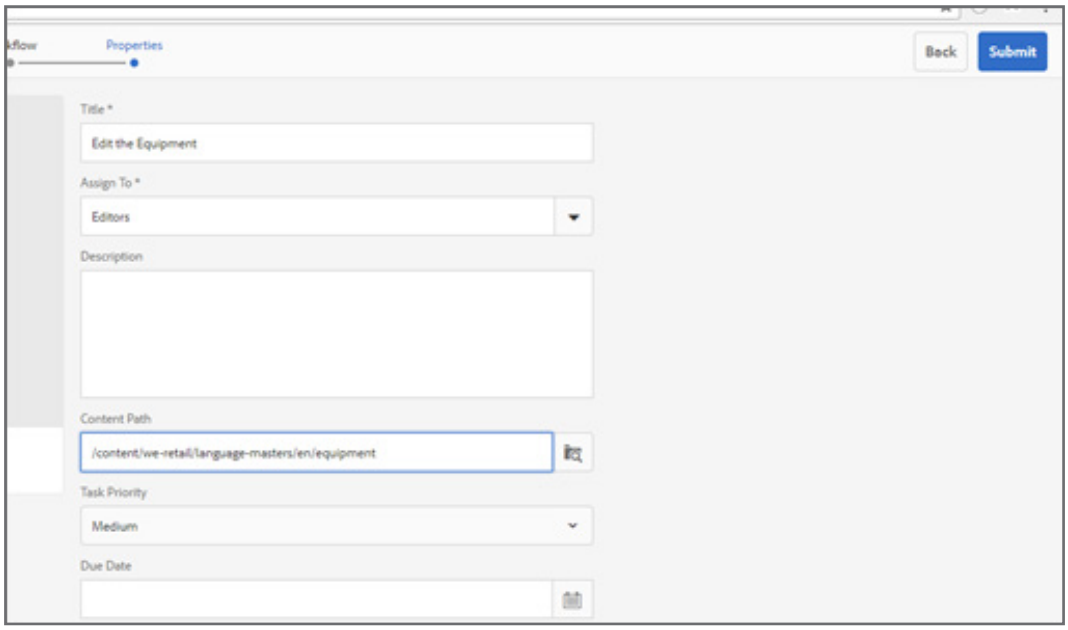


Note: If your workflow does not appear, double check the workflow added to the project template under /apps/trainingproject/projects/templates/training-project/workflows/models/dynamicediting.

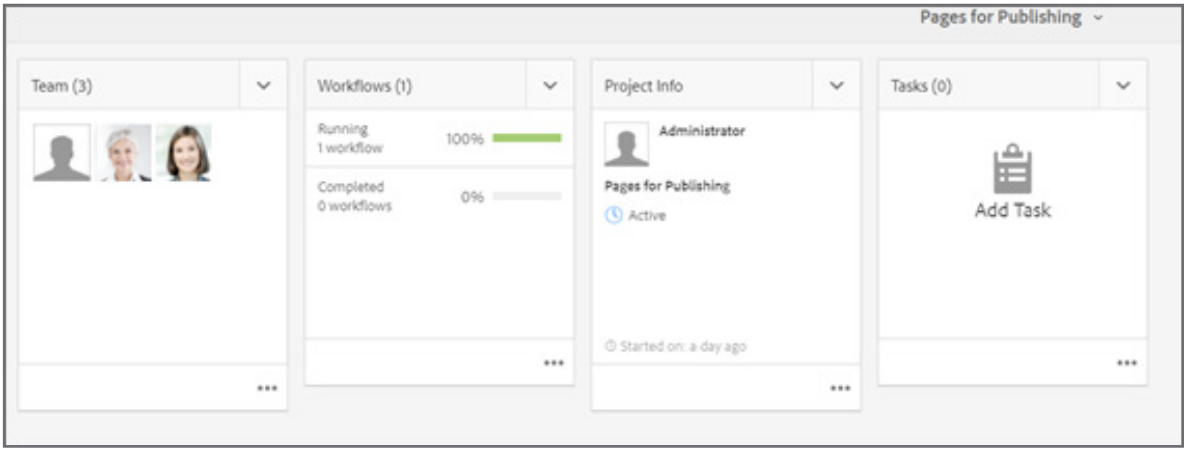
4. Select **Dynamic Editing, Approval, and Publish** and click **Next**, as shown. The **Properties** window opens.



5. Enter the details, as shown:
   - Title: **Edit the Equipment Page**
   - Assign To: **Editors**
   - Content Path: **/content/we-retail/language-masters/en/equipment**

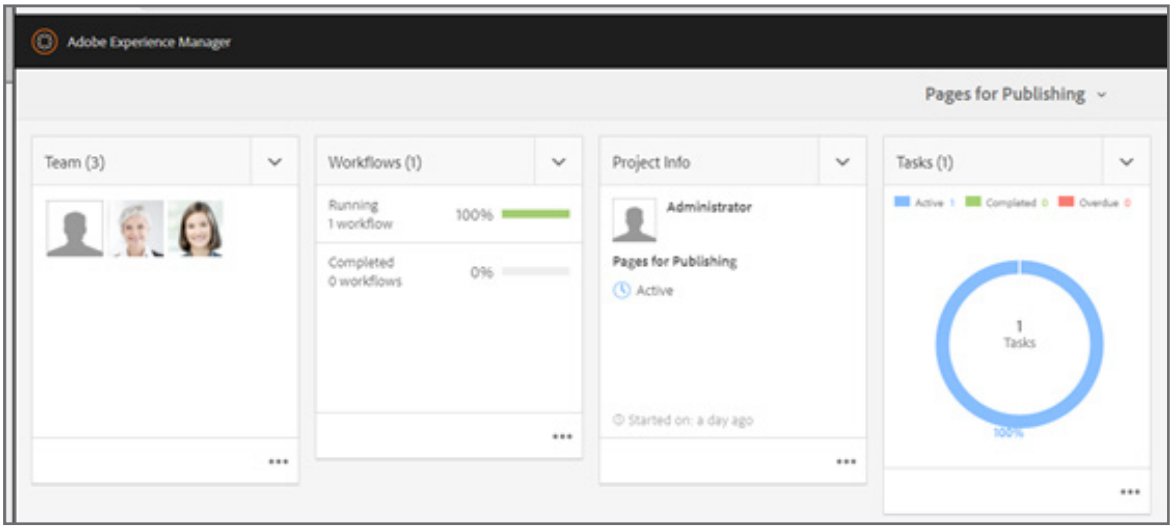


6. Click **Submit** in the top right corner. The Workflow is built, as shown:

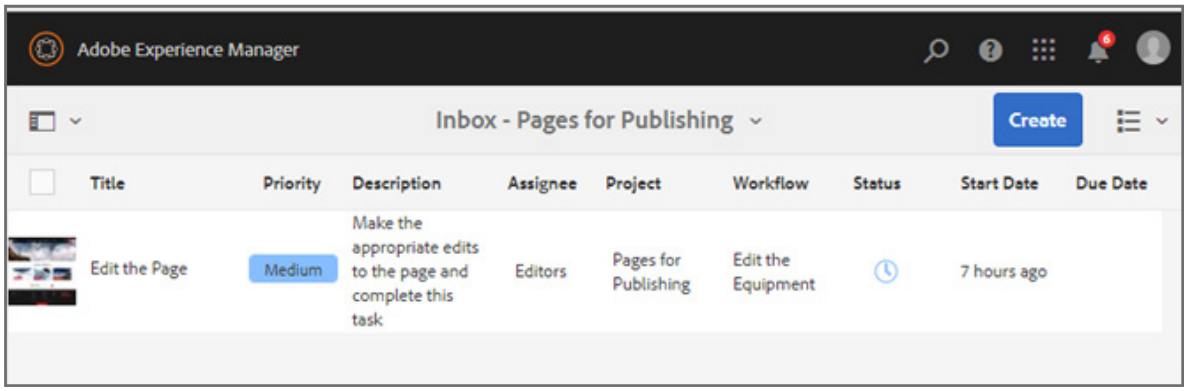


Note: At this point, you should be able to walk through the edit/approval steps. Even though the tasks are assigned to different project groups, the admin can see and run through all the tasks.

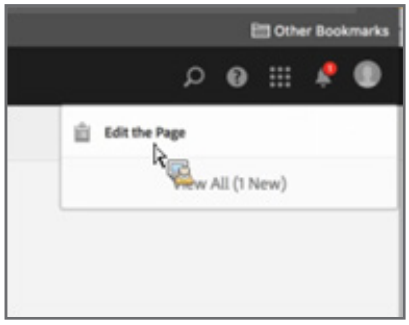
7. Refresh the browser and notice the new Task, as shown:



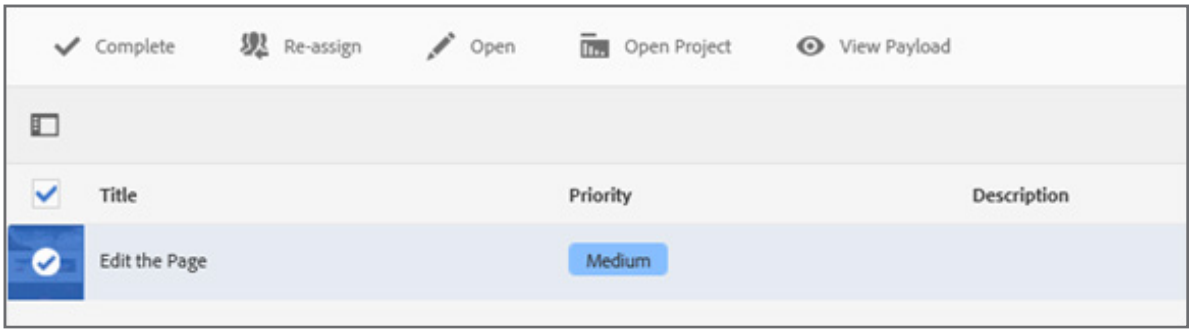
8. Click the ellipsis on the Task Tile. The Inbox opens, as shown:



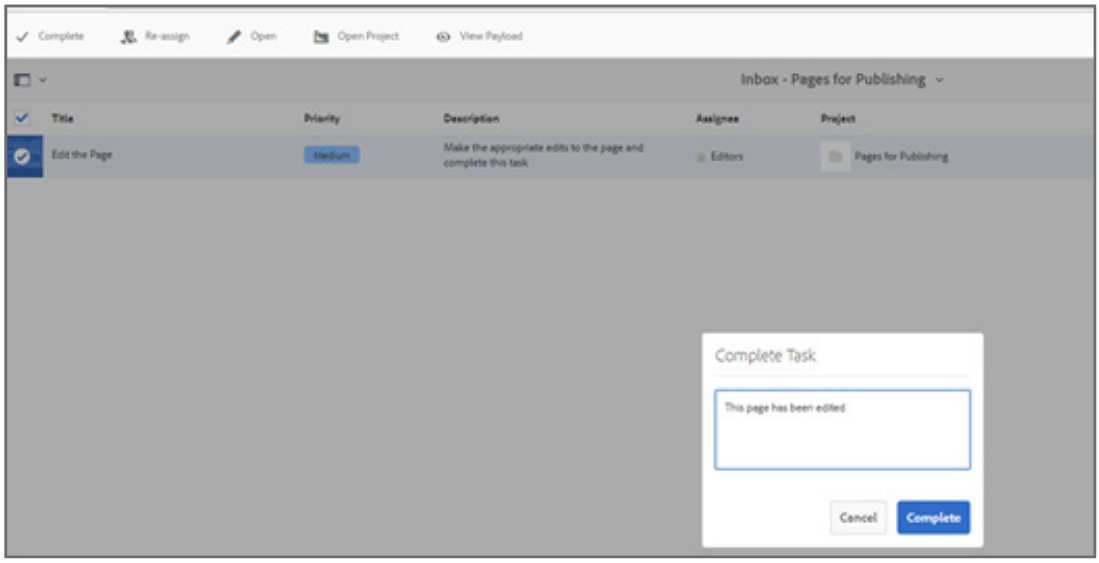
9. Find the new message in the inbox, as shown:



10. Select the **Edit the Page** and click **Complete**, as shown:

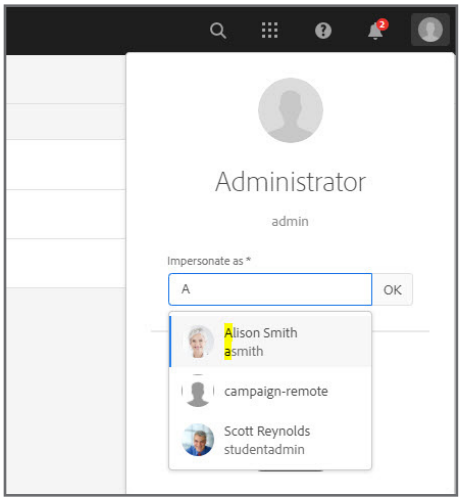


11. In the Complete Task pop-up window, enter a comment as shown and click Complete, as shown. The task is completed.

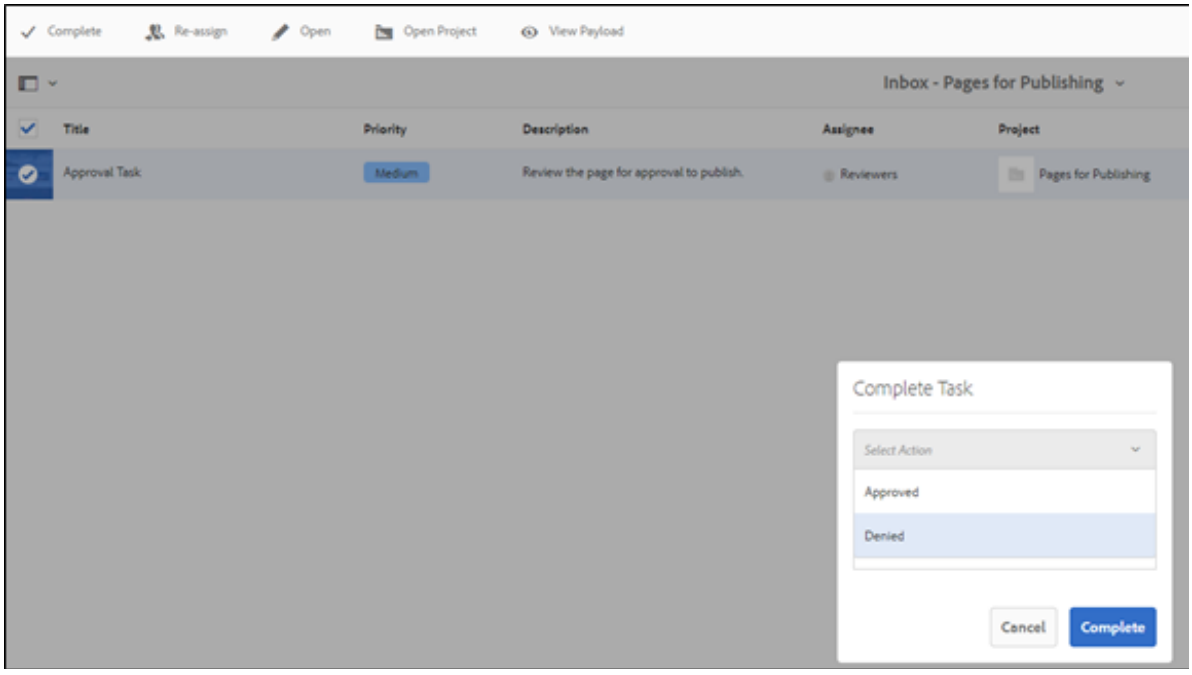


12. Notice there is an **Approval Task**, as shown:

13. In the Impersonate as field, type **Alison Smith**. When her name appears in the drop-down list, click to select it, and then click **OK**, as shown:



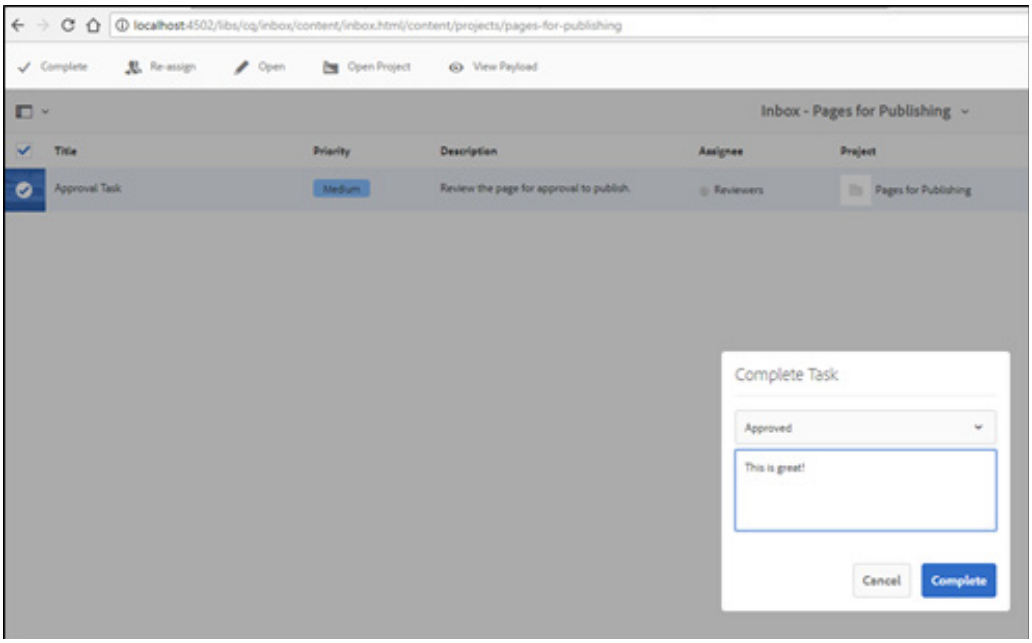
14. Select the **Approval Task**, and click **Complete**. The **Complete Task** pop-up window appears. Notice there is a drop-down with Approved and Denied, as shown:



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.

15. Complete the task by selecting **Approved** and add a comment, as shown:



16. Verify the page is published, as shown: