



Develop Websites and Components in Adobe Experience Manager



student guide

©2018 Adobe Systems Incorporated. All rights reserved.

Develop Websites and Components in Adobe Experience Manager

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

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

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

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

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

All other trademarks are the property of their respective owners.

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

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

05072018

Table of Contents

Adobe Experience Manager Technical Basics	7
Fluid Experiences	8
AEM Architecture	9
AEM Installation	14
Exercise 1: Install AEM author and publish instances	19
Task 1.1: Install and start an AEM author instance using the graphical method	19
Task 1.2: Install and start an AEM publish instance using the graphical method	22
Task 1.3: Start an AEM author instance using the command line method	24
(Optional) Task 1.4: Start AEM using nosamplecontent and change the admin password	25
Authoring Basics	28
AEM Sites Pages	31
Exercise 2: Create and edit a page in AEM	33
Task 2.1: Create a page	33
Task 2.2: Edit a page	37
Developer Tools	43
Exercise 3: Install, create, build, and download a package	48
Task 3.1: Install a package	48
Task 3.2: Create, build, and download a package	54
References	58
Introduction to Content Rendering	60
Java Content Repository Nodes and Properties	61
Folder Structure of Java Content Repository	63
Exercise 1: Create a project structure in JCR	64
Create a Base Page Component	67
Exercise 2: Create a base page component	68
Exercise 3: Create content to render on the page	73
Sling Resolution Process	75
References	80
Exercise 4: Search for a rendering script	81
Exercise 5: Manipulate selectors	85
Inheritance	87
Exercise 6: Inheritance with resourceSuperType	89
Exercise 7: Include other scripts	93

Introduction to HTML Template Language	96
HTL	97
Goals of HTL	98
HTL Syntax	99
Blocks and Expressions	99
References	101
Exercise 1: Render page content by using AEM global objects	102
Exercise 2: Render content value on the page	105
Exercise 3: Render page content by using HTL attributes	107
AEM Sites Development: Key Concepts	109
Templates	110
Core Components and Proxy Components	113
Responsive Layout Editing	116
Context-Aware Configurations	118
Exercise 1: Create and enable an editable template	121
Task 1.1: Create an editable template	121
Task 1.2: Enable the editable template	126
References	127
Creating Editable Templates and Pages	128
Types of Template	129
Creating Editable Templates	130
Exercise 1: Create an editable template	136
Task 1.1: Create a context-aware configuration	136
Task 1.2: Create an empty page template type	139
Task 1.3: Create a development template	143
Proxy Components	146
Exercise 2: Create proxy components	147
Exercise 3: Add content components to the template	149
Exercise 4: Enable and publish the template	152
Creating Pages from Editable Templates	154
Exercise 5: Create a content root	156
Exercise 6: Create a site structure	158
Creating Client-Side Libraries	163
Client-Side Libraries	164
Structure of Client-Side Libraries	164
Organizing Client-Side Libraries	166
Referencing Client-Side Libraries	166
Exercise 1: Create a client-side library	167
Exercise 2: Add a client-side library to a page component	171
Exercise 3: Add a client-side library to a template	176

Working with Components	181
Component Basics	182
HTL Business Logic	187
Exercise 1: Create a basic header component	189
Exercise 2: Add JavaScript business logic to the header component	196
Exercise 3: Add a Sling Model business logic to the header component	199
Dialogs	204
Exercise 4: Create an edit dialog for the component	206
Exercise 5: Add an editconfig node to the component	216
Design Dialogs	218
Exercise 6: Add a component client library to the component	220
Exercise 7: Add a cq:htmlTag to the component	223
Exercise 8: Create a content policy for a proxy component	225
Exercise 9: Create a design dialog for a component	229
Creating Custom Components	238
Features of Components	239
Exercise 1: Create a custom component	241
Exercise 2: Add a dialog field validation to the Stockplex component	245
Exercise 3: Add a base style to the Stockplex component	253
Exercise 4: Add a style system to the Stockplex component	255
Exercise 5: Add a Sling Model as the business logic	262
Exercise 6: Add a selector to the component	266
Exercise 7: Create the localization information	269
Exercise 8: Test the localized content	272
Sling Resource Merger	274
Exercise 9: Extend a core component	276
Exercise 10: Extend the Navigation UI	281
Preparing for Production	287
AEM in Production	288
Exercise 1: Complete the empty page template type	289
Exercise 2: Create production templates	292
Exercise 3: Create a code content package	303
AEM Environment	306
Instances	306
Dispatcher	307
Replication Agents	307
Performance Guidelines	309
Security Checklist	310
AEM in Production Ready Mode	310
References	312

Testing and Debugging Site Content	313
Developer Mode	314
Parameter Debugging	316
Hobbes Testing Framework	318
Exercise 1: Create a Hobbes testing suite	319
Exercise 2: Run the Hobbes testing suite	322
References	324
Customizing AEM Development Environment	325
Eclipse, Maven, and Lazybones	326
Brackets	328
Exercise 1: Install Brackets and the AEM Brackets extension	330
Exercise 2: Make changes to the repository by using Brackets	333
References	337
Appendix	338
Static Templates	338
Dialog Conversion Tool	341
AEM and GDPR Compliance	341



Adobe Experience Manager Technical Basics

Introduction

Adobe Experience Manager (AEM) is a content management system that helps you build and manage online content. It is a Java-based web application that is built using different technologies and has a user-friendly and flexible User Interface (UI). This helps customers build robust and scalable applications. In order to use AEM capabilities effectively, you should first understand the architecture, UI, basic authoring features, and the administrative consoles of AEM.

Objectives

After completing this course, you will be able to:

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

Fluid Experiences

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

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

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

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

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

AEM Architecture

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

Basics of AEM Architecture Stack

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

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



Granite Platform

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



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

Technically, at the core, Granite provides:

- An **application launcher** for a standalone Java or Web application archive for deployment in the existing servlet containers or application servers.
- An **OSGi Framework** into which all applications are deployed.
- **OSGi Compendium Services** to support building applications, such as Log Service, Http Service, Event Admin Service, Configuration Admin Service, Declarative Services, and Metatype Service.
- A comprehensive **Logging Framework** providing various logging APIs, such as SLF4J, Log4F, Apache Commons Logging, and OSGi Log Service.
- A repository based on **Apache Jackrabbit Oak** and **JSR-283**.
- The **Apache Sling Web Framework**.

Granite UI

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

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

It also adheres to the following requirements:

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



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

OSGi Framework

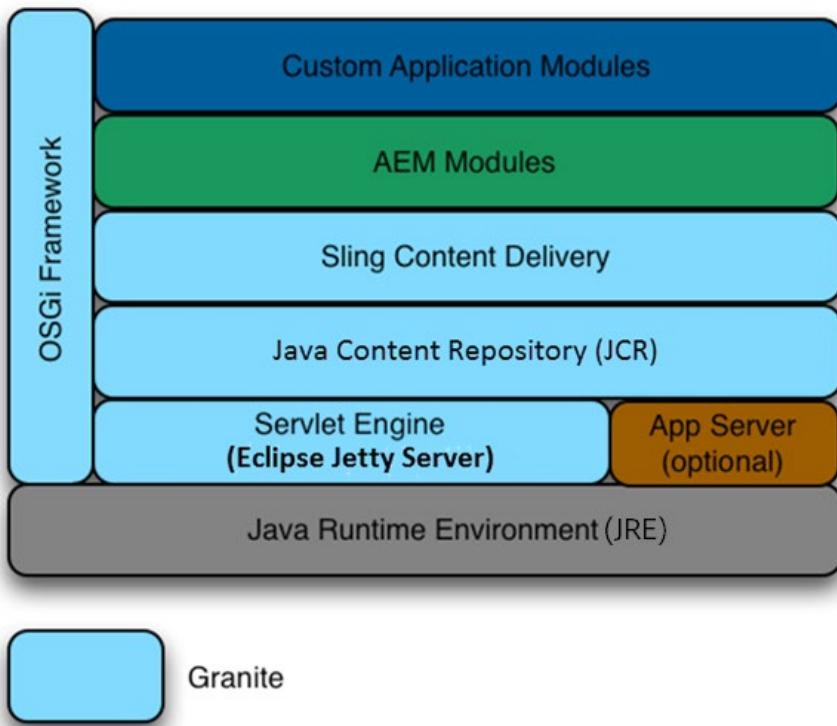
OSGi enables a collaborative and modular environment, where each application may be built and implemented as a small bundle. Each bundle is a collection of tightly coupled, dynamically loadable classes, JAR files, and configuration files that explicitly declare their external dependencies.



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

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

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



Apache Felix

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

JCR

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

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

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

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

The advantages of using JCR are:

- It provides a generic application data store for structured and unstructured content. While file systems provide excellent storage for unstructured and hierarchical content, the databases provide storage for structured data. This way, JCR provides the best of both the data storage architectures.
- It supports namespaces. Namespaces prevent naming collisions among items and node types that come from different sources and application domains. JCR namespaces are defined with a prefix, delimited by a single colon (:). For example, jcr:title. This means that this title property is defined in the jcr namespace.
- It provides one interface to interact with text and binary data. This helps in easy access and management of data in comparison to storing it in multiple places.

Apache Sling

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

The key features of Apache Sling include:

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

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

Application-Level Functions

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

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

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

AEM Installation

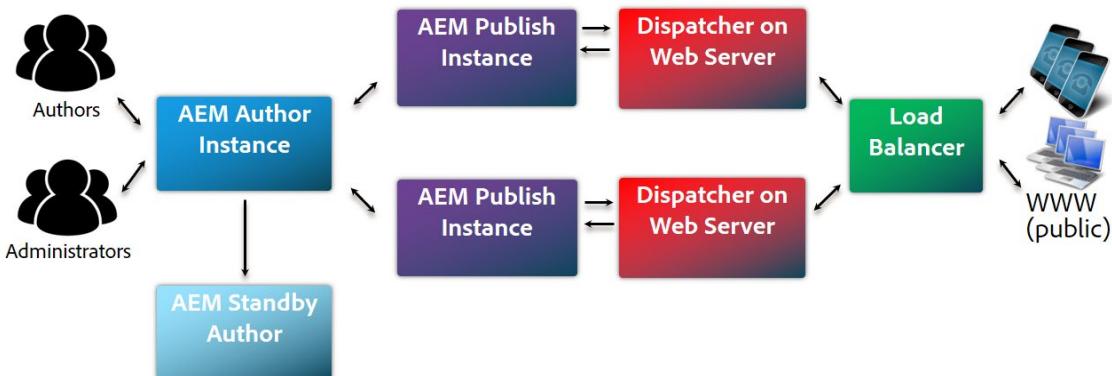
AEM runs on most operating systems that support the Java platform. All client interactions with AEM are done through a web browser.

Instances

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

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

The following graphic depicts a typical AEM implementation:



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

Installation Prerequisites

To install AEM, you need:

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

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



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



Note: Windows users may need to ensure their system's environment variables are set appropriately to run Java 1.8 before installing AEM. Open a command prompt and type java -version to ensure Java is set up properly. When you run the command, it should show the following result (java version "1.8.0_144"):

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

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

You can download Java 8 for your system from the following link:

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

Graphical and Command Line Methods to Install AEM

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

Graphical Method

This method involves using the *.jar file to start an AEM instance. In a Windows or Mac OS environment, you can double-click the aem-author-4502.jar file to start an author instance, or the aem-publish-4503.jar file to start a publish instance.

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

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



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

Command Line Method

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

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

Prior to the installation, you may want to know which parameters are available to configure quickstart.

Enter the following command in the command prompt to display a complete list of optional parameters:

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

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

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

-Xms --> assigns the initial heap size

Default value	64 MB for a JVM running on 32-bit machines, or 83 MB for 64-bit machines
Recommended	Specific to physical memory available and expected traffic
Syntax	-Xms512m (sets the initial heap size to 512 MB)

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

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

Example:

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

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

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

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

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

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



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

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

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

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

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

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

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

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

Additonal Methods to Start AEM: Batch Files

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

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

Exercise 1: Install AEM author and publish instances

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

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

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



Note: If you are attending a v/ILT class using ReadyTech, steps 1 through 3 were completed for you. Skip ahead to step 4.

To install an AEM author instance:

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

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

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

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



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

By renaming the file, you use a convention of passing the instance name (Webpathcontext) and port number through the file name, so no user interaction is needed during the installation process.

If no port number is provided in the file name, AEM will select the first available port from the following list in this specific order: 1) 4502, 2) 8080, 3) 8081, 4) 8082, 5) 8083, 6) 8084, or a random port.

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



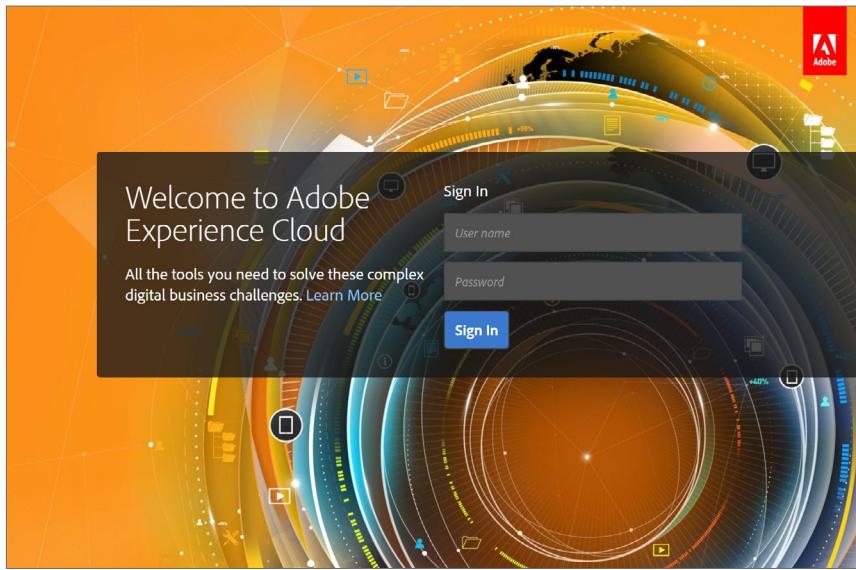
You have now completed the installation of AEM.



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

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

In addition, after AEM starts, your default browser will automatically open to AEM's start URL (where the port number is the one you defined on installation). For example: <http://localhost:4502>
A sign in screen is displayed as shown below:



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



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

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

This is the extracted repository that is created upon installation of AEM.

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

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

 **Note:** If you are attending a v/ILT class using ReadyTech, steps 1 through 3 were completed for you. Skip ahead to step 4.

To install an AEM publish instance:

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

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

 **Note:** If you have multiple author and publish instances, a best practice to consider is using an even/odd numbering paradigm for port numbers.

So, your author instances would be:

4502, 4504, 4506, ...

And, your publish instances would be:

4503, 4505, 4507, ...

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

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

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



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



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

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

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

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

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



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



To start an instance:

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

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

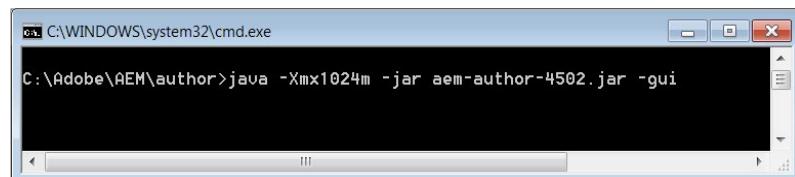
2. Prior to the installation, you may want to know which parameters are available to configure the AEM quickstart. Enter the following command to display a complete list of optional parameters you can use to install and start AEM (this command will not install and start AEM):

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

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

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

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



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

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

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

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

To begin installation:

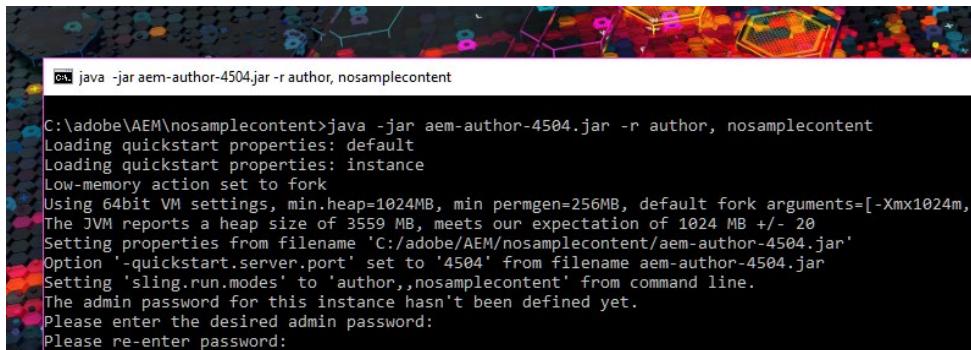
1. Create a separate sibling directory called **nosamplecontent** on your machine in the same location as your author and publish directories.
In Windows (and also if you are using ReadyTech), this would be: **C:\adobe\AEM\nosamplecontent**
2. Copy the **quickstart *.jar** file and the **license.properties** file from your author directory and paste it into this new directory.
3. Rename your quickstart file to **aem-author-4504.jar**.
4. Open a command prompt in the directory where your quickstart *.jar file is copied.
5. In the command prompt, run the following command:

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

This installs another author instance with the following instructions:

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

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



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

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

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

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

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

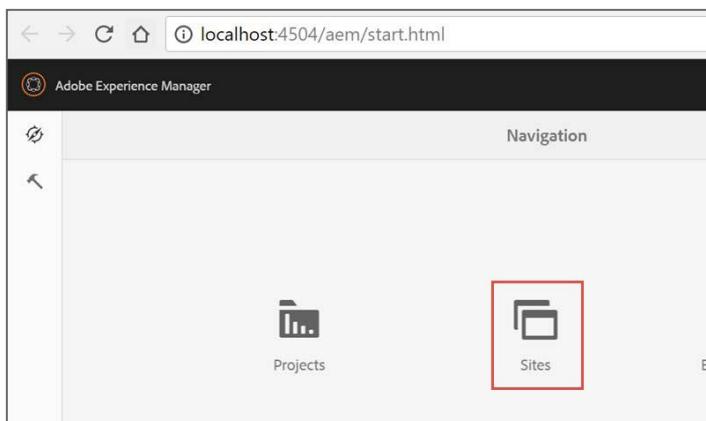


Note: If you did not initially install AEM using the command line, but used the *.jar file, the admin password is set for you. The default sign in using this method is:

User: admin

Password: admin

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

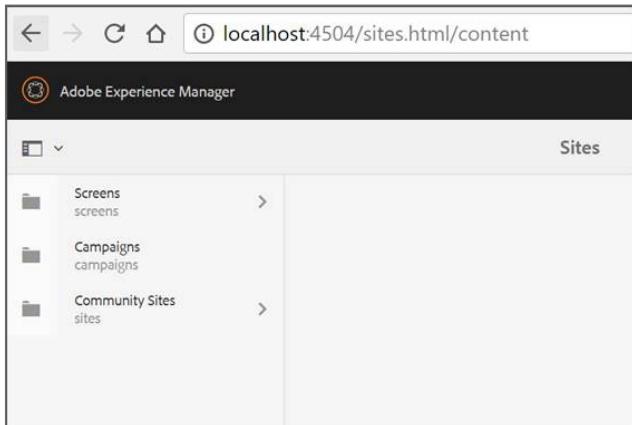


10. In your command window, type **CTRL+C** to shut down your additional AEM author instance on port 4504 (applicable for Windows only). Be patient as this may take up to 1 or 2 minutes.



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

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



Authoring Basics

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

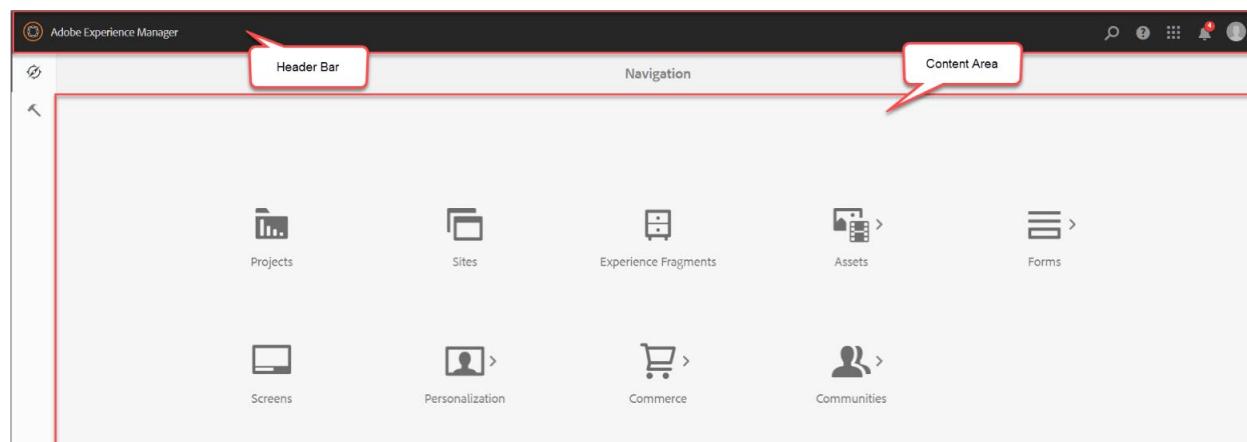
AEM UI: Key Features

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

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

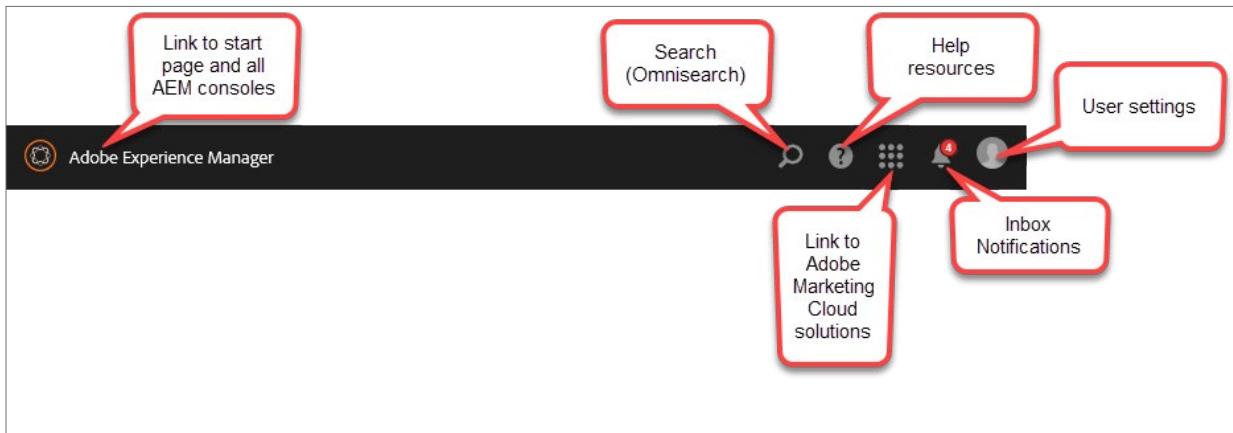
Device UI Actions	Desktop UI Actions
Tap	Click
Touch-and-hold	Double-click
Swipe	Hover

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



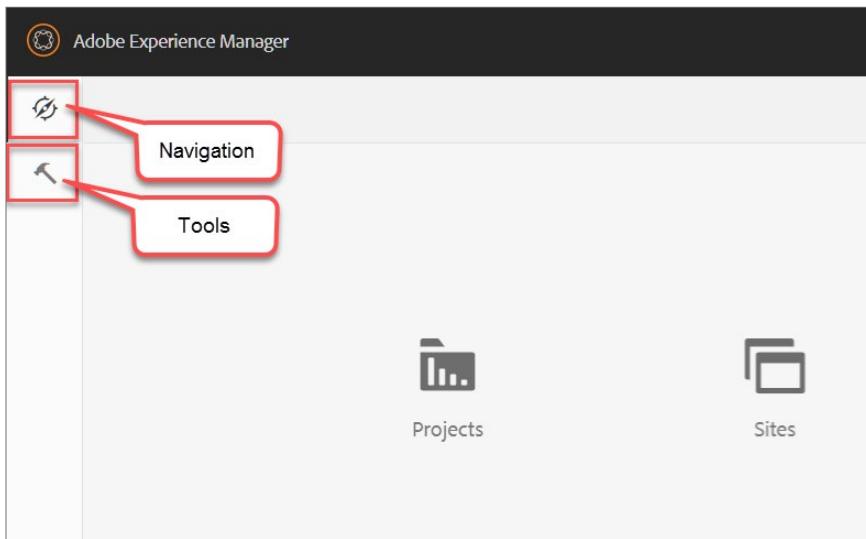
The content area, by default, displays all applications and capabilities of AEM, such as Projects, Sites, Experience Fragments and Assets.

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



Navigation and Tools

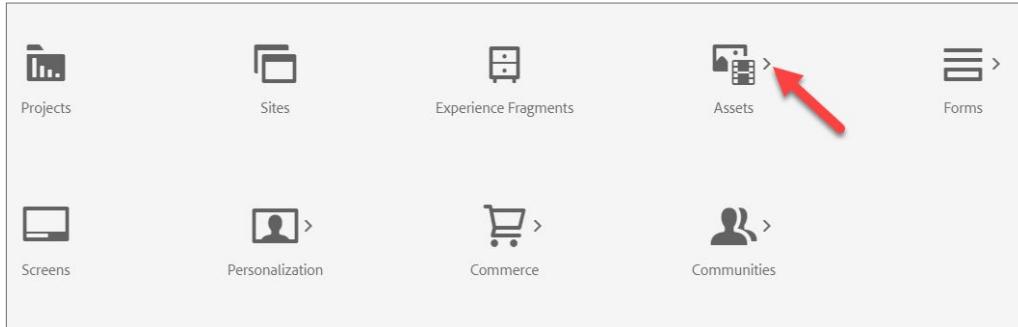
There are two main sections in the AEM UI: Navigation and Tools.



Navigation

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

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



Tools

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

AEM Sites Pages

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

Creating Pages

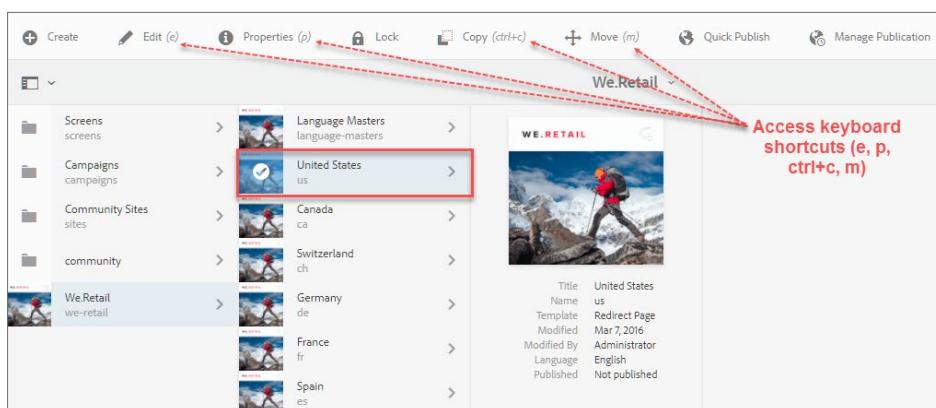
To create a page in AEM Sites, you need to follow a simple wizard where the page template is specified, and the page is given a title and a name. The title is displayed to the user and the name is used to generate the page URL (and can be derived from the title).

Editing Pages

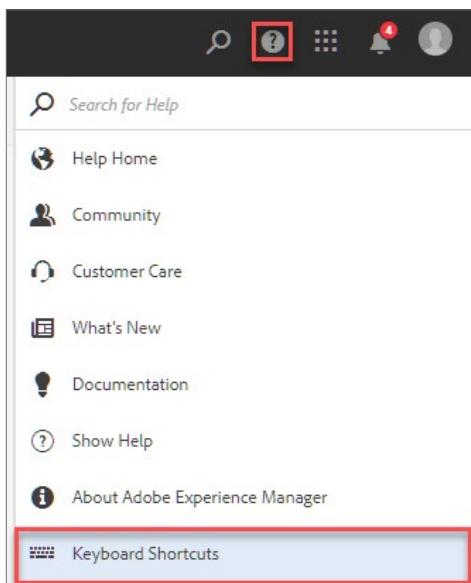
After you create a page, you can edit and add content to it. You can add content by dragging the components available in the side panel onto your page in the page editor.

AEM Keyboard shortcuts

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



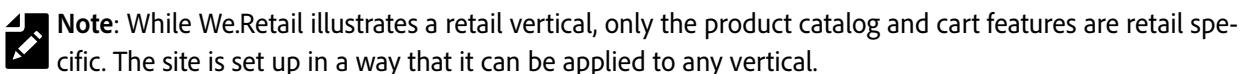
You can customize or enable/disable shortcuts by navigating to **Help > Keyboard Shortcuts**:



Reference Site: We.Retail

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

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



Exercise 2: Create and edit a page in AEM

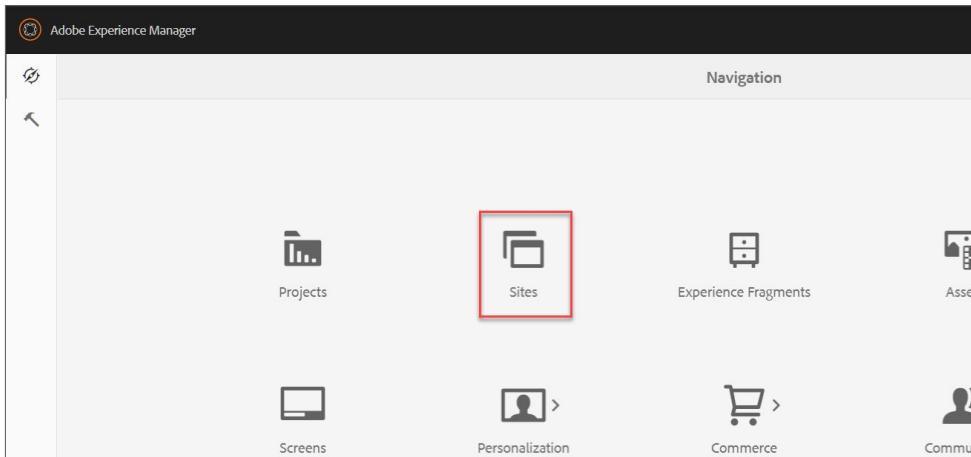
Scenario: As an author, you need to create and edit a page in AEM Sites under the built-in We.Retail site hierarchy.

Task 2.1: Create a page

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

To create a new page:

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



The column view is displayed.

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

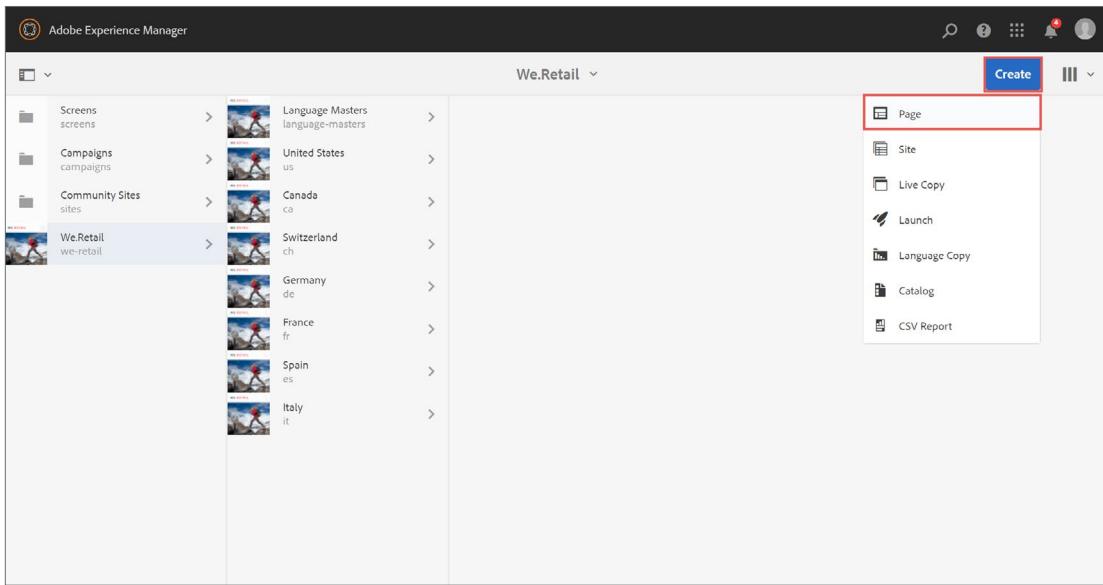
3. In the column view, click the right-pointing arrow next to We.Retail as shown:

The screenshot shows the Adobe Experience Manager interface in column view. The left sidebar lists categories like Screens, Campaigns, Community Sites, and We.Retail. The We.Retail item is selected, indicated by a red box around its thumbnail and a red arrow pointing to the right-pointing arrow icon next to it. A callout bubble labeled "Column view" points to the top right corner of the interface.

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

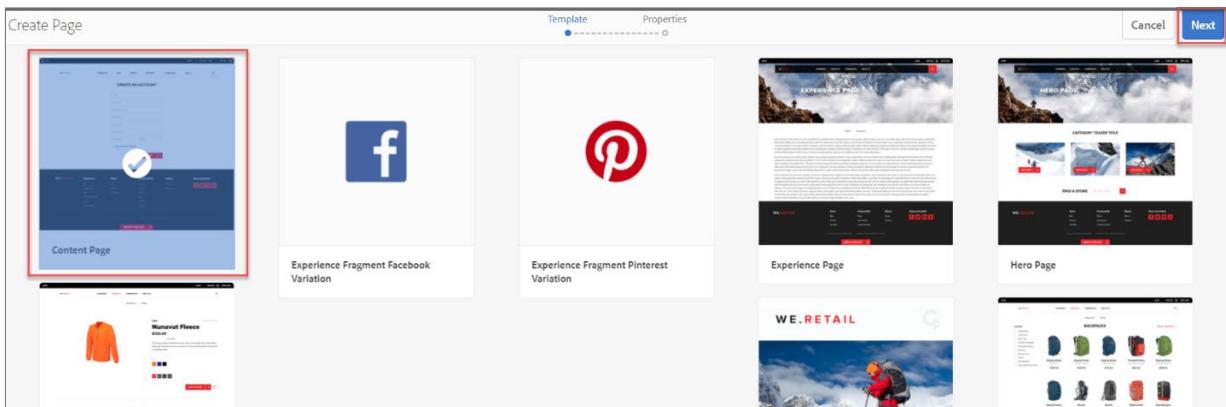
The screenshot shows the Adobe Experience Manager interface in column view. The left sidebar lists categories like Screens, Campaigns, Community Sites, and We.Retail. The We.Retail item is selected, indicated by a checkmark in its thumbnail. A red box highlights the thumbnail.

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



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

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



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

Create Page

Template — Properties

Back Create

Basic Advanced Social Media

Title and Tags

Title * Demo Page

Name demoPage

Tags

7. Click **Create** in the top-right corner to create the page. A **Success** dialog box is displayed with a message that your page has been created.
8. Click **Done**. The new page appears as a child page of We.Retail. The page name (**Demo Page**) appears in the column view. You can also see the page title (**demoPage**) below the page name.

Screens screens

Campaigns campaigns

Community Sites sites

community

We.Retail we-retail

Language Masters language-masters

United States us

Canada ca

Switzerland ch

Germany de

France fr

Spain es

Italy it

Demo Page

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

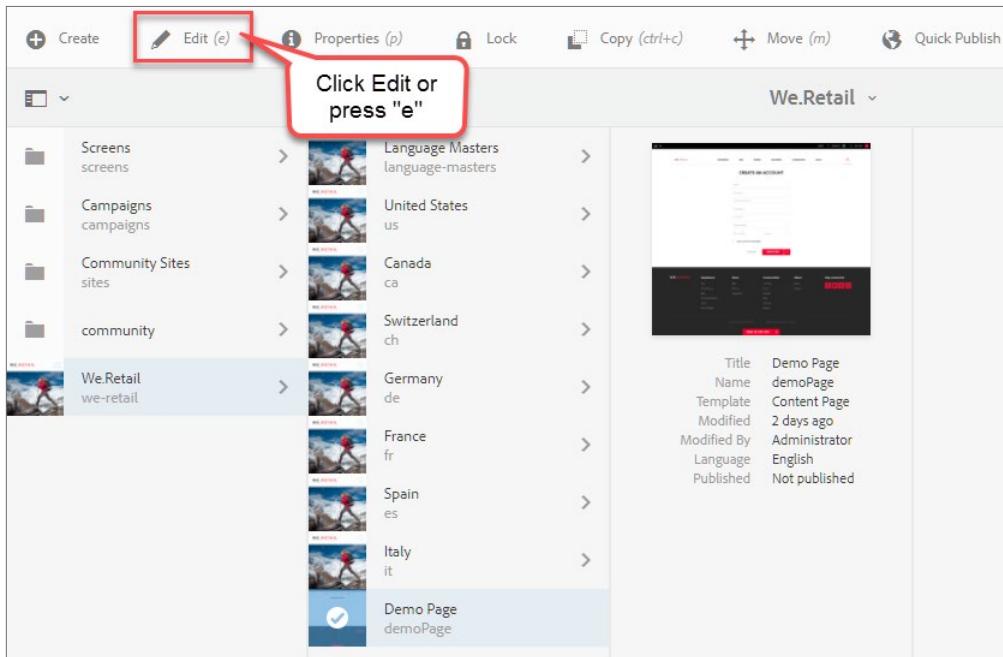
You can create subpages of any page using the same process.

Task 2.2: Edit a page

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

To edit the page:

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



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

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

Modes

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

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

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

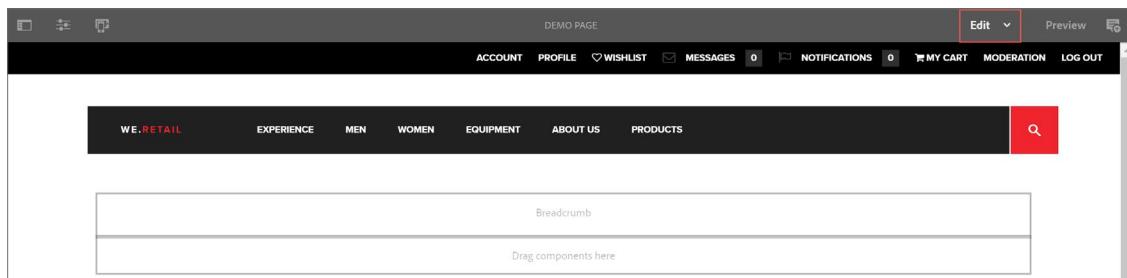
i Mind, to avoid accidentally navigating away from the page while editing, all links are disabled in Edit mode.

Don't show this again

• ○ ○ ○ ○

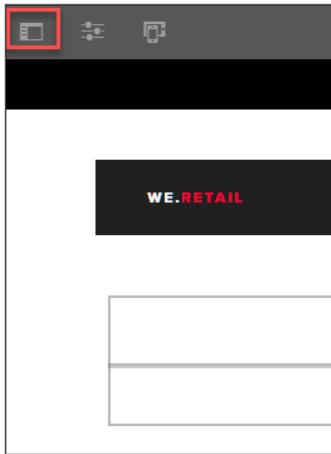
Close **Next**

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



To add a text component to the page:

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

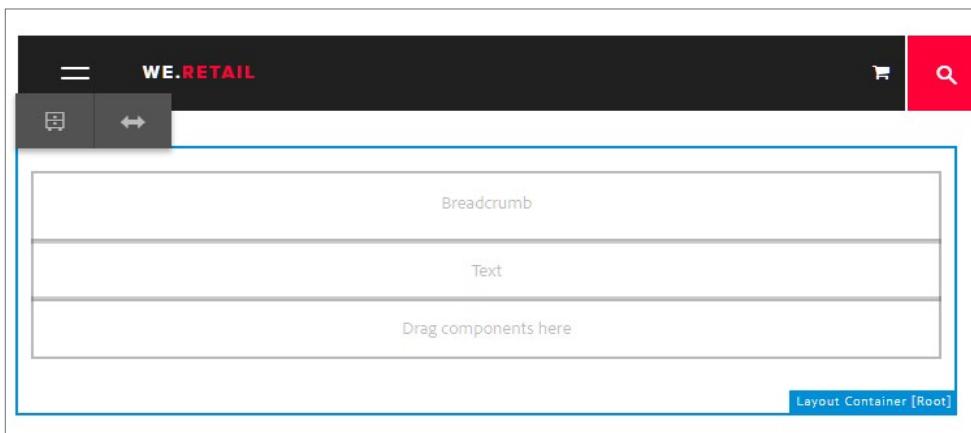


The side panel is displayed.

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

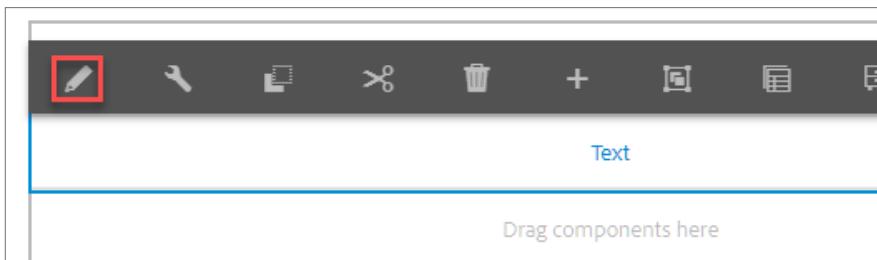


Your editor should look like the following:

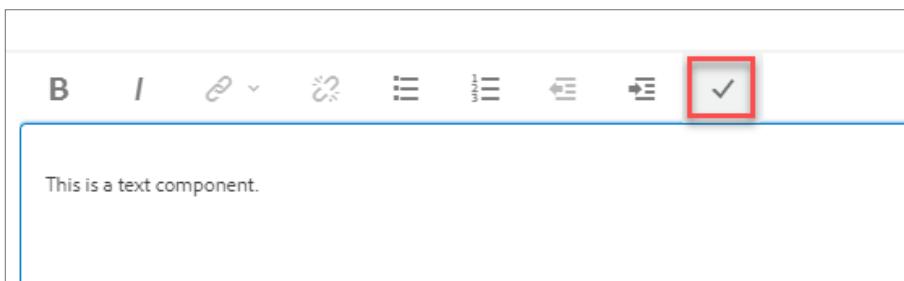


Tip: If the ordering is not correct, just drag the text component again to the **Drag components here** box.

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



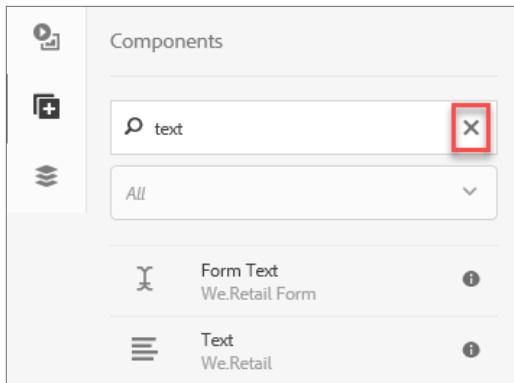
8. Add sample text of your choice in the text editing form that appears, and click **Done** (checkmark icon) to save the changes. The text is added to the page.



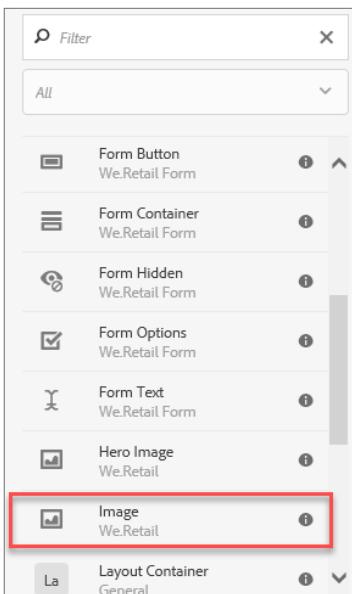
You have now added the first component to your page!

To add the image component to your page:

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



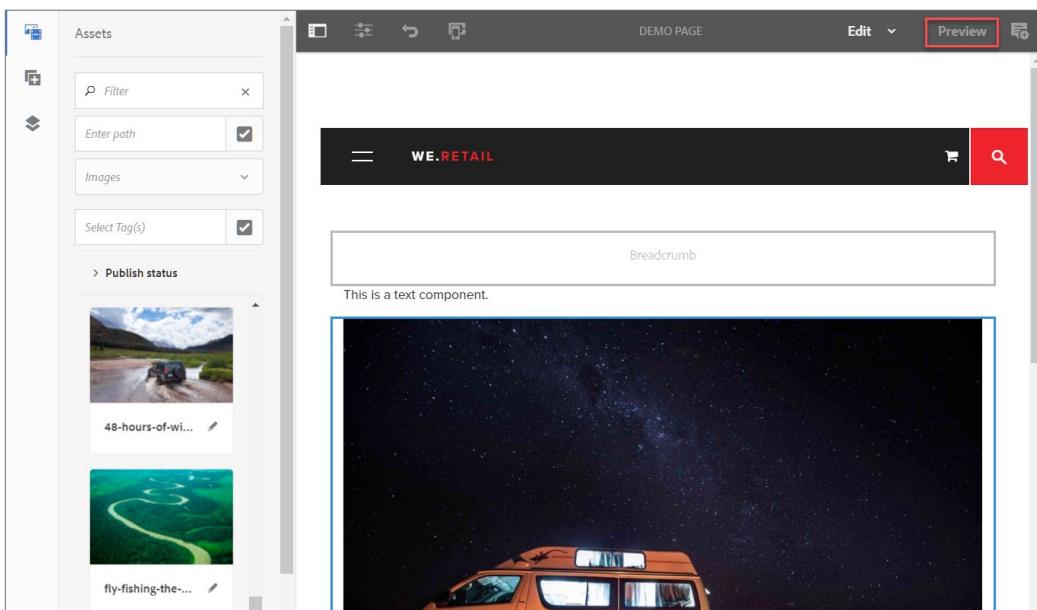
10. Scroll down and drag the **Image (We.Retail)** component onto the **Drag components here** box as shown:



11. Click the Assets icon in the side panel as shown:

12. Drag any image from the **Assets** tab on the **Image** component. AEM will add the image to the page.

13. Click **Preview** in the top-right corner to preview the changes to your page:



14. Press **CTRL+SHIFT+M** to go back to the edit mode. You may use this method to toggle between **Preview** and **Edit** mode, as you may need to switch often when adding and testing page content.

Note that this keyboard shortcut is AEM specific and does not depend on the operating system or the browser. In other words, this shortcut is universal to AEM.

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

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

Developer Tools

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

- CRXDE Lite
- Web Console
- Package Manager

CRXDE Lite

CRXDE Lite is embedded into AEM and allows you to perform common development and administration tasks within the browser. It is a light Integrated Development Environment (IDE) for quick access to the JCR. Because it is embedded in the server and is always available, CRXDE Lite is often the preferred tool for administrators and developers for working with nodes and properties in the JCR. It gives quick and direct access to the repository for monitoring, configuration, and development.

 **Note:** CRXDE Lite is primarily used by front-end developers who develop components and client libraries in AEM. DE in CRXDE stands for Development Environment.

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

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

You can use CRXDE Lite for:

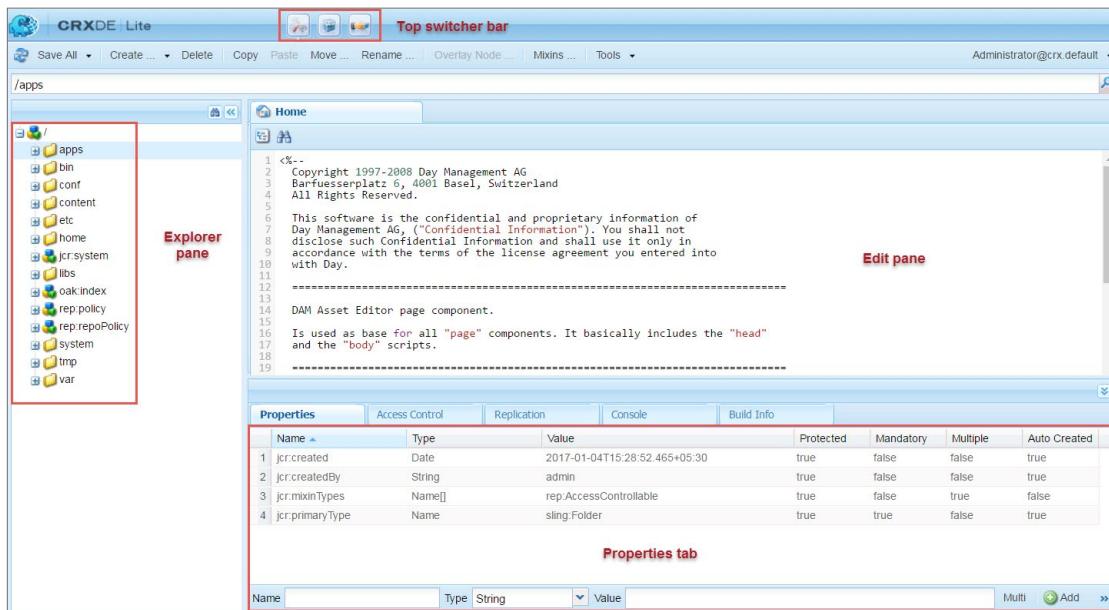
- Code/content validation
- Product training
- Operations debugging
- Overlays from /libs

The CRXDE Lite UI contains:

- **Top switcher bar:** Enables you to quickly switch between CRXDE Lite, Package Manager, and Package Share
- **Explorer pane:** Displays a folder tree structure of all the nodes in the repository

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

- Select the node and view its properties in the **Properties tab**. Examine all the JCR properties of different nodes.
- Right-click the node and perform an action on it, such as renaming the node, creating a new node, creating a folder, and creating a file.
- **Edit pane:** Allows you to edit the code. Double-click a file, such as a .jsp or a .html file, in the **Explorer pane** to display its content. You can then modify the code and save the changes.
- **Properties tab:** Displays the properties of the node that you selected. You can add new properties or delete existing ones.



Web Console

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

You can access the console at <http://localhost:4502/system/console> or by navigating to **Tools > Operations > Web Console** within AEM.



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

The Web Console contains a selection of tabs for maintaining OSGi bundles. The most important menu selections under the OSGi tab are:

- **Bundles**- Used for installing and managing bundles.
- **Components**- Used for managing and controlling the status of components required for AEM.
- **Configuration**- Used for configuring OSGi bundle, and is the underlying mechanism for configuring AEM parameters.



Package Management in AEM

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

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

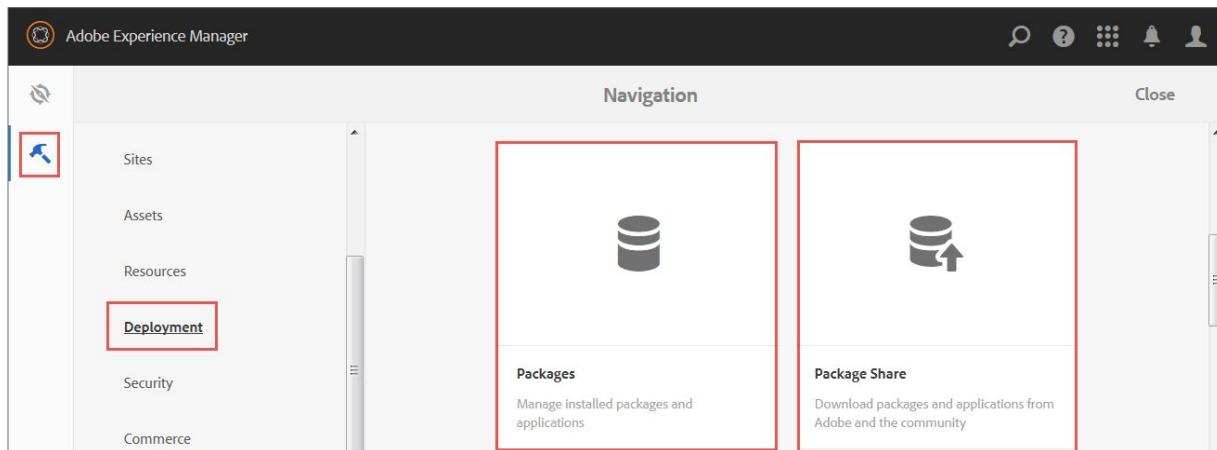
A package can contain:

- Page-related content
- Project-related content
- Assets-related content
- Vault meta information, such as filter definitions and import configuration information
- Other package information such as package settings, package filters, package screenshots, and package icons

You can use packages for any of the following:

- Install new functionality
- Transfer content between instances
- Back up repository content
- Export content to the local file system

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



Package Manager

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

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

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

Creating and Building New Packages

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

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

- **Rebuild:** Helps rebuild the package if there is a change in the repository content.
- **Edit:** Helps edit filters or rules applied to the package.
- **Test:** Helps perform a dry run of the installation.
- **Rewrap:** Helps recreate the package with additional information such as thumbnails and icons.

Downloading Packages to the File System

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

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

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

Package Share

Package Share is a centralized server where public packages are made available. These packages may include hotfixes, new functionality, updates or documentation. You can search, download, and install any package either to your instance or to your local file system.

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

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

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

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

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

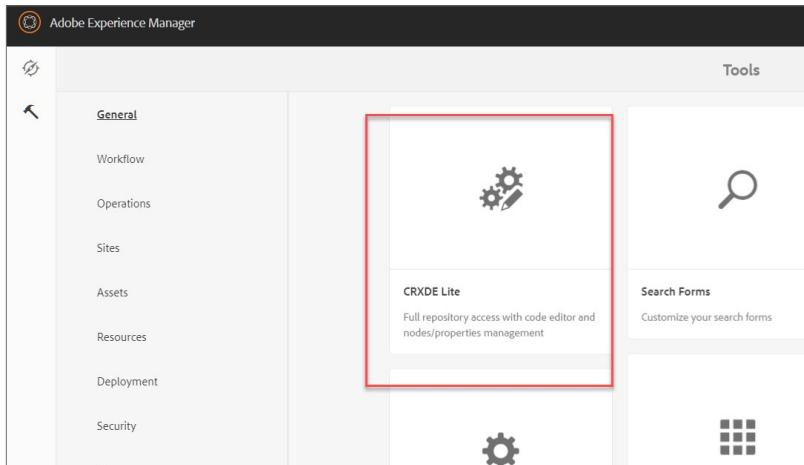
Scenario: As an AEM developer or administrator/development operations, you need to own all package management tasks including validating, installing, creating, building and downloading packages in AEM.

Task 3.1: Install a package

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

To validate a package:

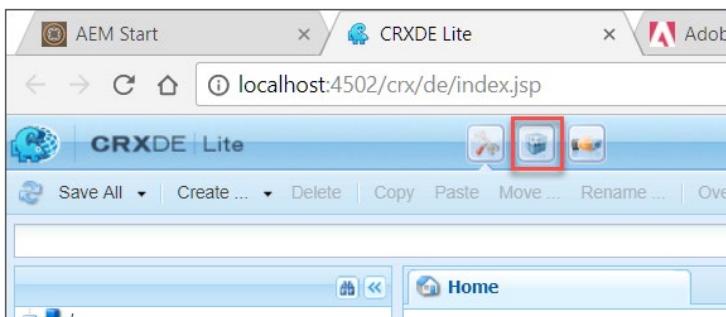
1. Ensure that you are logged on to your AEM author instance on port 4502 (<http://localhost:4502>).
2. Navigate to Tools > **CRXDE Lite** in your AEM author instance as shown:



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

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

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

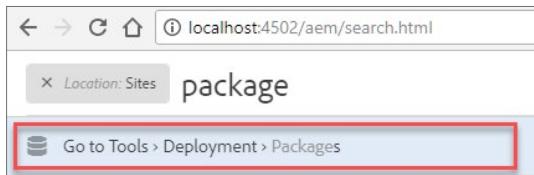


This will take you to the AEM Package Manager tool.

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

Tip: Other ways to navigate to Package Manager include:
Use Tools > Deployment > Packages within AEM.

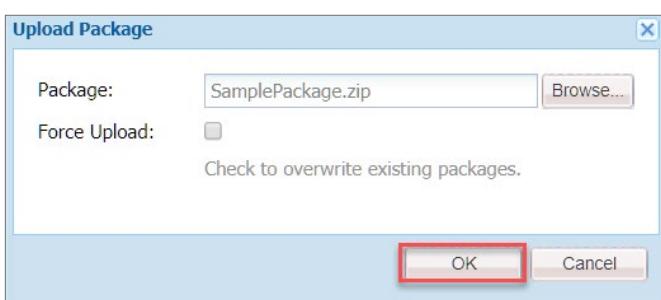
Type / within AEM to use the AEM Omnisearch feature. Type **package** and click the result:



- Click **Upload Package**.



- In the **Upload Package** dialog box, click **Browse** and select the **SamplePackage.zip** package from the **Exercise_Files** (in the \Module 4 – Package Manager folder) provided to you.
- Click **Open**, and then click **OK** as shown:



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

SamplePackage.zip
Build: 1 | Last built 2016/04/14 | admin
Install | 5.5 MB

Edit Build Install Download Share More ▾

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

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

SamplePackage.zip
Build: 1 | Last built 2016/04/14 | admin
Edit Build Install Download Share More ▾

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

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

More ▾

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

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

Activity Log

```
Validate Package: /etc/packages/training/SamplePackage.zip
Thu Mar 08 2018 15:53:49 GMT-0800 (Pacific Standard Time)

Validating content package

No unsatisfied OSGi package imports. No overlay rebase warnings. No ACL warnings.
Package validated in 0ms.
```



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

To install your package:

9. Click **Install** as shown:



The **Install Package** dialog box appears.

10. Ignore the **Advanced Settings** area and click **Install**.

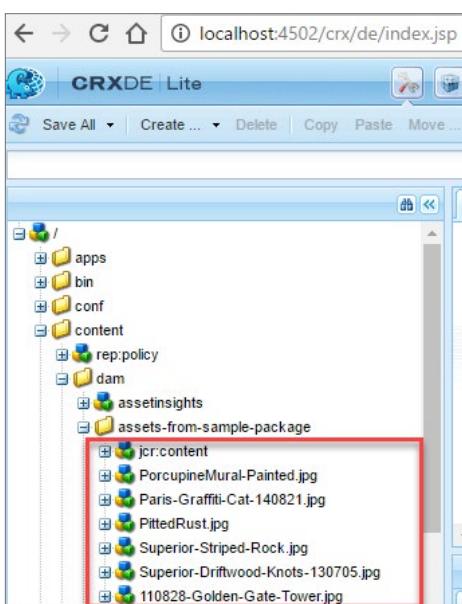
11. Check the **Activity Log**. You can see the content was added from the package. In the activity log, note that the contents of the package consist of a set of image assets being added to /content/dam/assets-from-sample-package in the JCR repository.

12. Click the Develop icon as shown:



This takes you back to CRXDE Lite.

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



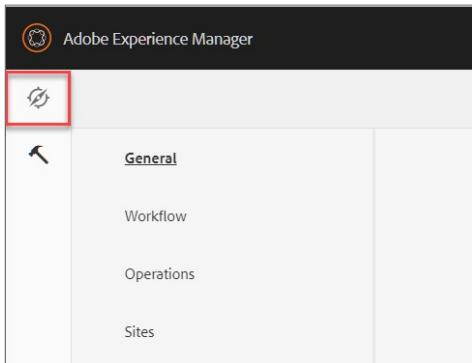
14. Click one of the nodes (such as **Superior-Striped-Rock.jpg**). Note that in the **Properties** tab, properties of that node are shown. In addition, each file has a child `jcr:content` node. So, at a basic level, all content in the JCR consists of nodes and properties.

The screenshot shows the AEM authoring interface. On the left is the JCR tree view, which lists several nodes under the 'assets-from-sample-package' folder. One node, 'Superior-Striped-Rock.jpg', is selected and highlighted with a blue background. A red arrow points from the text 'Node' to this selected node in the tree. To the right of the tree is a large red speech bubble pointing towards the 'Properties' tab of the node details panel. The 'Properties' tab is active, showing a table of properties for the selected node. The table has columns for Name, Type, and Value. The data is as follows:

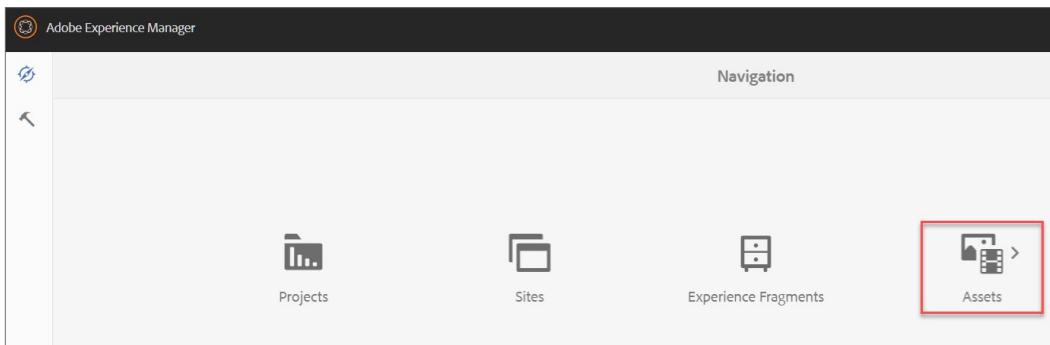
Name	Type	Value
1 jcr:baseVersion	Reference	3049616
2 jcr:created	Date	2018-03
3 jcr:createdBy	String	admin
4 jcr:isCheckedOut	Boolean	true
5 jcr:mixinTypes	Name[]	mix:vers

15. In your other browser tab, navigate back to the AEM start (<http://localhost:4502>).

16. Click the Navigation icon, then click **Assets** as shown:



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



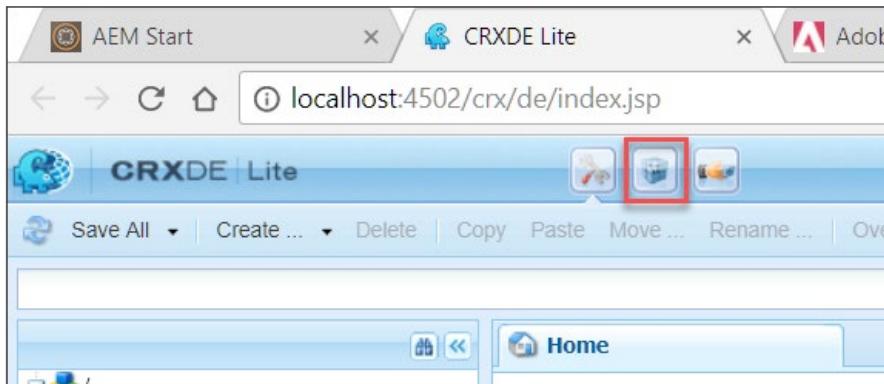
 **Note:** You may see a **Product Navigation** tutorial dialog guiding you through the navigation. You may click **Next** to proceed through the tutorial and learn the basic AEM user interface elements and navigation, or you may click **Close** to hide the tutorial.

Task 3.2: Create, build, and download a package

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

To create a package:

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



The **Package Manager** screen appears.

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

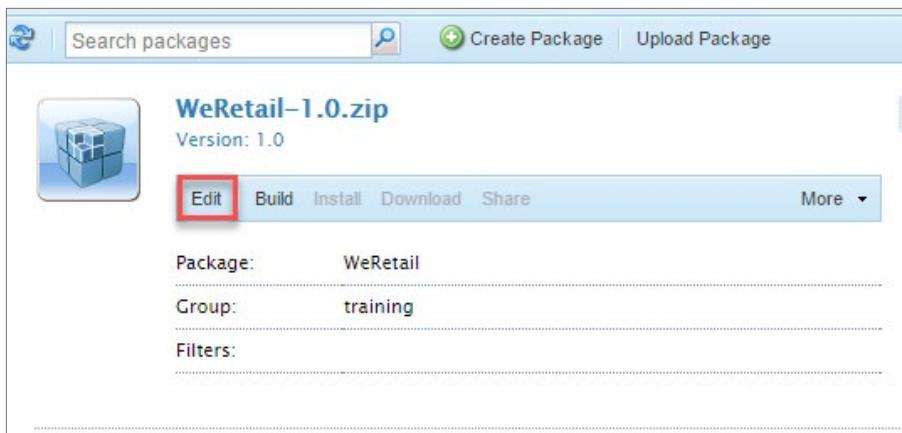


3. In the **New Package** dialog box, enter the following details:

- a. **Package Name:** We.Retail
- b. **Version:** 1.0
- c. **Group:** training

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

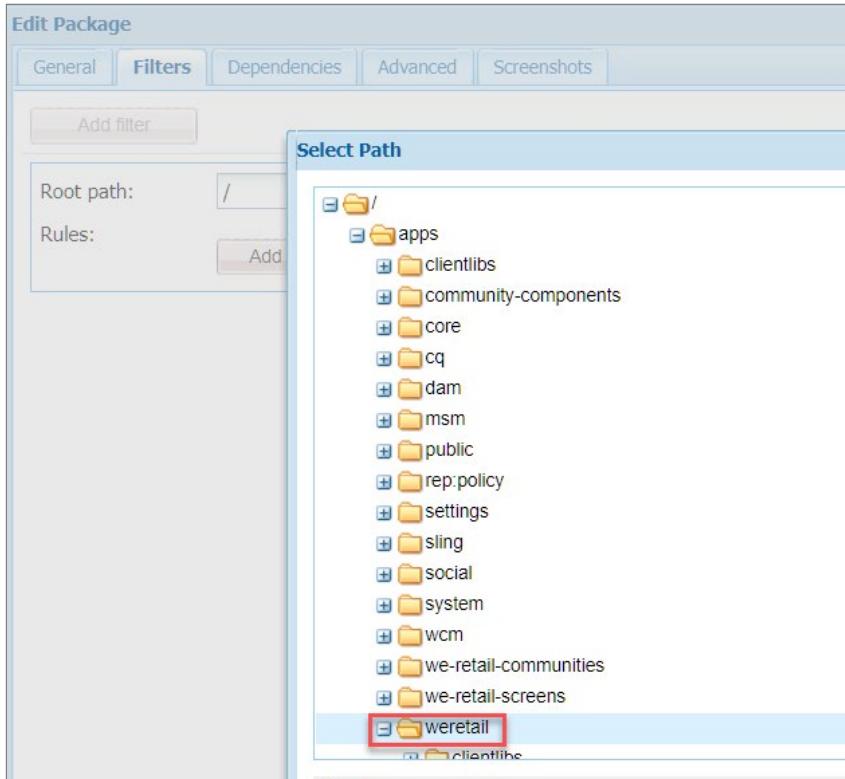
5. Click **Edit** on the newly created package as shown:



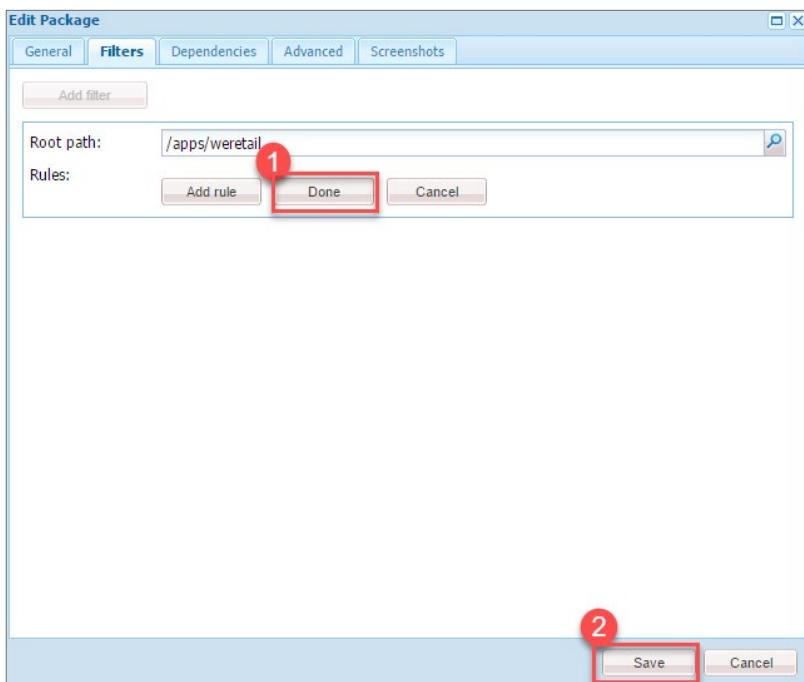
6. To add filters to the package, click the **Filters** tab, and click **Add Filter**.

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

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



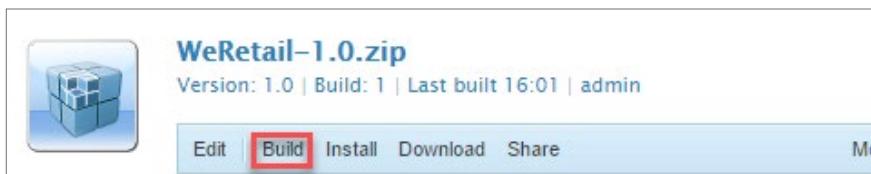
8. Click **Done**, and then click **Save**, as shown:



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

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

10. Click **Build** again in the confirmation dialog box. The package is now ready for download.



-
11. Click the **Download** link to download a copy of the package to your computer. The package is download to your computer's default download folder for the browser.
-

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



The screenshot shows the 'Groups' section of the AEM Package Manager. It lists several package groups with their respective counts:

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

The 'training (2)' group is highlighted with a red border.

References

You can use the following links for more information on:

- AEM AEM 6.4 Help Content (main page):

<https://helpx.adobe.com/support/experience-manager/6-4.html>

- Architecture:

 › OSGi: <http://www.osgi.org>

 › JCR 2.0 Specification: http://www.day.com/specs/jcr/2.0/2_Introduction.html

 › JSR-283: <https://jcp.org/en/jsr/detail?id=283>

 › Apache Sling: <https://sling.apache.org/>

 › Apache Felix: <https://felix.apache.org/>

 › Granite UI (based on Coral 3):

<https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/touch-ui-concepts.html>

- Installation:

 › Official list of supported platforms for AEM:

<https://helpx.adobe.com/experience-manager/6-4/sites/deploying/using/technical-requirements.html>

 › Local Machine Operating System Requirements for AEM:

<https://helpx.adobe.com/experience-manager/6-4/sites/deploying/using/deploy.html#GettingStarted>

- Authoring:

 › Authoring Pages:

<https://helpx.adobe.com/experience-manager/6-4/sites/authoring/using/page-authoring.html>

 › You can download the latest release of the We.Retail site using the following link:

<https://github.com/Adobe-Marketing-Cloud/aem-sample-we-retail/releases>

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



Introduction to Content Rendering

Introduction

Adobe Experience Manager is a Content Management System that helps organizations build custom websites and apps across different touch points. Before working with the features of AEM, you should first understand the key concepts such as Java Content Repository (JCR), base page component, Sling resolution and inheritance process.

Objectives

After completing this course, you should be able to:

- Explain JCR nodes, properties, and namespaces
- Explain JCR folder structure
- Create a project folder structure in JCR
- Create a base page component
- Create content to render on the page
- Explain the Sling resolution process
- Search for a rendering script
- Manipulate selectors
- Explain how Sling inheritance works
- Implement Sling inheritance through resourceType
- Modularize content by including other scripts

Java Content Repository Nodes and Properties

The JCR has a hierarchical tree structure with two items, nodes and properties.

Nodes

Nodes provide the structure and properties to store the data. The nodes of JCR:

- Can have parent and child nodes that are represented with paths.
- Have a type that sets rules about the kinds of properties and child nodes a node can or must have.
For example, a node of type cq:page must have a child node jcr:content of type cq:PageContent.
- Can have any number of properties.
- Can have zero to more mixin types that specify additional properties and child nodes a node must or may have. The common mixin types include mix:versionable and mix:referenceable.

Properties

The properties of a node:

- Have a name and value.
- Have a single or multi-valued property. A multi-valued property is notated and behaves like an array (values []).

parent node

└→ this node



Namespaces

All nodes and properties of JCR are stored in different namespaces. The common namespaces are:

- jcr: Basic data storage (part of jcr spec)
- nt: Foundation node types (part of jcr spec)
- rep: Repository internals (part of jcr spec)
- mix: Standard mixin node types (part of jcr spec)
- sling: Added by Sling framework
- cq: Added by the AEM application

The following table describes the common nt node types used in AEM:

Node Type	Description
nt:file	Represents a file in a filesystem.
nt:folder	Represents a folder in a filesystem
nt:unstructured	Allows any combination of child nodes and properties. It also supports client-orderable child nodes and stores unstructured content and commonly used (for touch UI) dialog boxes.

The following table describes the common AEM node types:

Node Type	Description
cq:Page	Stores the content and properties for a page in a website
cq:Template	Defines a template used to create pages
cq:ClientLibraryFolder	Defines a library of client-side JavaScript CSS
cq>EditConfig	Defines the editing configuration for a component including drag and drop and in-place editing.
cq:InplaceEditingConfig	Defines an in-place editing configuration for a component. It is a child of cq>EditConfig

Folder Structure of Java Content Repository

You can view the folder structure of JCR from CRXDE Lite. The following table describes the folder structure within the repository:

Folder	Description
/apps	Contains all project code such as components, overlays, client libraries, bundles, i18n translations, and static templates created by an organization
/conf	Contains all configurations for your website. This folder is used to store the dynamic templates and policies for your website.
/content	Contains content created for your website
/etc	Contains resources related to utilities and tools.
/home	Contains AEM users and group information
/libs	Contains the libraries and definitions that belong to the core of AEM. The subfolders in /libs represent the out-of-the-box AEM features.
/oak:index	Contains Jackrabbit Oak index definitions. Each node specifies the details of one index. The standard indexes for the AEM application are visible and help create additional custom indexes.
/system	Is used by Apache Oak only
/tmp	Serves as a temporary working area
/var	Contains files that change and are updated by the system; such as audit logs, statistics, and event-handling. The subfolder /var/classes contains the Java servlets in source and compiled forms that have been generated from the components scripts.

 Caution: You may need to change the JCR folder structure during website development. However, you should fully understand the implications of any changes you make. You must not change the /libs path. For configuration and other changes, copy the item from /libs to /apps and make any changes within /apps.

Exercise 1: Create a project structure in JCR

To create a project structure in JCR:

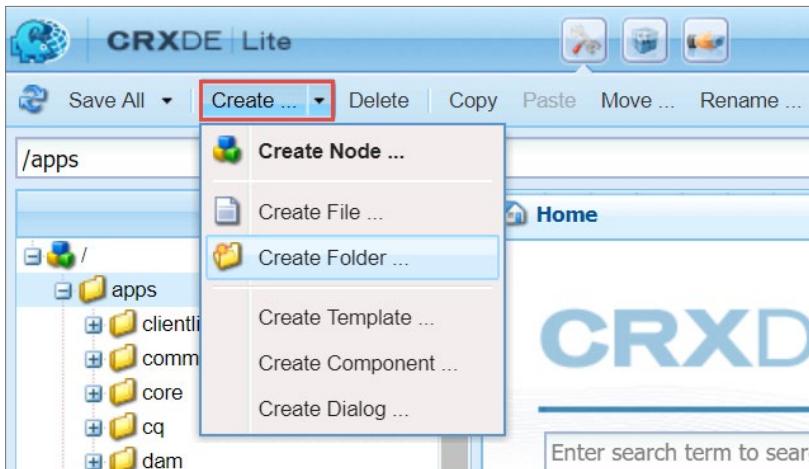
1. In the AEM author instance, click the **Adobe Experience Manager** icon, and then click the Tools icon. The Tools console opens.
2. Click **CRXDE Lite** or type the <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens as shown:

The screenshot shows the CRXDE Lite interface. The left sidebar displays the JCR root node structure with nodes like apps, bin, conf, content, etc., expanded. The main panel shows the 'Home' page with the title 'CRXDE | Lite' and a search bar. Below the title is a table of properties for the root node. The table has columns: Name, Type, Value, Protected, Mandatory, and Multi. The properties listed are:

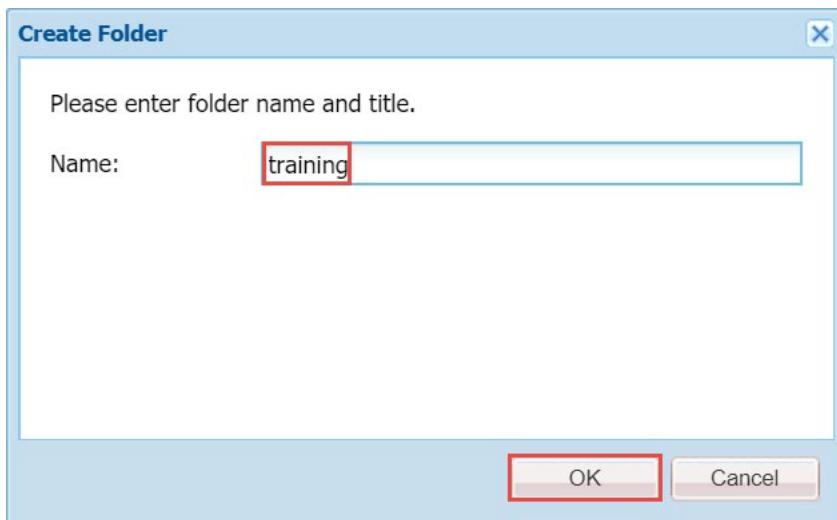
Name	Type	Value	Protected	Mandatory	Multi
1 jcr:mixinTypes	Name[]	rep:... true	false	true	
2 jcr:primaryType	Name	rep:... true	true	false	
3 sling:resourceType	String	sling... false	false	false	
4 sling:target	String	/ind... false	false	false	

At the bottom of the properties table, there is a form to add new properties: Name (text input), Type (dropdown: String), Value (text input), Multi (checkbox), and Add (button).

3. Select the **/apps** folder, click **Create** from the actions bar, and then **Create Folder** from the drop-down menu as shown. The **Create Folder** dialog box opens.



4. Enter **training** in the **Name** field, and click **OK** as shown. The training folder is created under the **/apps** folder.



 **Note:** Node names are a restricted set. In node names, ensure there are no whitespaces or special characters and use the lower-case letters. Do not use an "_" (underscore) character in node names. Instead, use hyphens ("") if you must separate two words or phrases in the name of a node.

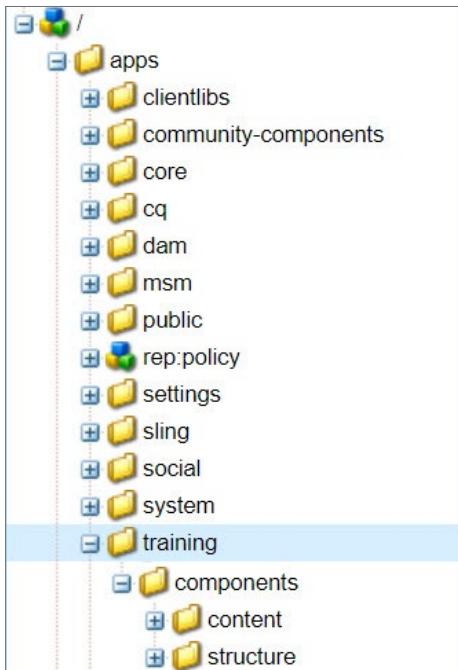
5. Click **Save All** in the upper left of the actions bar as shown:



6. Navigate to **/apps**, select the **training** folder, click **Create** from the actions bar, and then select **Folder** from the drop-down menu. The **Create Folder** dialog box opens.
7. Enter **components** in the **Name** field, and click **OK**. The components folder is created under the training folder
8. Click **Save All**.

 **Note:** You must click **Save All** or use the keyboard shortcuts Ctrl+S (Windows) and Cmd+S (Mac) to save any and all changes in CRXDE Lite. You must do this every time you make a change.

9. Navigate to the **/apps/training** folder, select the **components** folder, create two child folders named **content** and **structure** and click **Save All**. Your project structure should look similar to the given in the below screenshot:



Create a Base Page Component

AEM has a set of node types to make content rendering more powerful and flexible. To store content, cq:Page is used and to store the script logic, cq:Component nodes are used. The cq:Component nodes are containers for rendering scripts. They help a developer add interfaces for content authoring called dialogs through a JCR node-based configuration method.

A page component:

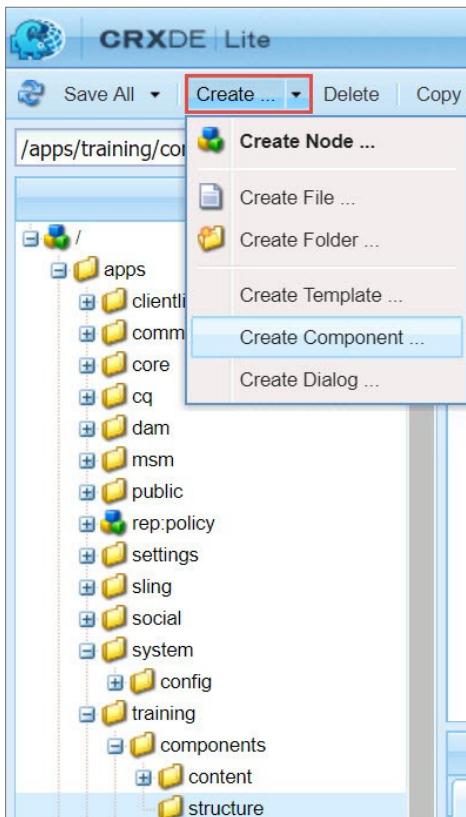
- Is a resourcetype.
- Is a modular and reusable unit that implements a specific functionality or logic to render the content of your website.
- Contains a collection of scripts (for example, HTL files, JSPs, and Java servlets) that completely realize a specific function.

A page component is the beginning of the script rendering process for a Page. Typically, the page component inherits from the Core Page component by using a `sling:resourceSuperType` property. This property provides a single container to control all the pages of a website. All templates will use this single page component.

Exercise 2: Create a base page component

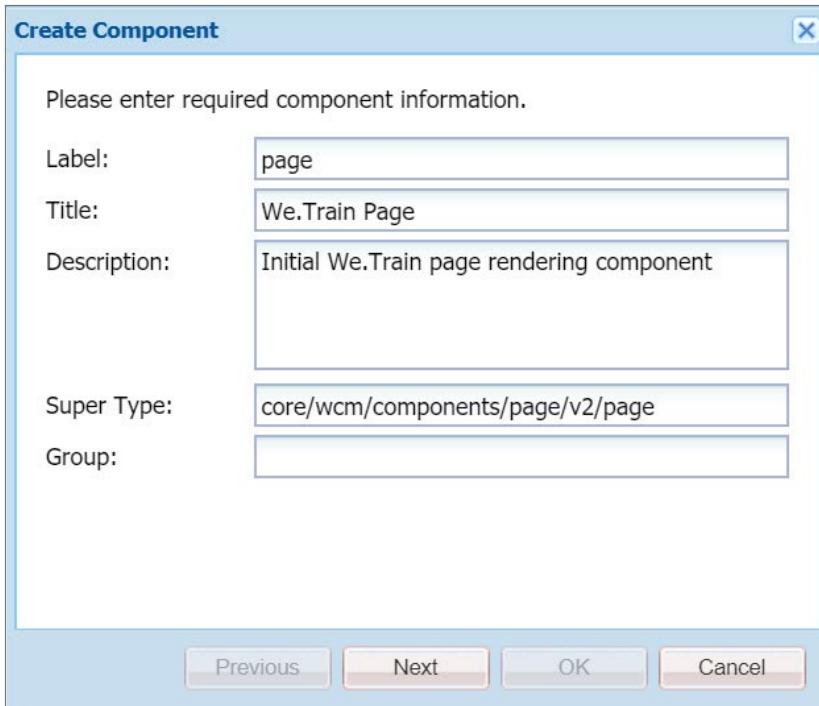
To create a base page component:

1. In CRXDE Lite, navigate to the `/apps/training/components` folder, select the **structure** folder, click **Create** from the actions bar, and then click **Create Component** from the drop-down menu as shown. The **Create Component** dialog box opens.

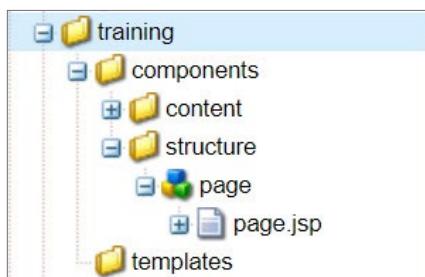


2. In the **Create Component** dialog box, enter the following values as shown:

Field	Value
Label	page
Title	We.Train Page
Description	Initial We.Train page rendering component
SuperType	core/wcm/components/page/v2/page

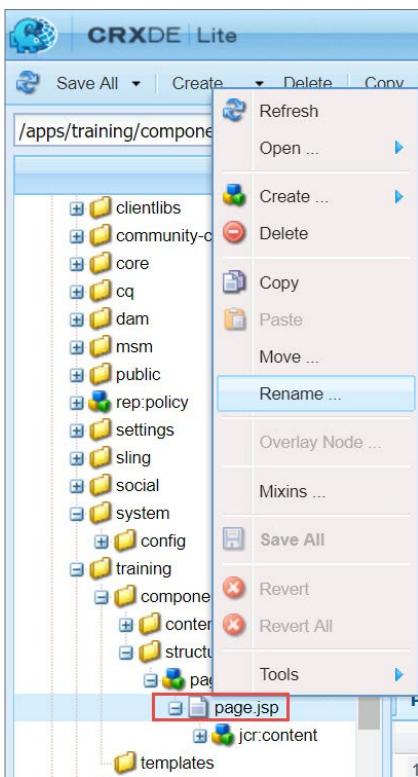


3. Click **Next**, click **OK**, and then click **Save All**. Your component should look similar to the one given in the below screenshot:



 **Note:** You need to click **Save All** or use the keyboard shortcuts Ctrl+S (Windows) and Cmd+S (Mac) to save any and all changes in CRXDE Lite. You must do this every time you make a change.

4. Navigate to the **training/structure/page** node, click the + (plus) sign to expand the page node, right-click the **page.jsp**, and then click **Rename** as shown. The field name becomes editable.



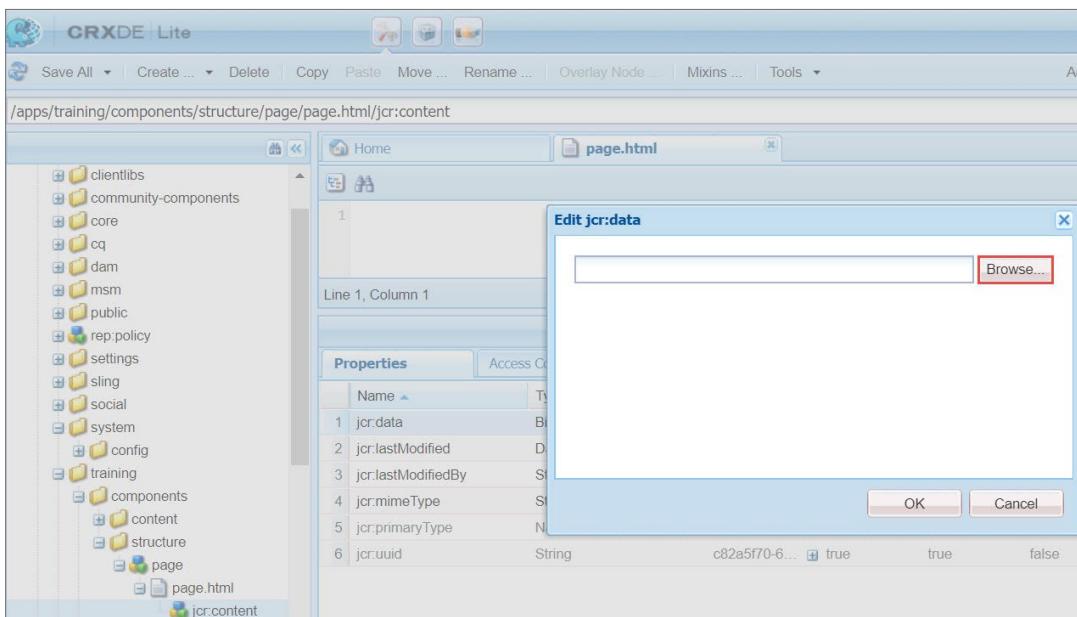
5. Rename the **page.jsp** to **page.html**. Renaming the file helps populate the page content with HTML.
6. Click **Save All**.
7. Double-click the **page.html** to open the script for editing.

You can replace HTML, JS, and any other code in CRXDE Lite by using CRXDE Lite's method of populating code in the **jcr:data** node or by opening code files from the extracted **Exercise_Files** folder in Notepad++.

*Option A: If you are using CRXDE Lite's method of populating code in the **jcr:data** node:*

8. Select the **jcr:content** node that is below **page.html**. The **Properties** tab opens.
9. Double-click the **jcr:data** property. The **Edit jcr:data** dialog box opens.

10. Click **Browse** as shown. The **Open** dialog box opens.



 **Note:** Your instructor will provide you a **Exercise_Files.zip** folder. Please ensure to save the folder in your file system and unzip it.

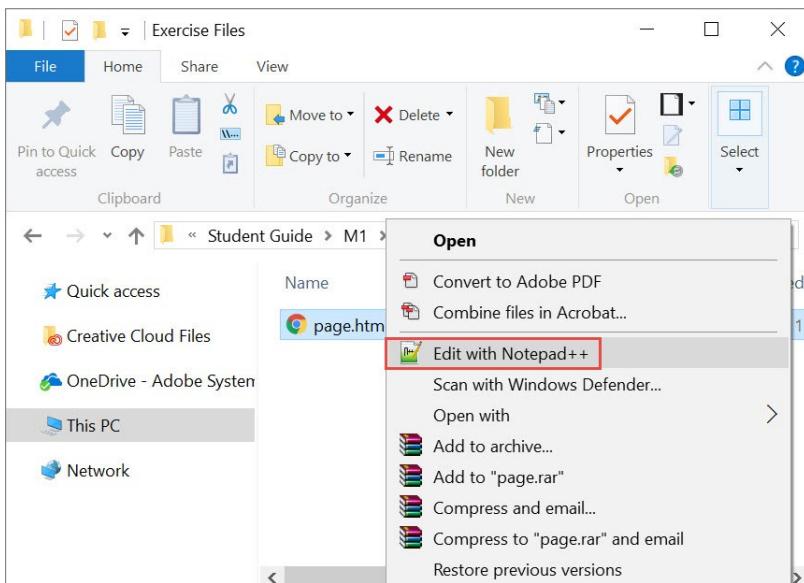
11. Navigate to the **Exercise_Files** folder on your file system.
12. Select **page.html**, and then click **Open**. The file is uploaded.
13. Click **OK** in the **Edit jcr:data** dialog box. The default sample code is replaced with the code in **page.html**.
14. Click **Save All**.
15. Open the **page.html** to view the new code that you added.

```
01. <h1>My Page Script</h1>
```

You can use this method for all situations when you create a new file, or replace the contents of files. If you are creating a new blank file, remember to first save the file and use the method of double-clicking on the **jcr:data** property.

Option B: If you are using the method of opening the code files from the extracted Exercise_Files folder in Notepad++:

16. Navigate to the **Exercise_Files** folder on your file system. Double-click the folder to open it, select the **page.html**, right-click and select **Open with Notepad++** as shown. The page opens in Notepad++.



17. Copy the code from **page.html** of Notepad++.
18. Navigate to the **CRXDE Lite** tab of the browser, paste the code in **page.html**, and then click **Save All**. The sample code is replaced with the new code from **page.html**.



Note: The recommended approach for replacing the sample code in the course is using the CRXDE Lite's method (*Option A*).

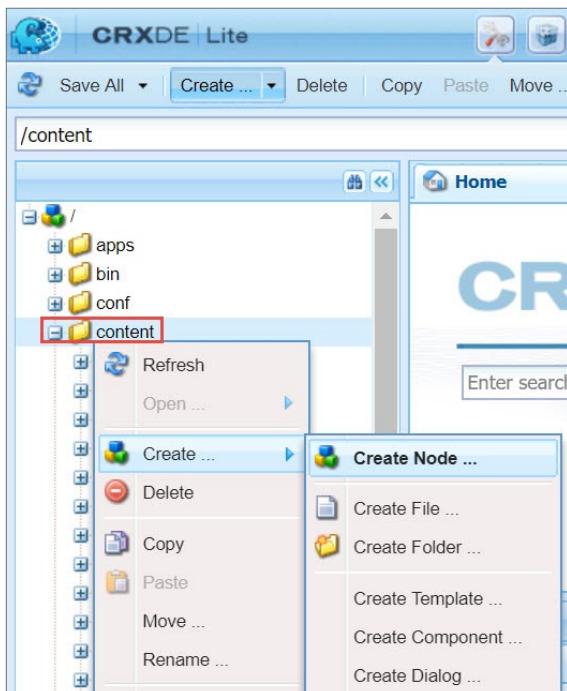
Exercise 3: Create content to render on the page

In this exercise, you will create a content node that will render the page component. This teaches the basics of rendering the code (the component) with content. Creating content through CRXDE Lite is not typical though. Typically, authors will create the content using the AEM Sites console.

1. From CRXDE Lite, navigate to the **/content** folder.

 **Note:** Select the **/content** folder that is at the root, not within your **/training** project structure.

2. Right-click the **/content** folder, click **Create**, and select **Create Node** from the drop-down menu as shown. The **Create Node** dialog box opens.



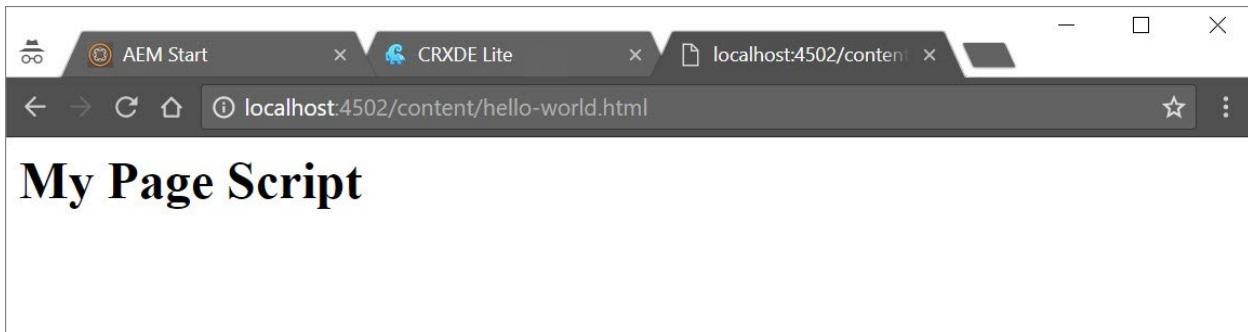
3. Enter the following:
 - a. **Name:** hello-world
 - b. **Type:** Ensure **nt:unstructured** is selected.
4. Click **OK**. The node is created.
5. Click **Save All**.
6. Select the **hello-world** node and ensure the **Properties** tab is open on the left-side panel.
7. Add the following property to the hello-world node:

Name	Type	Value
sling:resourceType	String	training/components/structure/page

8. Click **Add** as shown. The property is added to the node.

Name	Type	Value	Protected	Mandatory	Multiple	Auto Created
1 jcr:primaryType	Name	nt:unstructured	true	true	false	true

9. Click **Save All**.
10. Open a new tab in your browser, enter <http://localhost:4502/content/hello-world.html> in the address bar, and then press Enter.
11. Observe, how the property `sling:resourceType` renders the content from the page component (`page.html`) on the `hello-world.html` page as shown:



Sling Resolution Process

Apache Sling is resource-oriented and all resources are maintained in the form of a virtual tree. A resource is usually mapped to a JCR node. However, you can also map to a file system or database. The common properties that a resource can have are Path, Name, and Resource Type.

Resource First Request Processing

A request URL is first resolved to a resource, and then based on the resource, Sling selects the servlet or the script to handle the request.

The following table lists the differences between a traditional framework and a Sling framework request processing:

Traditional Web Application Framework	Sling Framework
Selects the servlet or controller based on the request URL.	Places data in the center.
Loads data from the database to render the result.	Uses the request URL to resolve the data to process.

Basic Steps of Processing Requests

Each content item in JCR is exposed as an HTTP resource. After the content is determined, the script or the servlet to be used to handle the request is determined through the following:

- Properties of the content item
- HTTP method used to make the request
- Simple naming convention within the URL that provides secondary information

The following steps are performed to resolve a URL request:

1. Decompose the URL
2. Search for a servlet or a vanity URL redirect
3. Search for a node indicated by the URL
4. Resolve the resource
5. Resolve the rendering script/servlet
6. Create a rendering chain
7. Invoke a rendering chain

Decomposing the URL

Consider the following URL:

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

This URL can be decomposed into the following components:

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

where:

- Protocol: Is the Hypertext transfer protocol (HTTP)
- Host: Is the name of the website
- Content path: Is the path specifying the content to be rendered
- Selector(s): Is used for alternative methods of rendering the content
- Extension: Is the content format that also specifies the script to be used for rendering
- Suffix: Can be used to specify additional information
- Param(s): Are any parameters required for dynamic content

Resolving Requests to Resources

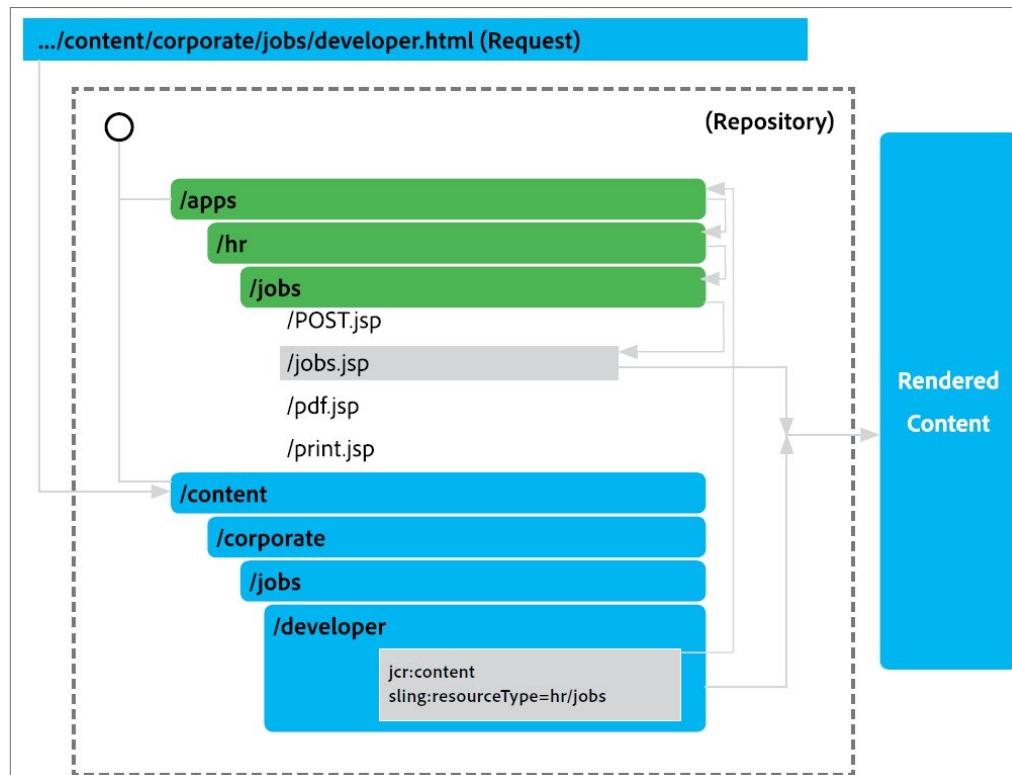
Example 1: URL: <http://myhost/tools/spy.html>

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:

1. The Sling Resource Resolver process first looks for a redirection rule (such as a vanity URL), or a servlet. If the rule/servlet does not exist or is not found, then the script resolution process begins.
2. As part of the script resolution process, Sling searches for the spy node.
3. If a node is found, the sling resource type for that node is extracted, and used to locate the script to be used for rendering the content.
4. If no node is found, Sling will return the http code 404 (Not Found).

Example 2:

Consider the URL request: <http://myhost/content/corporate/jobs/developer.html> and corresponding diagram. Notice how the properties of the developer node are used to provide the rendered content:



Locating and Rendering Scripts

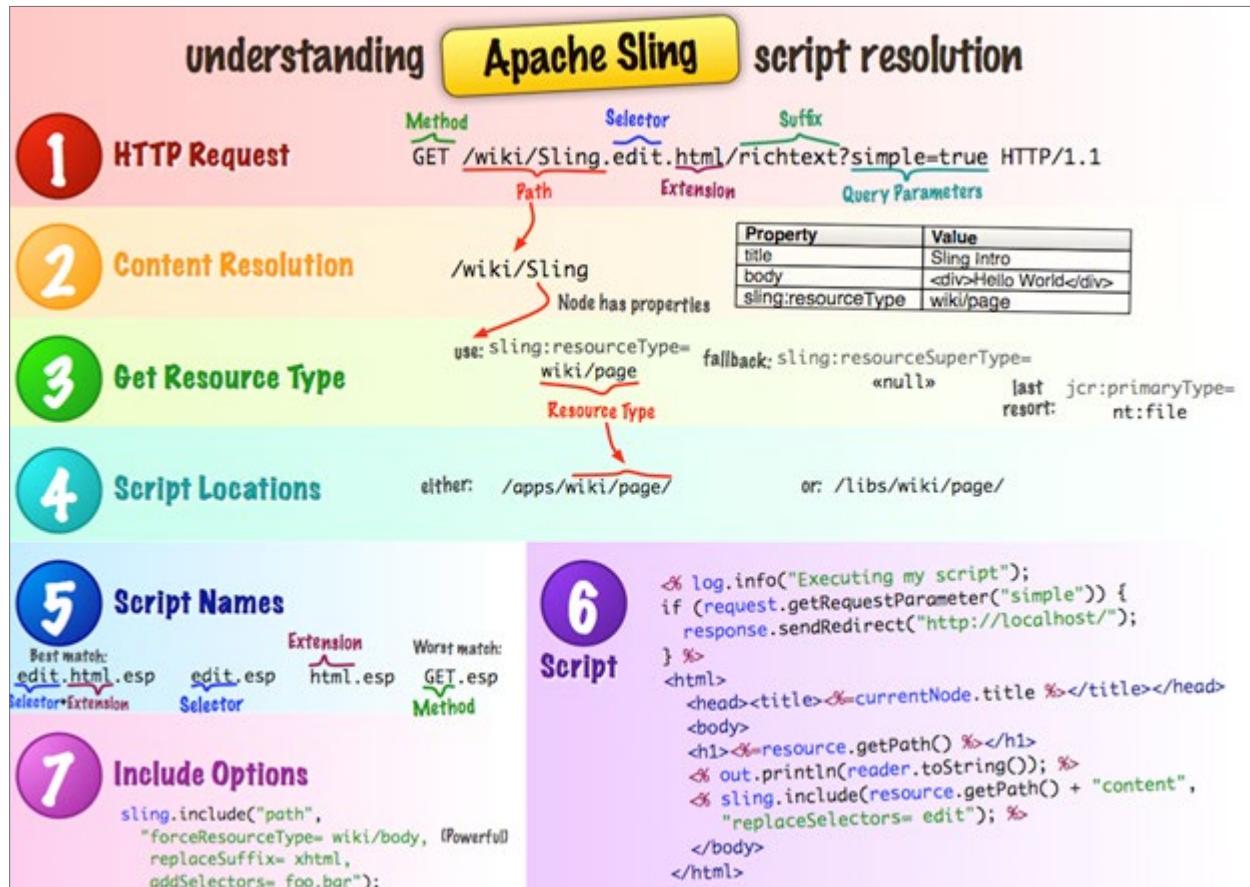
When the resource is identified from the URL, its resource type property is located and the value extracted. This value is either an absolute or a relative path that points to the location of the script to be used for rendering the content. All scripts are stored in either the /apps or /libs folder, and are searched in the same order. If no matching script is found in either of the folders, then the default script is rendered. For multiple matches of the script, the script name with the best match is selected. The more selector matches, the better, as shown in the below screenshot:

<p><u>Files in repository under hr/jobs</u></p> <ul style="list-style-type: none">1. GET.jsp2. jobs.jsp3. html.jsp4. print.jsp5. print.html.jsp6. print/a4.jsp7. print/a4/html.jsp8. print/a4.html.jsp	<p><u>REQUEST</u> URL: /content/corporate/jobs/developer.print.a4.html sling:resourceType = hr/jobs</p> <p><u>RESULT</u> Order of preference: 8 > 7 > 6 > 5 > 4 > 3 > 2 > 1</p>
---	--

URL Decomposition

Each content item in the JCR is exposed as an HTTP resource, and the request URL addresses the data to be processed, not the procedure that does the processing. After the content is determined, the script or servlet to be used to handle the request is determined in cascading resolution that checks.

In chronological order, the Sling Resource Resolver first tries to resolve the URL to a servlet or redirect rule. If it does not exist or is not successful, the script resolution process described in the below screenshot takes place. If no node is found, a 404 error is returned.



References

You can use the following links for more information on:

- Script Resolution:

<https://sling.apache.org/documentation/the-sling-engine/url-to-script-resolution.html>

- URL Decomposition:

<https://sling.apache.org/documentation/the-sling-engine/url-decomposition.html>

- Sling Cheatsheet:

<https://helpx.adobe.com/experience-manager/6-3/sites/developing/using/sling-cheatsheet.html>

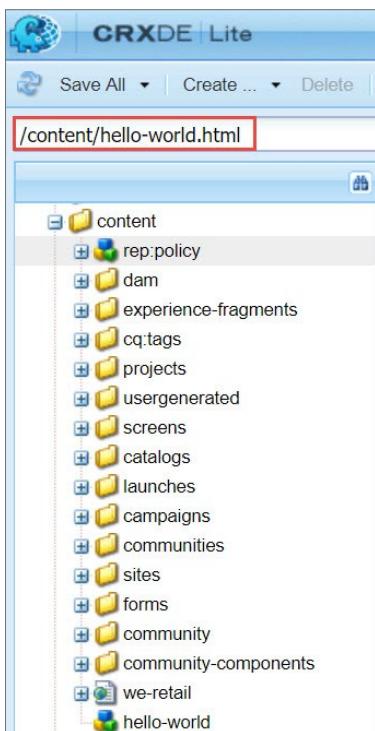
Exercise 4: Search for a rendering script

When presented with a request (URL), Sling will take the following actions:

1. Disregard the protocol, server, port, and any suffix from the URL.
2. Examine the remaining portion of the URL, use the information contained therein to assist with resource solution and find the associated rendering script.

Let's follow the Sling's actions to see how the hello-world resource was rendered in the previous exercise.

1. Consider the request (URL) <http://localhost:4502/content/hello-world.html>
2. Disregard **http://localhost:4502**.
3. Ensure the **CRXDE Lite** page is open. If not, add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
4. Navigate to **/content/hello-world.html**. Notice that the path does not exist in the repository as shown:

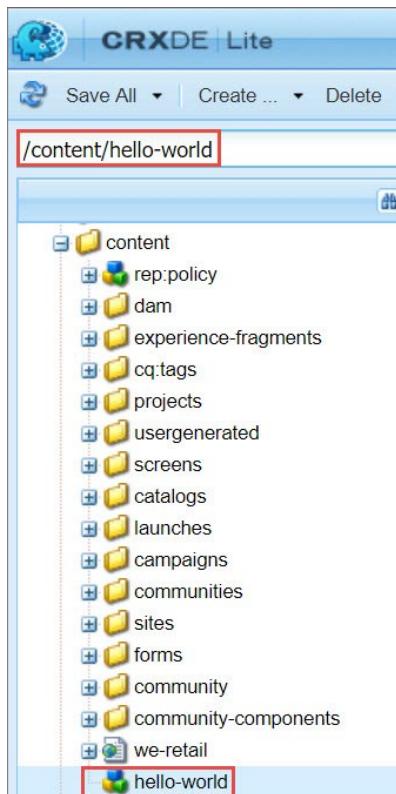


- At this point, Sling will back off the remaining part of the request until the first ".", and everything after the "." is considered the extension as shown:

~~http://localhost:4502/content/hello-world.html~~

- In CRXDE Lite, navigate to the /content/hello-world node as shown.

You found a node in the repository that matches the resource in the request. What you just did is called *resolving the resource*. You have successfully resolved a request URL to a resource that is represented by a node in the repository as shown:



Note: You can map resources to the items in the relational databases and/or the file system. The Apache Sling specification does not mandate a JCR database.

Now, you need to find the rendering script.

7. Navigate to the `/content/hello-world` node, find the **sling:resourceType** property. The value of the **sling:resourceType** property is what Sling uses to begin its search for a rendering script.
8. Notice that the **sling:resourceType** property points to the page-rendering script that you created previously as shown:

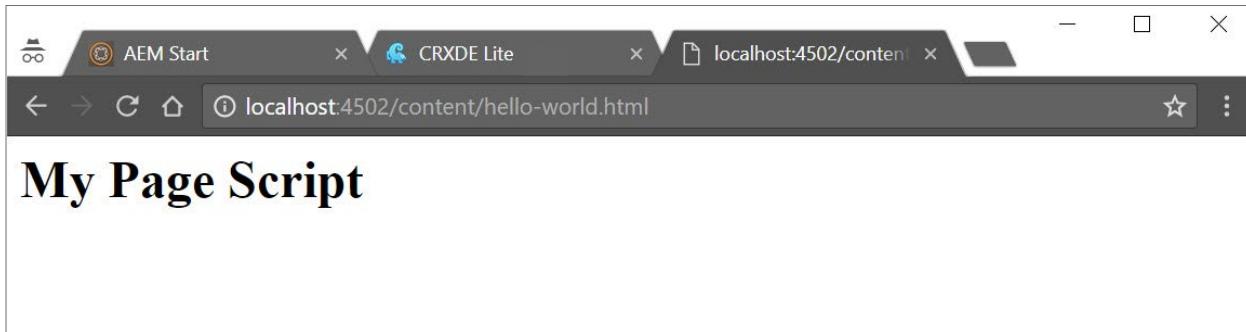
Properties		
Name	Type	Value
1 jcr:primaryType	Name	nt:unstructured
2 sling:resourceType	String	training/components/structure/page

9. Navigate to the `/apps/training/components/structure/page` node.
10. Notice there is one script, `page.html`. This is called the default script because the script's name matches the name of the folder `/component` in which the script resides. The script outputs **My Page Script** as shown:

Properties		
Name	Type	Value
1 jcr:created	Date	2018-03-02T17:50:02
2 jcr:createdBy	String	admin

Later, you will explore many options that Sling might use in selecting the *right* script. For this exercise, a default script is available. Given the specified request and the available selection of scripts, the default script is the best match and Sling chooses it to render the resource.

You have just seen how the request <http://localhost:4502/content/hello-world.html> results in the following browser output:



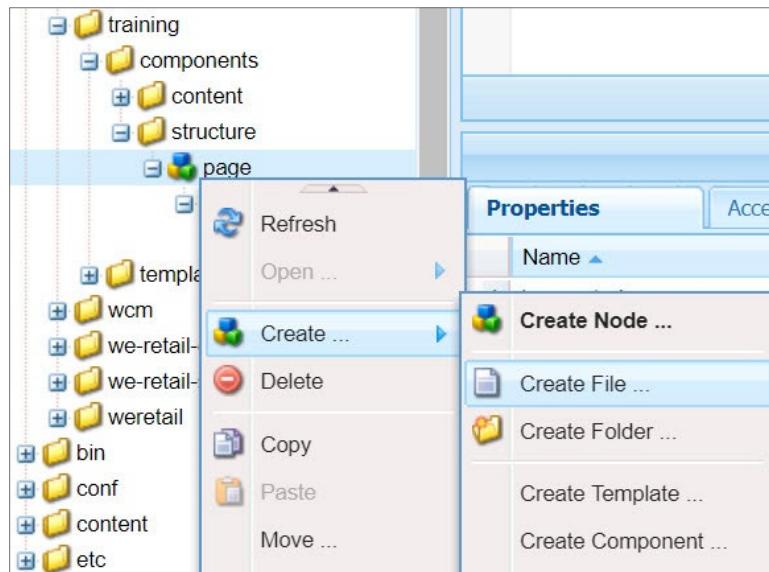
In this exercise, you have successfully resolved the resource, found the rendering script, and invoked the rendering chain.

Exercise 5: Manipulate selectors

A component can render the content in several rendering variations. Each variation is described in its own script file. The selectors provide a way to choose the variation of the script that should be rendered.

In this exercise, you will learn how to manipulate Sling to choose the script that you want to render.

1. Ensure the **CRXDE Lite** page is open. If not, type <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components/structure/page** node.
3. Select the **page** node, right-click it, select **Create**, and then **Create File** from the drop-down menu as shown. The **Create File** dialog box opens.



4. Enter **blue.html** in the **Name** field, and then click **OK**. The blue.html page opens in the left side panel.
5. Click **Save All**.

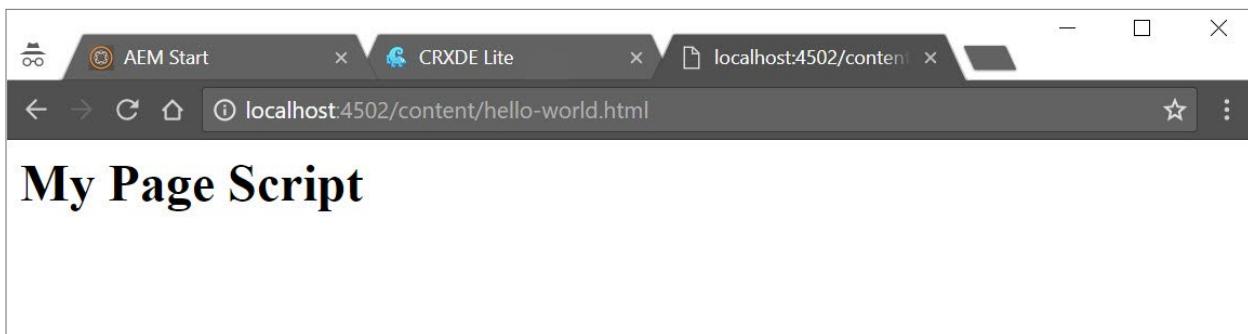
To add code to the blue.html page:

6. Use *Option A: CRXDE Lite's method* to add code to the blue.html page (perform steps 8 through 14 of Exercise 2).
7. Click **Save All**.
8. Open the blue.html file and observe the code:

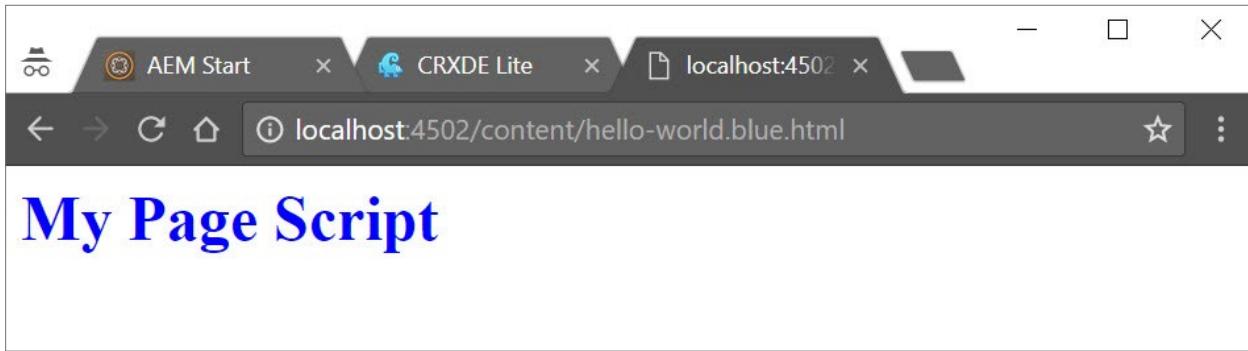
```
01. <h1 style="color:blue">My Page Script</h1>
```

To test the selector:

9. Open a new tab of your browser, type the URL <http://localhost:4502/content/hello-world.html> in the address bar, and then press Enter. The hello-world node is rendered as shown:



10. Add blue to the URL as shown: <http://localhost:4502/content/hello-world.blue.html>. Notice the difference, the code from blue.html is chosen as the script name matches the selector in the URL as shown:



Inheritance

A cq:Component has various capabilities such as selectors and inheritance mechanisms. Components can be given a hierarchical structure to implement the inheritance of script files and dialog boxes. Therefore, it is possible for a specific *page* component (or any component) to inherit from a *base* component. For example, allowing inheritance of a script file for a specific part of the page by using the `<head>` tag.

The components within AEM are subject to the following different hierarchies:

- Resource Type
- Container
- Include

Resource Type Hierarchy

The resource type hierarchy is used to extend components using the property `sling:resourceSuperType`. property. This enables the component to inherit from a core component. For example, a text component will inherit various attributes from the core text component, including:

- Scripts (resolved by Sling)
- Dialog boxes
- Descriptions (including thumbnail images and icons)

It is important to note that a local copy or instance of a component element will take precedence over an inherited element.

Container Hierarchy

The container hierarchy is used to populate configuration settings to the child component and is most commonly used in a responsive grid scenario. For example, you can define the configuration settings for the edit bar buttons, control set layout (edit bars, rollovers, and so on), and dialog box layout (inline, floating, and so on) on the parent component , which are then propagate to the child components. The configuration settings (related to edit functionality) in cq:editConfig and cq:childEditConfig are propagated.

Include Hierarchy

The include hierarchy is imposed at runtime by the sequence of includes. It can be used to separate logic within a component further for a more modularized approach. Breaking out a component's logic into multiple scripts and rendering them with a series of includes allows for a component to be easily extended and customized.

Exercise 6: Inheritance with resourceSuperType

When you defined the `/apps/training/components/structure/page` component, the Core page component was declared as the **superType** of page-rendering component. This helped the page component to initialize the Web Content Management (WCM) to take full advantage of the authoring environment. By declaring the Core page component as its superType, the page component will also inherit and make use of properties, scripts, and other features of the Core page component.

To view the **superType** of the page-rendering component:

1. Ensure the **CRXDE Lite** page is open. If not, add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the `/apps/training/components/structure/page` node. Notice the **sling:resourceSuperType** property as shown:

Name	Type	Value
1 jcr:created	Date	2018-03-02T17:50:02.275+05:30
2 jcr:createdBy	String	admin
3 jcr:description	String	Initial We.Train page rendering component
4 jcr:primaryType	Name	cq:Component
5 jcr:title	String	We.Train Page
6 sling:resourceSuperType	String	core/wcm/components/page/v2/page

In `core/wcm/components/page/v2/page`, a lot of functions are implemented. You can inherit that functionality from the core page component by setting a **sling:resourceSuperType** property to the page component.

Let's create a new component and add a superType property.

To create a new component:

3. In CRXDE Lite, navigate to the **/apps/training/components** folder, select the **structure** folder, click **Create** from the actions bar, and then click **Create Node** from the drop-down menu. The **Create Node** dialog box opens.
4. Enter **inherited-page** in the **Name** field, select **cq:Component** from the **Type** drop-down menu, and then click **OK**. The node is created.
5. Click **Save All**.
6. Add the following properties to the **inherited-page** node:

Name	Type	Value
jcr:title	String	Inherited Page
sling:resourceSuperType	String	training/components/structure/page

7. Click **Save All**.

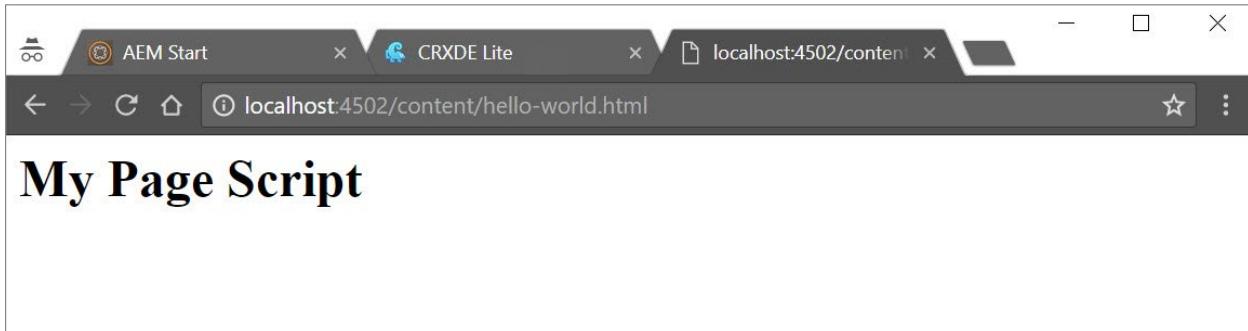
Let's update the **sling:resourceType** property of the **hello-world** content node and observe the changes:

8. Navigate to the **/content/hello-world** node.
9. Select the **hello-world** node, from the **Properties** panel, select the **sling:resourceType** property, double-click the **Value** field and add **training/components/structure/inherited-page** as the new value as shown:

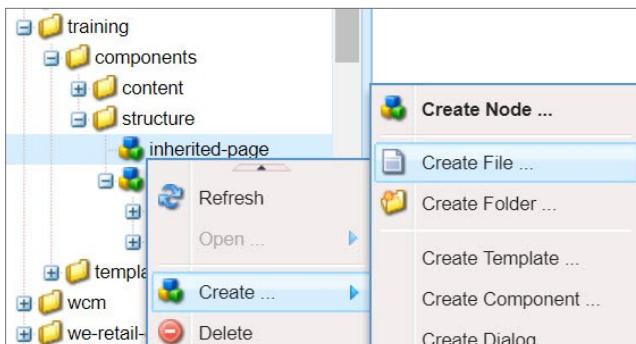
The screenshot shows the CRXDE Lite interface with the left sidebar containing nodes like campaigns, communities, sites, forms, community, community-components, we-retail, and hello-world. The hello-world node is selected. The right panel has tabs for Properties, Access Control, Replication, Console, and Build Info. The Properties tab is active, showing two properties: jcr:primaryType (Name, nt:unstructured) and sling:resourceType (String, currently empty). A red box highlights the sling:resourceType input field, which is being edited with the value "training/components/structure/inherited-page".

10. Click **Save All**. The changes you made will be saved.

11. Open a new tab of your browser, add the URL <http://localhost:4502/content/hello-world.html> in the address bar, and then press Enter. Notice, how the resource (hello-world) calls the inherited page component because it has no scripts, it looks at the property **sling:resourceSuperType** and calls the page component. It then renders the default script you created earlier.



12. Click the **CRXDE Lite** tab of your browser.
13. From the **/apps/training/components/structure** folder, select the **inherited-page** node, right-click the node, click **Create** and select **Create File** from the drop-down menu, as shown. The **Create File** dialog box opens.



14. Enter **inherited-page.html** in the **Name** field, and click **OK**.

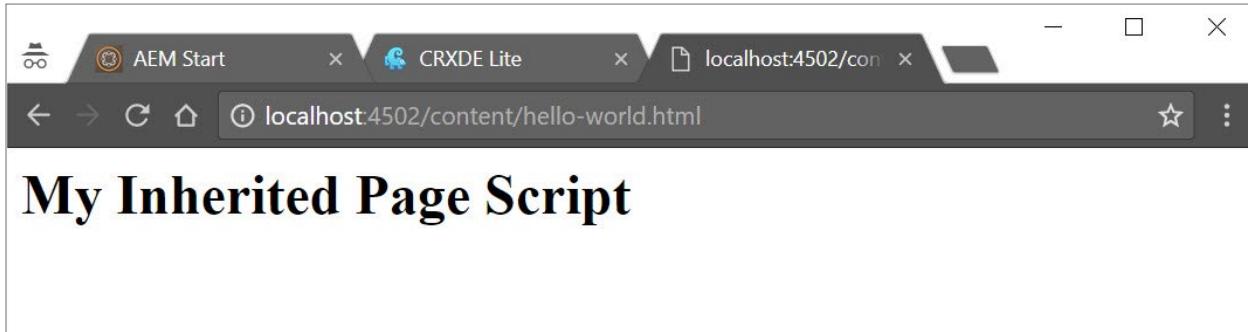
15. Click **Save All**.

To add code to the `inherited-page.html` page:

16. Use *Option A: CRXDE Lite's method* to add code to the `inherited-page.html` (perform steps 8 through 14 of Exercise 2).
17. Click **Save All**.
18. Open the `inherited-page.html` file and observe the code:

```
01. <h1>My Inherited Page Script</h1>
```

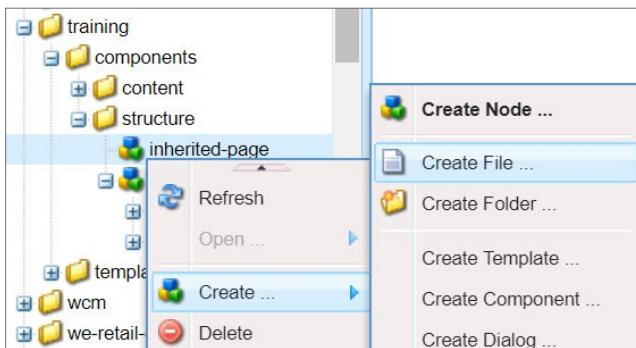
19. Reload the <http://localhost:4502/content/hello-world.html> tab of the browser. Notice, how the inherited-page.html script is being used because it is the local script and the resourceSuperType is no longer needed for rendering.



Exercise 7: Include other scripts

In this exercise, you will learn how to modularize our components by adding local scripts to the component.

1. In CRXDE Lite, navigate to the `/apps/training/components/structure` folder, select the **inherited-page** component, right-click the node, click **Create** and select **Create File** from the drop-down menu as shown. The **Create File** dialog box opens.



2. Enter **red.html** in the **Name** field, and click **OK**.
3. Click **Save All**.

To add code to the red.html page:

4. Use *Option A: CRXDE Lite's method* to add code to the red.html page (perform steps 8 through 14 of Exercise 2).
5. Click **Save All**.
6. Open the red.html file and observe the code:

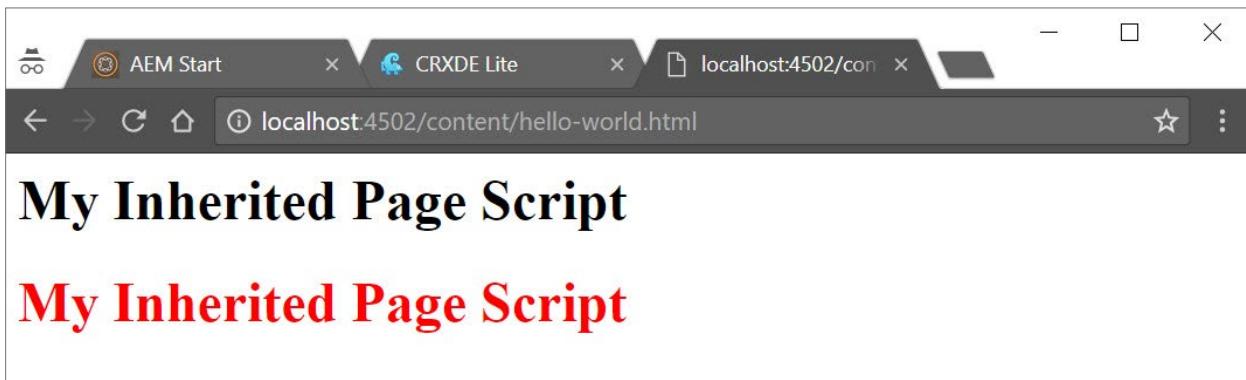
```
01. | <h1 style="color:red">My Inherited Page Script</h1>
```

Let's update the inherited-page.html with new code that includes the red.html page.

7. Update the **inherited-page.html** script by replacing the existing code with new code (2_inherited-page.html) from the **Exercise_Files** folder.
8. Click **Save All**.
9. Open the inherited-page.html file and observe the code:

```
01. <h1>My Inherited Page Script</h1>
02. <sly data-sly-include="red.html" />
```

10. Reload the <http://localhost:4502/content/hello-world.html> tab of the browser. Notice, how you are able to include local scripts from the component as shown. This helps modularize our component.

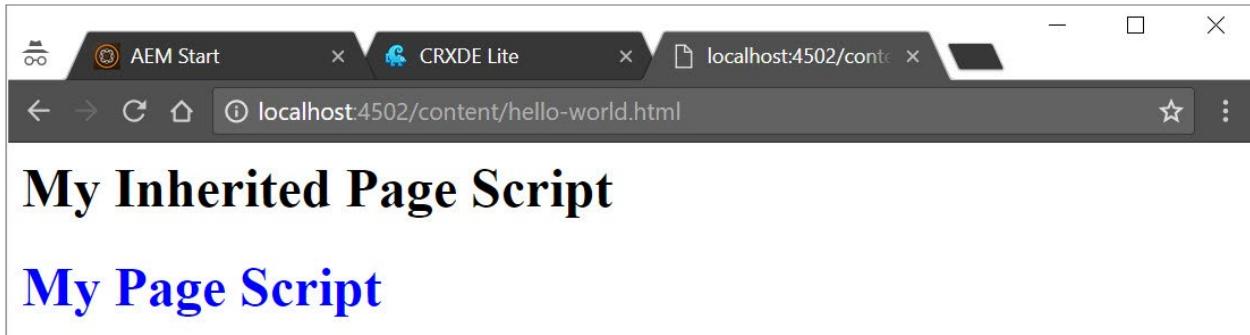


Let's update the inherited-page.html with new code that includes red.html page and observe the changes.

11. Update the **inherited-page.html** script by replacing the existing code with new code (3_inherited-page.html) from the **Exercise_Files** folder.
12. Click **Save All**.
13. Open the inherited-page.html file and observe the code:

```
01. <h1>My Inherited Page Script</h1>
02. <sly data-sly-include="blue.html" />
```

14. Reload the <http://localhost:4502/content/hello-world.html> tab of the browser. Notice, the blue.html is rendered by Sling using the **resourceSuperType** property as shown:





Introduction to HTML Template Language

Introduction

HTML Template Language (HTL) is developed and supported by Adobe to replace Java Server Pages (JSP). HTL offers a highly productive enterprise-level web framework that increases security and helps HTML developers without Java knowledge to work on AEM projects easily.

Objectives

After completing this module, you should be able to:

- Explain HTL
- Explain the goals of HTL
- Explain HTL syntax
- Render page content using AEM global objects
- Render content values on a page
- Render page content by using HTL attributes

HTL

HTML is the recommended language for developing components in AEM. The AEM reference site, We.Retail that is available out-of-the-box is built by using HTL.

An HTL template defines an HTML output stream by specifying the presentation logic and the values that need to be inserted into the stream dynamically based on the background business logic.

HTL differs from other templating systems in the following ways:

- HTL is HTML5: A template created in HTL is a valid HTML5 file. All HTL-specific syntax is expressed within a data attribute or within HTML text. Any HTL file opened as HTML in an editor will automatically benefit from features such as auto-completion and syntax highlighting that are provided by an editor for regular HTML.
- Separation of concerns (web designer versus web developer): The expressiveness of the HTL markup language is purposely limited. Only simple presentation logic can be embedded in the actual markup. All complex business logic must be placed in an external helper class. The HTL's Use API defines the structure of the external helper.
- Secure by default: HTL automatically filters and escapes all text being output to the presentation layer to prevent cross-site-scripting vulnerabilities.
- Compatibility: Compatible with JSP or ECMAScript Pages (ESP).

Goals of HTL

The main goals of HTL are the following:

- Provide increased security through automated and context-sensitive cross-site scripting protection
- Simplify development by helping HTML developers to write more intuitive and efficient code
- Reduce cost through reduced effort, faster Time To Market (TTM), and lower Total Cost of Ownership (TCO)

HTL Syntax

HTL uses an expression language to insert pieces of content into the rendered markup, and HTML5 data attributes to define statements over blocks of markup such as conditions or iterations. As HTL is compiled into Java Servlets, the expressions and the HTL data attributes are both evaluated entirely server-side, and nothing remains visible in the resulting HTML.

Blocks and Expressions

HTL uses the following kinds of syntaxes:

- **Block Statements:** To define structural elements within the template, HTL employs the HTML data attribute. The data attribute is HTML5 attribute syntax intended for custom use by third-party applications. All HTL-specific attributes are prefixed with `data-sly-`.
- **Expression Language:** HTL expressions are delimited by characters `${ }` . At runtime, these expressions are evaluated and their value is injected into the outgoing HTML stream. They can occur within the HTML text nodes or within the attribute values. In other words, you can include HTL expressions within HTML tags.

A list of a few of the basic HTL statements, expressions, and tags are.

- Comments (HTL comments are HTML comments with additional syntax. They are delimited as shown below):
 - › Example: `<!--/* An HTL Comment */-->`
- Expressions:
 - › Examples: `${true} , ${properties.text}`
- URI Manipulation:
 - › Example: `${'example.com/path/page.html' @ scheme='http'}` (Resulting output: `http://example.com/path/page.html`)
- Enumerable objects:
 - › Examples: `pageProperties, properties, inheritedPageProperties`

- **HTML Block Statements:**

- › **use:** <div data-sly-use.nav="navigation.js">\${nav.foo}</div>
- › **list:** <dl data-sly-list="\${currentPage.listChildren}">
- › **data-sly-include and data-sly-repeat**

- **Special HTML tags:**

- › **Example:** <sly>

Expressions contain literals and variables.

- **Literals can be:**

- › **Boolean:** \${true} \${false}
- › **Strings:** \${'foo'} \${"answer"}
- › **Positive integers:** \${42}
- › **Arrays:** \${[42, true, 'Hello World']}

- **Variables are accessed through:** \${properties.myVar}

References

You can use the following links for more information on:

- HTL resources:

<https://helpx.adobe.com/experience-manager/htl/using/update.html>

<https://helpx.adobe.com/experience-manager/htl/using/overview.html>

<https://helpx.adobe.com/experience-manager/htl/using/getting-started.html>

- HTL Specifications:

<https://github.com/Adobe-Marketing-Cloud/htl-spec>

<https://github.com/Adobe-Marketing-Cloud/htl-spec/blob/master/SPECIFICATION.md>

Exercise 1: Render page content by using AEM global objects

In this exercise, you will use HTL to render the content residing on the page script.

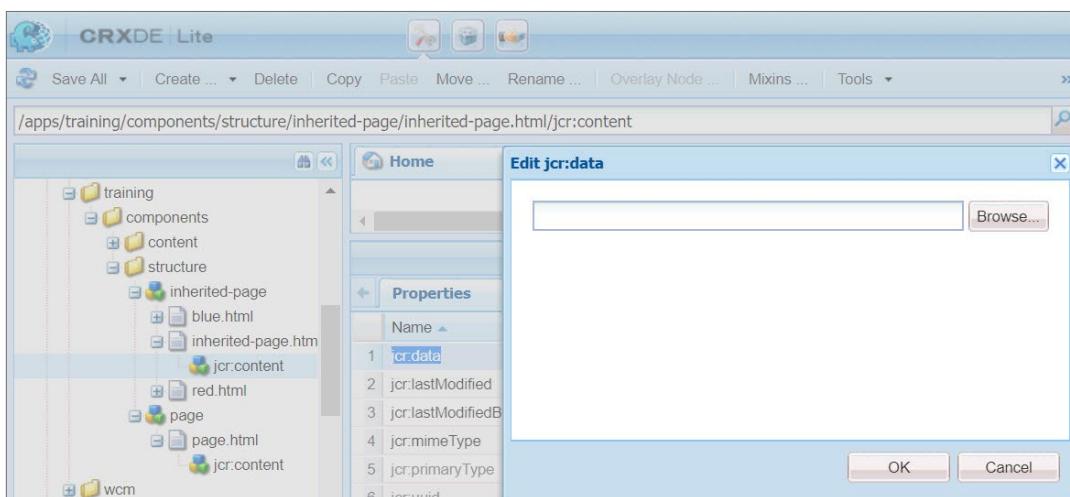
You will install the **adls-training-project-v1.0.zip** package on your AEM author instance. The package contains the training project structure and the page component that was already created for you.

To render the content residing on the page script:

1. If not already open, open CRXDE Lite (<http://localhost:4502/crx/de>).
2. Navigate to the `/apps/training/components/structure/inherited-page` node.
3. Double-click the `inherited-page.html` to open the script for editing.

To update the code on the `inherited-page.html` page:

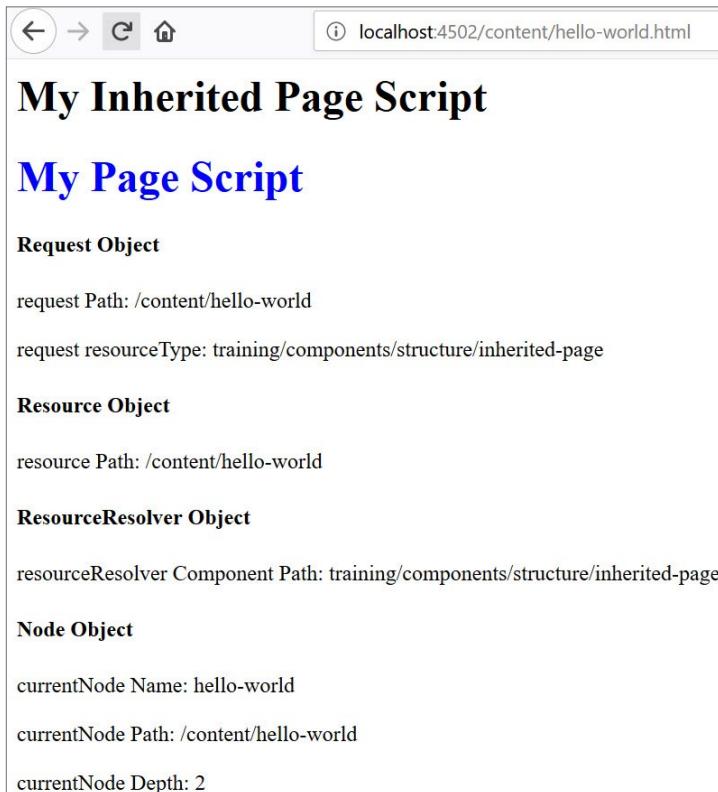
4. Select the `jcr:content` node that is below `inherited-page.html`. The **Properties** tab opens on the left side panel.
5. Double-click the `jcr:data` property. The **Edit jcr:data** dialog box opens.
6. Click **Browse** as shown. The **Open** dialog box opens.



7. Navigate to the **Exercise_Files** folder on your file system.
8. Select **4_inherited-page.html**, and then click **Open**.
9. Click **OK** in the **Edit jcr:data** dialog box. The default sample code is replaced with the code in **page.html**.
10. Click **Save All**.
11. Open the **inherited-page.html** to view the new code that you added:

```
01. <h1>My Inherited Page Script</h1>
02.
03. <sly data-sly-include="blue.html" />
04.
05. <h4> Request Object </h4>
06. <p>request Path: ${request.resource.path}</p>
07. <p>request resourceType: ${resource.resourceType}</p>
08.
09. <h4> Resource Object </h4>
10. <p>resource Path: ${resource.path}</p>
11.
12. <h4> ResourceResolver Object </h4>
13. <p>resourceResolver Component Path: ${resourceResolver.findResource @ path=resource.resourceType}</p>
14.
15. <h4> Node Object </h4>
16. <p>currentNode Name: ${currentNode.Name}</p>
17. <p>currentNode Path: ${currentNode.Path}</p>
18. <p>currentNode Depth: ${currentNode.Depth}</p>
```

12. Open a new tab in your browser, add <http://localhost:4502/content/hello-world.html> in the address bar, and then press Enter. The page displays the rendered content as shown:



13. Examine the rendered content on the page. Notice, the page content is rendered from the property values associated with the page.

Exercise 2: Render content value on the page

In this exercise, you will add a property to the content root and render the content value on the page.

To add a property to the content root:

1. Ensure you are in CRXDE Lite or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the **/content/hello-world** node.
3. Select the **hello-world** node and add the following new property:

Name	Type	Value
myProperty	String	Content is King!

4. Navigate to the **/apps/training/components/structure/inherited-page** node.
5. Double-click the **inherited-page.html** to open the script for editing.
6. Update the **inherited-page.html** script by replacing the existing code with new code (**5_inherited-page.html**) from the **Exercise_Files** folder.
7. Click **Save All**.

8. Open the **inherited-page.html** to view the new code that you added:

```
01. <h1>My Inherited Page Script</h1>
02.
03. <sly data-sly-include="blue.html" />
04.
05. <h3>My Property: ${properties.myProperty} </h3>
06.
07. <h4> Request Object </h4>
08. <p>request Path: ${request.resource.path}</p>
09. <p>request resourceType: ${resource.resourceType}</p>
10.
11. <h4> Resource Object </h4>
12. <p>resource Path: ${resource.path}</p>
13.
14. <h4> ResourceResolver Object </h4>
15. <p>resourceResolver Component Path: ${resourceResolver.findResource @ path=resource.resourceType}</p>
16.
17. <h4> Node Object </h4>
18. <p>currentNode Name: ${currentNode.Name}</p>
19. <p>currentNode Path: ${currentNode.Path}</p>
20. <p>currentNode Depth: ${currentNode.Depth}</p>
```

9. Open a new tab in your browser, add <http://localhost:4502/content/hello-world.html> in the address bar, and then press Enter. Notice, how you can use scripts to render the properties on the resource dynamically as shown:



The screenshot shows a browser window with the URL `localhost:4502/content/hello-world.html` in the address bar. The page content is as follows:

My Inherited Page Script

My Page Script

My Property: Content is King!

Request Object

request Path: /content/hello-world

request resourceType: training/components/structure/inherited-page

Resource Object

resource Path: /content/hello-world

ResourceResolver Object

resourceResolver Component Path: training/components/structure/inherited-page

Node Object

currentNode Name: hello-world

currentNode Path: /content/hello-world

currentNode Depth: 2

Exercise 3: Render page content by using HTL attributes

In this exercise, you use HTL attributes to render page content.

To render the page content by using HTL attributes:

1. Navigate to the [/apps/training/components/structure/inherited-page](#) node.
2. Double-click the **inherited-page.html** to open the script for editing.
3. Update the **inherited-page.html** script by replacing the existing code with new code ([6_inherited-page.html](#)) from the **Exercise_Files** folder.
4. Click **Save All**.
5. Open the **inherited-page.html** to view the new code that you added:

```

18. </p>
19.
20. <h3>Ex 1: Child resources of ${rootPage.name}</h3>
21. <ul data-sly-list=${rootPage.listChildren}>
22.   <li >
23.     ${item.name}
24.   </li>
25. </ul>
26.
27. <p>
28.   ~~~~~<br/>
29.   In example 2 we get the Page Title from the jcr:title property under the jcr:content node.
30.   We then use data-sly-test to only list the Page nodes under the resource.<br/>
31.   ~~~~~<br/>
32. <p>
33.
34. <h3 data-sly-use.jcrContent="${rootPage.path}/jcr:content">Ex 2: Child Pages of ${jcrContent.jcr:title}</h3>
35.
36. <ul data-sly-list=${rootPage.listChildren}>
37.   <li data-sly-test="${item.name != 'jcr:content'}">
38.     <a href="${item.path @ extension='html'}">${item.name}</a>
39.   </li>
40. </ul>
41.
42. <p>
43.   ~~~~~<br/>
44.   In Examples 3 and 4 we use the currentNode object <br/>
45.   Learn more about the <a href="https://docs.adobe.com/docs/en/spec/jcr170/javadocs/jcr-2.0/javax/jcr/Node.html">Node API</a><br/>
46.   ~~~~~<br/>
47. </p>
48.
49. <h3>Ex 3: Sibling Nodes of ${currentNode.name}</h3>
50. <ul data-sly-list=${currentNode.parent.nodes}>
51.   <li >
52.     ${item.name}
53.   </li>
54. </ul>
55.
56. <p>
57.   ~~~~~<br/>
58.   In example 4, only the siblings of the currentNode with primaryType=cq:Page are listed <br/>
59.   ~~~~~<br/>
60. </p>
61.
62. <h3>Ex 4: Sibling Pages of ${currentNode.name}</h3>
63. <ul data-sly-list=${currentNode.parent.nodes}>
64.   <li data-sly-test="${item.primaryNodeType.toString == 'cq:Page'}">
65.     <a href="${item.path @ extension='html'}">${item.name}</a>
66.   </li>
67. </ul>
```

6. Open a new tab in your browser, add <http://localhost:4502/content/hello-world.html> in the address bar, and then press Enter. Notice, how you can use the resource, node APIs, and HTL attributes to render the page content as shown:

The screenshot shows a web browser window with the following details:

- Address Bar:** localhost:4502/content/hello-world.html
- Toolbar:** Back, Forward, Stop, Refresh, Home, etc.
- Search Bar:** Search
- Content Area:**
 - ## My Inherited Page Script
 - ### My Page Script
 - The examples below use HTL attributes
Learn more about these attributes in the [HTML specification](#)
 - In examples 1 and 2 we use the resource /content/we-retail/language-masters/en
Learn more about the [Resource API](#)
 - Section Header:** Ex 1: Child resources of en
 - jcr:content
 - experience
 - men
 - women
 - equipment
 - about-us
 - products
 - user
 - In example 2 we get the Page Title from the jcr:title property under the jcr:content node. We then use data-sly-test to only list the Page nodes under the resource.



AEM Sites Development: Key Concepts

Introduction

Adobe Experience Manager (AEM) helps build and manage online content. To use AEM capabilities effectively, you should first understand the key concepts such as templates, core and proxy components, responsive pages, and context-aware configurations.

Objectives

After completing this module, you should be able to:

- Explain different types of templates
- Explain core components and proxy components
- Explain responsive layout editing
- Explain context-aware configuration
- Create and enable an editable template

Templates

AEM helps organizations build a website and add content to webpages. Based on business requirements, authors create pages by using a specific template, and add content to it by using different components.

A template defines the structure of the resultant page, any initial content, and the components that can be used (design properties) on a webpage.

AEM offers two basic types of templates for creating pages:

- Static
- Editable

Static Templates

A static template is a hierarchy of nodes that has the same structure as the page to be created, but without any actual content. Developers can define and configure static templates.

Editable Templates

An editable template retains a dynamic connection to any pages created from them. This ensures that any changes to the template are reflected in the pages. Template authors create and manage editable templates.

Creating Templates: Roles

Template creation requires collaboration between the following roles:

- Admin: Creates a new folder for templates.
- Developer: Focuses on technical/internal details and provides the template author with some required information. Experience in working with development environment.
- Template/Power author: Creates templates and configures the use of components. Template authors require privileges and permissions for creating templates and technical information from the developer. They should have experience in technical tasks, such as using patterns when defining paths.

Templates Console

The Templates console helps template authors or power authors create a new template:

- Create a new template
- Edit the template by using the template editor
- Manage the template life cycle

You can access the Templates console from the **Tools > General** section.

Template Editor

The template editor helps template authors:

- Add the available components to the template and position them on a responsive grid
- Preconfigure components
- Define the components that you can edit on the resultant pages (created from the template)
- Compose templates out of available components
- Manage the lifecycle of templates

The template editor has the following modes:

- Structure
- Initial Content
- Layout

Structure

Helps template authors define the components, such as header, footer, and layout container, that need to be on the pages. Template authors need to add a paragraph system to the template to allow page authors to add and remove the components added in the Structure mode from the resultant pages. When components are locked, page authors cannot edit the content. In structure mode, any components that are the parent of an unlocked component cannot be moved, cut, or deleted.

Initial Content

Helps power authors define the content that needs to be included in all pages, by default. When a component is unlocked, template authors can define the initial content that will be copied to the resultant page(s) created from the template. Page authors can edit these unlocked components on the resultant page(s). In the initial content mode (and on the resultant pages), you can delete any unlocked components with an accessible parent, such as components within a layout container.

Layout

Helps power authors predefine the template layout for the required device formats.

Content Policies

The template editor helps create content policies for a template and components in Structure mode.

The content policies:

- Connect the predefined page policies to a page. These page policies define various design configurations.
- Enable you to assign a predefined design configuration to selected components. This helps preconfigure a component's behavior on the resultant page.
- Help add the allowed components and default components to the template.
- Define the number of columns (responsive settings) in the template.

Core Components and Proxy Components

In AEM, components are the structural elements that constitute the content of the pages being authored. Components make page creation simple but powerful for the author. Core components make page authoring more flexible and customizable. Developers can extend core components to offer custom functionality.

The Core components are developed in and delivered through GitHub. You can find the code on [GitHub](#).

GitHub helps with frequent Core component updates and gathers feedback from the AEM developer community. This helps customers and partners to follow similar patterns when building custom components.

Features of Core Components

The key features of Core components are as follows:

- Flexible configuration options to accommodate many use cases and preconfigurable capabilities to define the features available to page authors
- Frequent incremental functionality improvements
- Availability of the source code on GitHub
- Periodic release of content packages for component upgrades
- Component versioning through:
 - › Compatibility within a version with the flexibility for components to evolve
 - › Multiple versions of one component can coexist on the same environment
- Modern implementation through:
 - › Markup defined in HTML Template Language (HTL)
 - › Content model logic implemented with Sling Models

- Lean markup through:
 - › Block Element Modifier (BEM) notation:
 - » v1 Components follow Bootstrap naming conventions
 - » The current notation is v2
- Capability to serialize as JSON the content model for headless Content Management System (CMS) use cases



Note: Core components are not immediately available to authors; the development team must first integrate them into the AEM author environment and preconfigure the Core components from the template editor to make them available for authors.

When to Use Core Components?

Core components offer multiple benefits, and it is recommended you use them for new AEM projects.

Adobe recommends the following when using Core components:

- For new projects:
 - › Use the Core components wherever they match the needs of the project, or optionally extend them.
 - › Do not use the foundation components, except for the various Layout Containers and Paragraph Systems.
- For existing projects:
 - › Use the Foundation components, unless a site or component refactoring is planned.
- New custom components:
 - › Assess if an existing Core component can be customized.
 - › Build a new custom component by using the [Component Guidelines](#)

Proxy Components

To use Core components in your project, you must download and install core components package on your AEM author instance, create proxy components, load the core styles, and add components to the templates.

Core components must not be directly referenced from the content. To avoid direct reference, the Core components are added to a hidden component group (.core-wcm or .core-wcm-form). This prevents displaying them in the template editor.

Instead, of using the core components, you must create site-specific components called *proxy components*. The proxy components define the required component name and group to display to page authors and refers to a Core component as their super type. The proxy components do not contain anything and serve mostly to define the version of a component to use in the site. However, when customizing the Core components, these proxy components play an essential role for markup and logic customizations.



Note: It is a best practice to create proxy components from core components and use the proxy components in your project.

Responsive Layout Editing

It is important that websites offer customized views across devices, such as desktops, tablets, and mobile phones. AEM provides the responsive design capability to achieve this customized view.

Responsive Design

In responsive design, the website will respond to fit any screen size through client-side feature detection by using media queries.

Responsive design provides:

- Easy reading and access to content
- Clear and easy navigation
- Minimum of resizing, panning, and scrolling across devices

Making Content Responsive

You can make the content responsive by using the traditional workflow or responsive layout editing.

Traditional Workflow

In a traditional workflow:

- A designer mocks different breakpoints
- A developer implements the breakpoints for a specific template
- The author picks that template and fills out the content

Responsive Layout Editing

In responsive layout editing, the author:

- Adds the content to the page
- Changes the page layout according to the device

Responsive Layout

AEM helps add the responsive layout to pages by using the following combination of mechanisms:

- Layout container component: Provides a grid-paragraph system to add and position components within a responsive grid
- Layout mode: Helps position the content within the responsive grid after the layout container is positioned on the page
- Emulator: Helps create and edit responsive websites that rearrange the layout according to the device or window size by resizing components interactively. It also helps the user to view how the content will render on different devices such as laptops or mobile phones

Responsive Grid

With the responsive grid mechanisms you can:

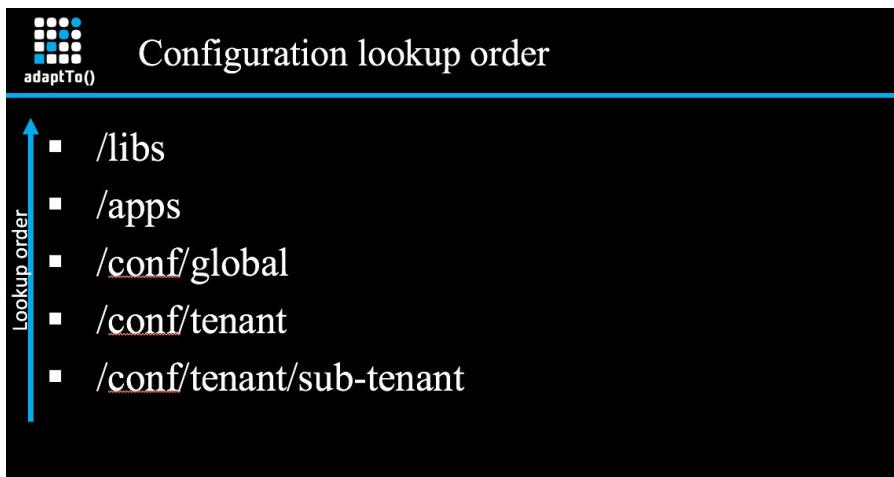
- Define differing content layouts based on device width (related to device type and orientation) by using breakpoints
- Ensure that the content is responsive to the size of the browser window on the desktop by using the same breakpoints and content layouts
- Place components in the grid, resize as required, and define when they should collapse/reflow to be side-by-side or above/below by using the horizontal snap to grid
- Hide components for specific device layouts
- Use nesting for column control

Context-Aware Configurations

The context-aware configuration's default implementation consists of lookup and persistence of configuration data, resource and property inheritance, and context path detection.

The context-aware configuration implementation provides a set of Service Provider Interfaces (SPI) that helps overlay, enhance, or replace the default implementation and adapt to your needs.

The previous model of context-aware configuration supported the /apps and /libs as the lookup order. The new model supports a more granular level of configuration, as shown:



Each of the above configuration containers are stored in the settings (subcontainer bucket). The resolution order considers the following locations:

- /libs/settings
- /apps/settings
- /conf/global/settings
- /conf/tenant/settings

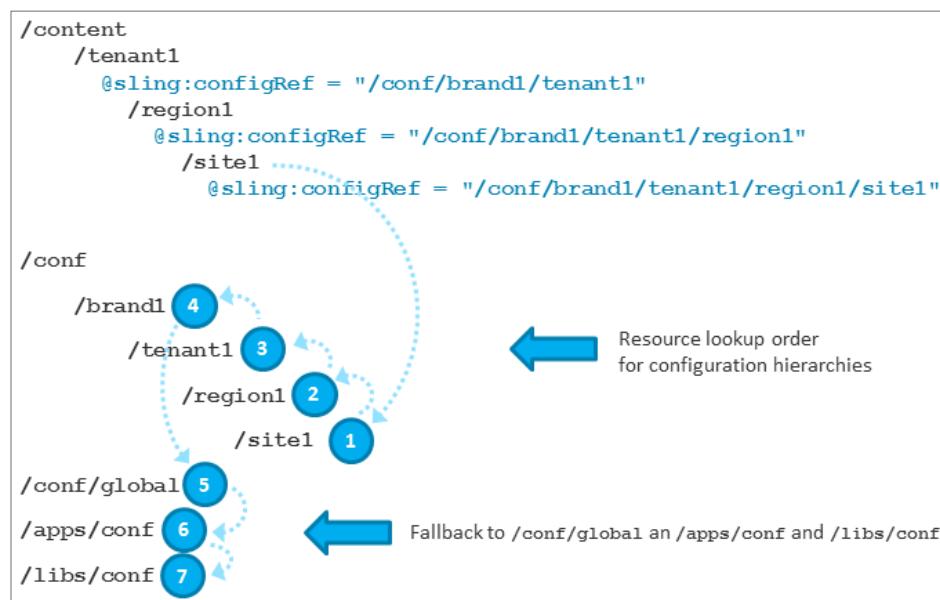
The context-aware configuration supports:

- Editable templates
- Content fragment models
- Translation cloud services
- ContextHub segments
- Workflows
- Asset configurations (schemas and profiles)
- AEM search filters

By default, all configuration data is stored in /conf. The fallback paths are /conf/global, /apps/conf, and /libs/conf. These paths are configurable in the service configuration.

The content resource hierarchy is defined by setting `sling:configRef` properties. Each resource that has a `sling:configRef` property set defines the root resource of a context, where the whole subtree is the context. Within the subtree, you can further define nested contexts.

The following illustration shows an example for configuration resource lookup:



The rules you must follow in resource lookup are:

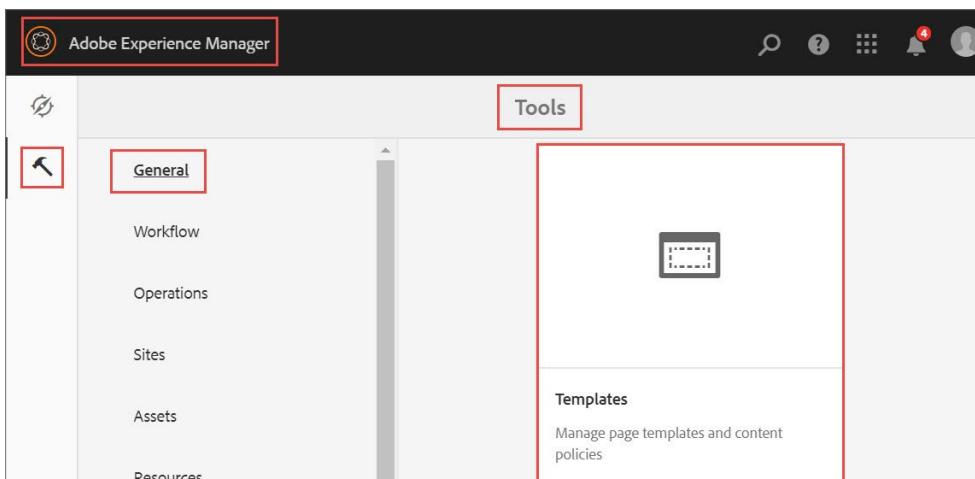
- Look up the content resource tree until a resource with `sling:configRef` is found. This is the inner most context.
- Check if a configuration resource exists at the path the property points to.
- Check for parent resources of the references configuration resource (below `/conf`).
- Look up the content resource tree for parent contexts, and check their configuration resources (as they may reference a completely different location below `/conf`).
- Check fallback paths.

Exercise 1: Create and enable an editable template

In this exercise, you will create a template with a page header and footer, title, and a paragraph system where page authors can drop any allowed components. Later, enable the template to make it available for page authors to create pages using the template.

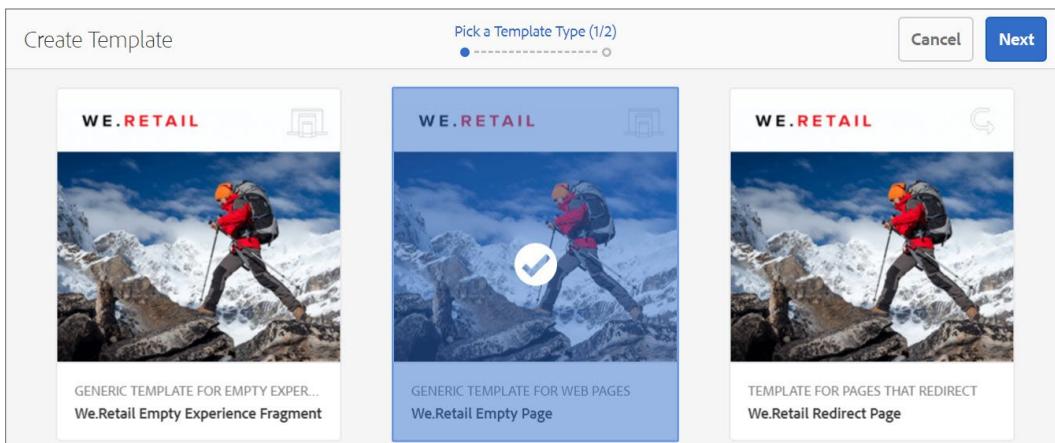
Task 1.1: Create an editable template

1. Click the **Adobe Experience Manager** from the header bar, click the Tools icon > General, and click **Templates** as shown. The **Templates** console opens.



2. Navigate to the **We.Retail** folder. You will see some existing templates.
3. Click **Create** in the upper right.

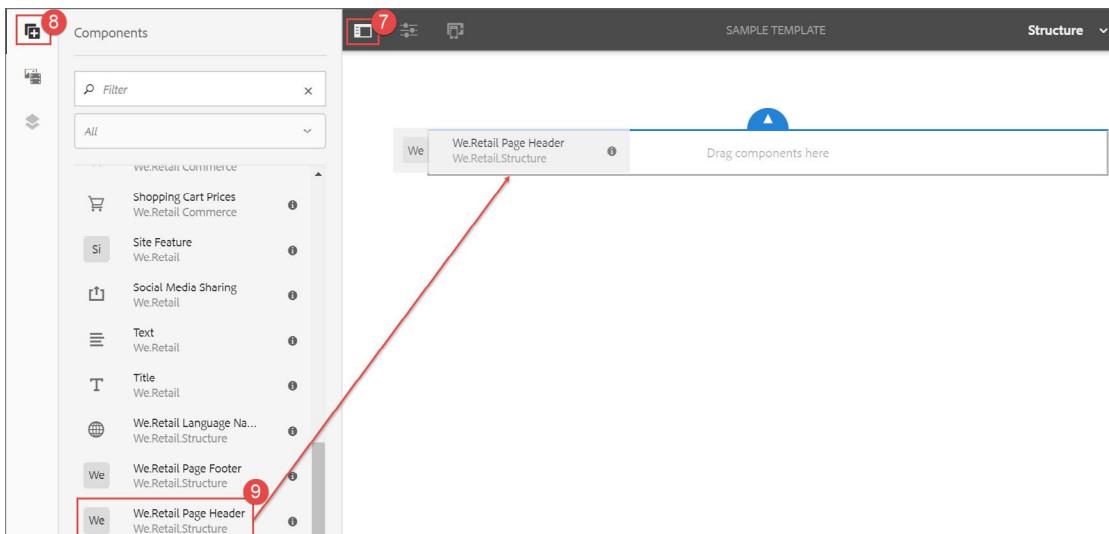
4. Select the **We.Retail Empty Page** template and click **Next** as shown:



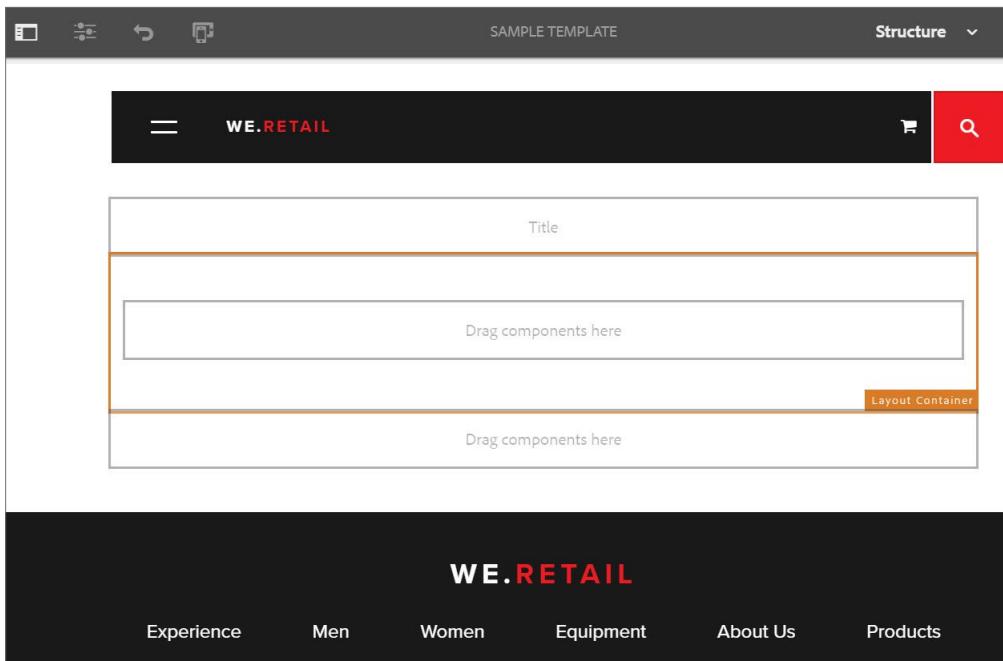
5. Enter **Sample Template** as the template title, and then click **Create**. The **Success** dialog box opens.
6. Click **Open**. The blank template opens in a new tab of the browser.

Let's add a page header and footer, and a title component to the template.

7. Ensure the Sample Template is open and click the Toggle Side Panel icon (as shown in the below screenshot based on this step's number). The panel opens.
8. Click the Components icon. The Components browser opens.
9. Look for the **We.Retail Page Header** component, drag it and drop onto the **Drag components here** placeholder as shown:



10. Similarly, drag the **We.Retail Page Footer** component and drop it below the header component.
11. Drag the **Title** component from components browser and drop it in between the previously added header and footer components.
12. Drag the **Layout Container** component from the components browser and drop it below the **Title** component. Your page should look similar to the one given in the below screenshot:



You do not want the header and the footer to be editable on the resultant pages. This is because what they display should be determined by their own logic, like building the top-level navigation from the page tree.

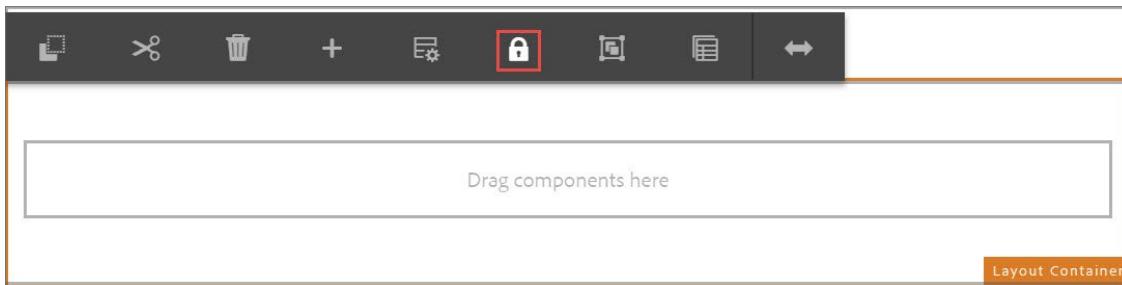
The title component does not necessarily need to be editable. By default, it displays the page title. The layout container, however, needs to be made editable or it will be quite pointless in its current state.

To make the layout container editable, you need to unlock it and assign a content policy, so it has a list of allowed components for resultant pages.

To unlock the layout container component on the template:

13. Select the **Layout Container** and click the Unlock structure component (lock) icon as shown.

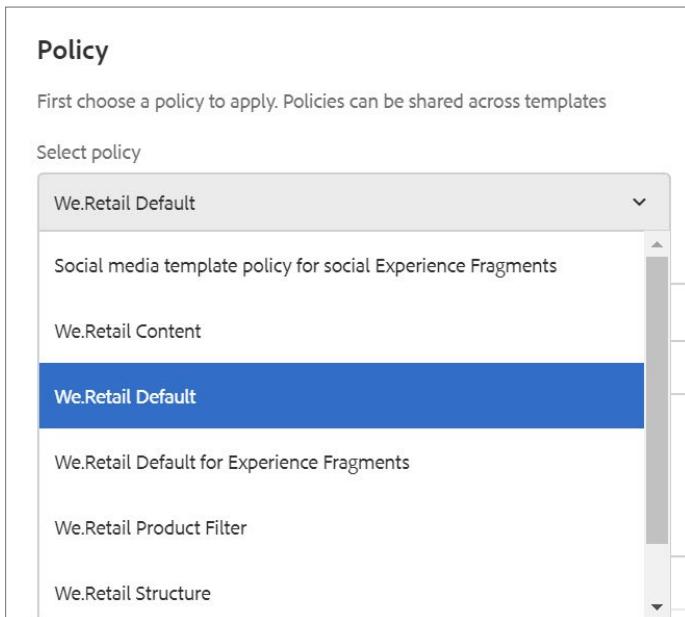
This unlocks the Layout Container component for resultant pages.



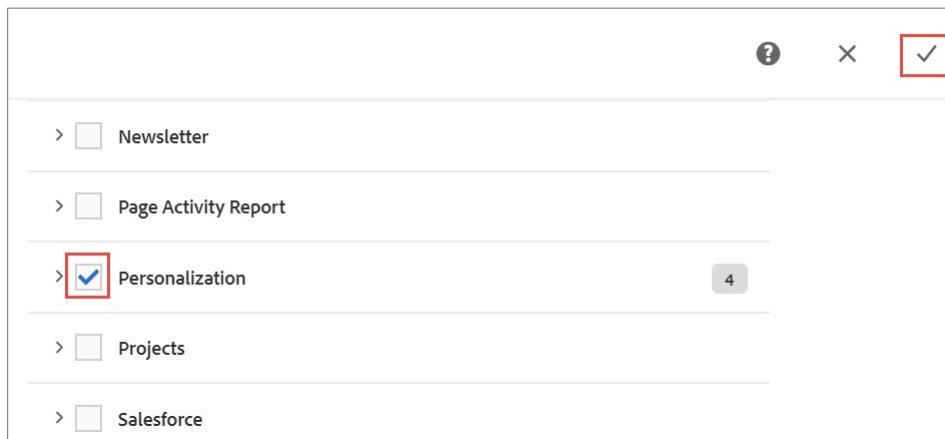
14. Select the **Layout Container** and click the Policy icon as shown. The **Layout Container Design** wizard opens on the same tab.



15. In the **Policy** section, select **We.Retail Default** from the **Select Policy** drop-down menu as shown:



16. In the **Properties** section, click the **Allowed Components** tab.
17. Scroll down and select the checkbox beside the **Personalization** group to add all the components of the Personalization group ((in this case, it will be four components, which is why you see 4 in this row)) to the Allowed Components list.
18. Click Done (the checkmark icon) in the upper right as shown. You will be taken back to the **Sample Template**.



19. Notice, how the allowed components display on the Layout Container as shown:

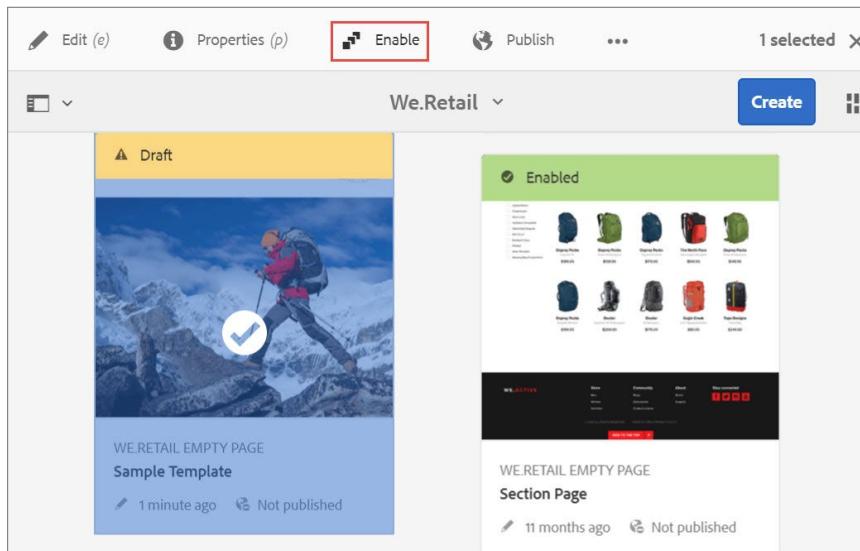
A screenshot of the 'SAMPLE TEMPLATE' interface. At the top, there is a header with a logo, the text 'SAMPLE TEMPLATE', and a 'Structure' dropdown. Below the header is a dark navigation bar with the text 'WE.RETAIL'. The main content area features a 'Layout Container' component. To its left is a panel titled 'ALLOWED COMPONENTS' containing a grid of component icons: Title, Text, Target, Social Media Sharing, Site Feature, Shopping Cart Prices, Shopping Cart, Product Grid, Product Recommendations, Product, Order History, Order Details, Navigation, Mini Shopping Cart, List, Link Button, Layout Container, and Image. Below this panel is a large, empty box with the placeholder text 'Drag components here'. At the very bottom of the screen is a dark footer navigation bar with links for 'Experience', 'Men', 'Women', 'Equipment', 'About Us', and 'Products'. The word 'WE.RETAIL' is centered above the footer.

Task 1.2: Enable the editable template

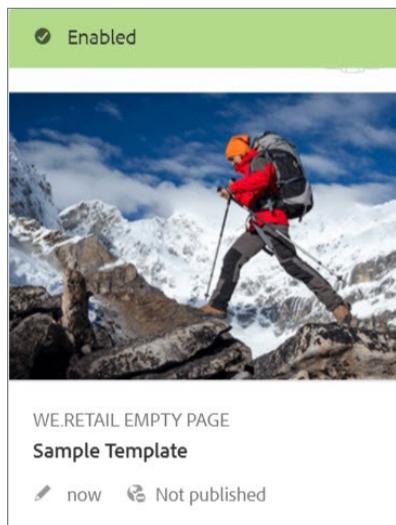
You need to enable the template to make it available for the page authors to create pages.

To enable the template:

1. Click the **Adobe Experience Manager** from the header bar, click the Tools icon > General, and click **Templates**. The **Templates** console opens.
2. Navigate to the **We.Retail** folder and select the **Sample Template**, and then click **Enable** from the actions bar as shown. The **Enable** dialog box opens.



3. Click **Enable**. Notice, the status of the Sample Template tile changes from **Draft** to **Enabled** as shown. You can now use the template to create your pages in **Sites** console.



4. Click **1 selected x** in the upper right to deselect the template.

References

You can use the following links for more information on:

- Templates:

<https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/templates.html>

- Editable Templates:

<https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/page-templates-editable.html>

- Responsive Design:

<https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/responsive.html>

- Context-Aware Configurations:

<http://sling.apache.org/documentation/bundles/context-aware-configuration/context-aware-configuration-default-implementation.html>



Creating Editable Templates and Pages

Introduction

In Adobe Experience Manager (AEM), templates authors use templates for creating pages. AEM provides both static templates and editable templates. Authors can create and configure the editable templates without a development project or iteration. However, to enable this capability in AEM, developers need to set up the required environment and create client libraries and components to be used in the page.

Objectives

After completing this module, you should be able to:

- Explain the types of templates
- Create an editable template
- Explain proxy components
- Create proxy components
- Create a content root
- Create a site structure

Types of Template

For content authors, AEM provides an easier way to create Page /content. The Sites console makes it easy for authors to create pages with the help of a wizard interface.

The AEM template defines the content structure with JCR nodes and properties. Templates use the node type **cq:Template** as the root node.

The following table describes both static and editable templates:

Static Templates	Editable Templates
Are developed and configured by developers	Are created and edited by authors
Have the same structure as the page	Help define the structure, initial content, and content policies for pages
Are copied to create a new page and do not have a dynamic connection with the page	Maintain a dynamic connection between the template and pages
Use the Design mode to persist design properties.	Use content policies (edited from the template editor) to persist the design properties
Are stored under /apps	Are stored under /conf

Creating Editable Templates

The steps to create an editable template are:

1. Create a template folder to organize your templates.
2. Create a template type for your template.
3. Create a new template from the template type.
4. Define additional properties for the template if required.
5. Edit the template to define the structure, initial content, layout, and content policies.
6. Enable the template (in order) to use it when creating a page.
7. Allow the template for the required page or the branch of your website.
8. Publish the template to make it available on the publish environment.

Creating the Template Folder

To organize your templates, you can use the following folders:

- global
- site-specific

In a standard AEM author instance the global folder exists in the **Templates** console. This folder contains the default templates and acts as a fallback if no policies and/or template-types are found in the current folder.

You can add your default templates to global folder or create a new folder (recommended). The site-specific folders help organize templates specific to your site.

 **Note:** It is best practice to create a new folder to save your customized templates and not to use the global folder. Folders must be created by a user with admin rights.

You can create a template folder from:

- CRXDE Lite
- Configuration Browser console

Creating a Template Folder from CRXDE Lite

In CRXDE Lite, to create a template folder, you need to :

1. Create the following structure:

```
/conf
<your-folder-name> [sling:Folder]
    settings [sling:Folder]
        wcm [cq:Page]
        templates [cq:Page]
        policies [cq:Page]
```

2. Define the following properties on the folder root node:

```
<your-folder-name> [sling:Folder]
    a. Name: jcr:title
    b. Type: String
    c. Value: The title (for the folder) you want to appear in the Templates console.
```

3. Assign group(s) and define the required Access Control Lists (ACLs) for authors to create templates in the new folder (in addition to the standard authoring permissions and privileges; for example, content-authors).
4. Assign authors to the **template-authors** group.

Creating a Template Folder from the Configuration Browser Console

You can create template folders from the **Configuration Browser** console. You can access this console from the **Tools** console. You can create a multi-tenant environment where tenants can have different editable templates, configurations, cloud services, and data models.



Note: The above methods are standard processes for creating a template folder. You will perform this task in Exercise 1.

Creating a Template Type

When creating a new template, you must specify a template type. The template type is copied to create the template, and the structure and initial content of the selected template type is used to create the new template. After the template type is copied, the only connection between the template and the template type is a static reference for information purposes only.

Template types help you define:

- The resource type of the page component.
- The policy of the root node, which defines the components allowed in the template editor.

The out-of-the box template types are stored under **/libs/settings/wcm/template-types** as shown:

The screenshot shows the CRXDE Lite interface with the URL `/libs/settings/wcm/template-types` in the address bar. The left sidebar shows a tree view of the repository structure, with a red box highlighting the `template-types` folder under `policies`. The main content area displays the properties of this folder. A table titled "Properties" lists the following columns: Name, Type, Value, Protected, Mandatory, Multiple, and Auto Created. The rows show the following data:

Name	Type	Value	Protected	Mandatory	Multiple	Auto Created
1 jcr:created	Date	2018-03-13T10:12:45.000+00:00	true	false	false	true
2 jcr:createdBy	String	admin	true	false	false	true
3 jcr:mixinTypes	Name[]	rep:AccessControl	true	false	true	false
4 jcr:primaryType	Name	cq:Page	true	true	false	true

Template Definitions

Definitions for editable templates are stored user-defined folders (recommended) or alternatively in the global folder. For example:

- `/conf/<my-folder>/settings/wcm/templates`
- `/conf/<my-folder-01>/<my-folder-02>/settings/wcm/templates`

The root node of the template is of type **cq:Template** with the following skeleton structure:

```
<template-name>
  initial
    jcr:content
      root
        <component>
          ...
        <component>
      jcr:content
        @property status
      policies
        jcr:content
          root
            @property cq:policy
            <component>
              @property cq:policy
              ...
            <component>
              @property cq:policy
```

The main elements are:

- <template-name>
- jcr:content
- structure
- initial
- policies
- thumbnail.png

jcr:content

This node holds the following properties of the template:

- Name: jcr:title
- Name: status
 - › Type: String
 - › Value: draft, enabled, or disabled

structure

The structure node defines the structure of the resultant page. The structure of the template is merged with the initial content (/initial) when creating a new page. Any changes to the structure will be reflected in any pages created with the template.

The root (structure/jcr:content/root) node defines the list of components that will be available in the resulting page. The components defined in the template structure cannot be moved on or deleted from any resultant pages. After a component is unlocked the editable property is set to true. After a component that already contains content is unlocked, this content will be moved to the initial branch. The **cq:responsive** node holds definitions for the responsive layout.

initial

The initial node defines the initial content that a new page will have upon creation. The initial node contains a **jcr:content** node that is copied to any new pages and is merged with the structure (/structure) when a new page is created. Any existing pages will not be updated if the initial content is changed after creation. The root node holds a list of components to define what will be available in the resulting page. If content is added to a component in Structure mode and that the component is subsequently unlocked (or vice versa), the content is used as initial content.

policies

The content (or design) policies define the design properties of a component. For example, the components available or the minimum/maximum dimensions. These are applicable to the template (and pages created with the template). Content policies can be created and selected in the template editor. The property `cq:policy`, on the `/conf/<your-folder>/settings/wcm/templates/<your-template>/policies/jcr:content/root` root node, provides a relative reference to the content policy for the page's paragraph system. The property `cq:policy`, on the component-explicit nodes under root, provides links to the policies for the individual components. The actual policy definitions are stored under `/conf/<your-folder>/settings/wcm/policies/wcm/foundation/components`.

 **Note:** The paths of policy definitions depend on the path of the component. `cq:policy` holds a relative reference to the configuration itself.

Layout

When editing a template, you can define the layout. The layout uses the standard responsive layout that can be configured.

Page Policies

Page policies help define the content policy for the page (main parsys), in either the template or resultant pages.

Creating a Template

After you create a template type, you can use the template type to create a new template in the **Templates** console.

Editing the Template

After creating a template, you can edit the template from the template editor. The changes made to the template will have impact on the existing pages depending on the template editor modes. For example, if you make changes to the structure of the template, it will apply immediately to all pages created from the template immediately. It is also possible to define the initial content for a template, which will be copied over to newly-created pages.

Enabling the Template

Before you use a template, you should enable it:

- From the **Templates** console

OR

- By setting the status property on the jcr:content node
 - For example, on `conf/<your-folder>/settings/wcm/templates/<your-template>/jcr:content`
 - Define the following properties:
 - Name: status
 - Type: String
 - Value: enabled

To allow the template, you need to:

- Define the **Allowed Template path(s)** on the **Page Properties** of the appropriate page or root of the page of a sub-branch.
- Set the property **cq:allowedTemplates** on the jcr:content node of the required branch. For example, `/conf/<your-folder>/settings/wcm/templates/*`.

Publishing the Template

As the template is referenced when a page is rendered, the (fully configured) template needs to be published from the **Templates** console to make it available on the publish environment.

Exercise 1: Create an editable template

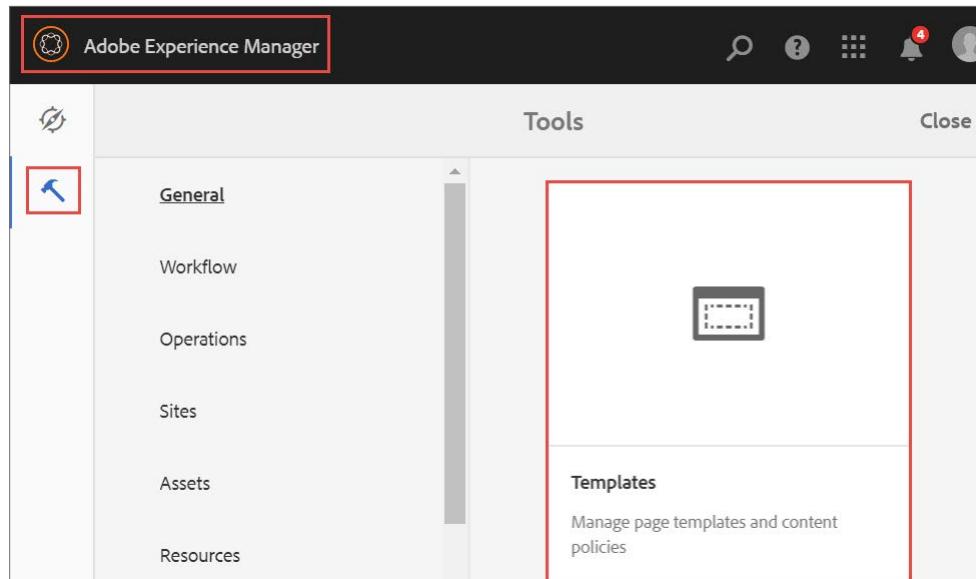
In this exercise, you will perform the following tasks:

1. Create a context-aware configuration
2. Create an empty template type
3. Create a development template

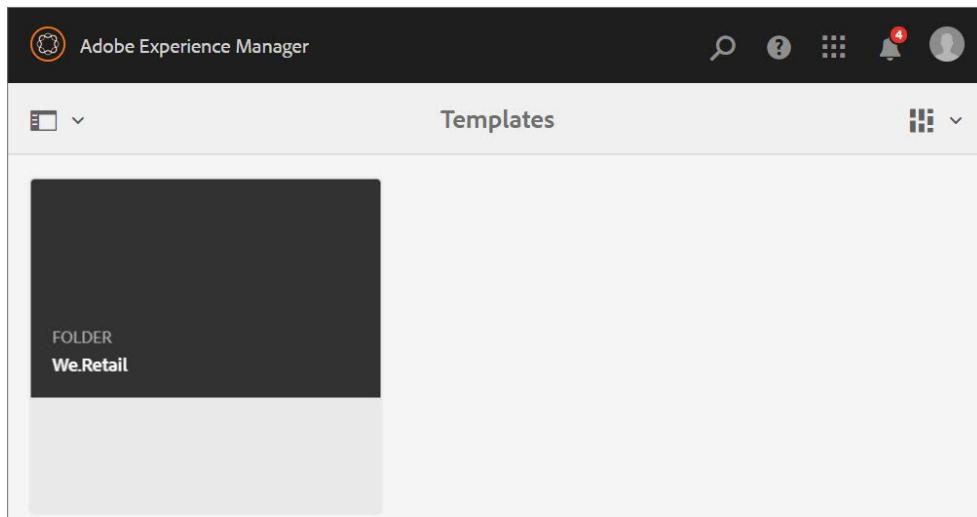
Task 1.1: Create a context-aware configuration

Before you create a context-aware configuration, let's check if there is a way to create a template folder in the **Templates** console.

1. Click the **Adobe Experience Manager** from the header bar to open the navigation pane.
2. Click the Tools icon, and then click **Templates** as shown. The **Templates** console opens.

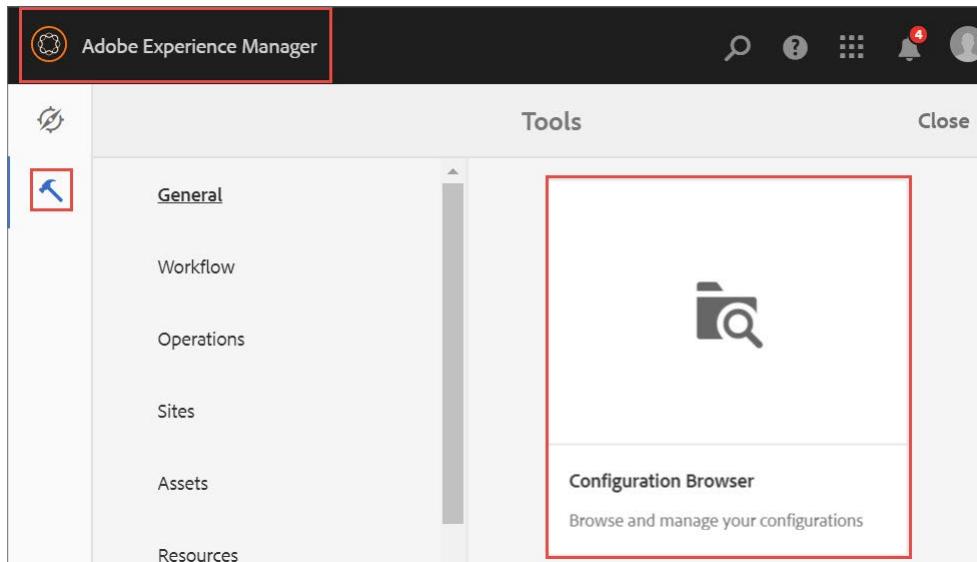


Notice, there is no way to create a new template folder in the **Templates** console as shown:



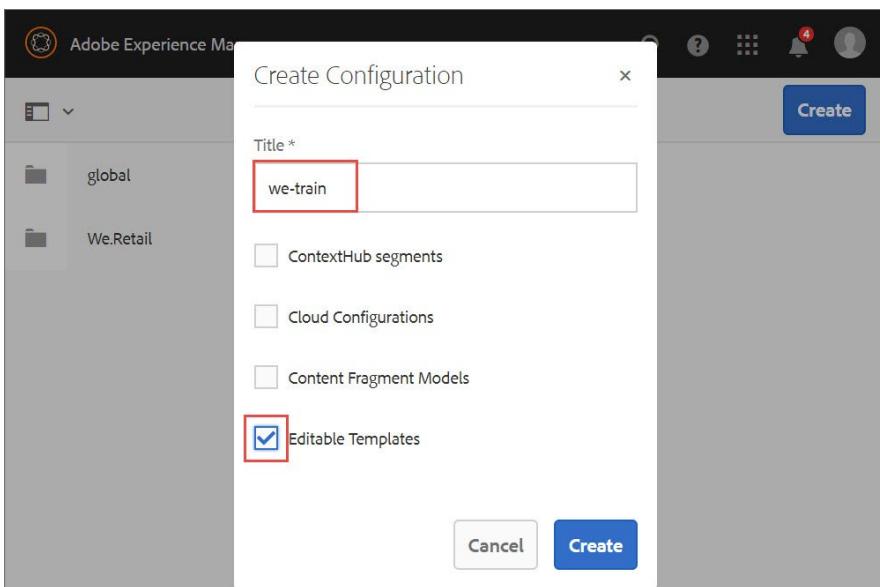
To create a new folder for editable templates, you need to create a context-aware configuration.

3. Click the Tools icon and then click **Configuration Browser** as shown. The **Configuration Browser** console opens.



4. Click **Create** from the actions bar. The **Create Configuration** dialog box opens.

5. Enter and select the following:
 - a. **Title: we-train**
 - b. Select the **Editable Templates** checkbox
6. Click **Create** as shown. The green success message that confirms the configuration is created appears at the top of the page.



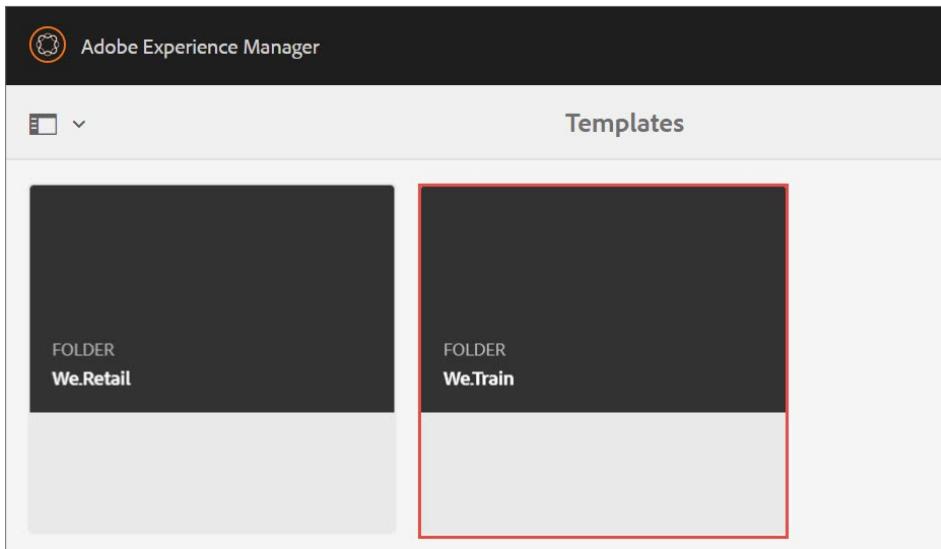
7. Click the thumbnail icon beside the **we-train** configuration, and then click **Properties** from the actions bar as shown. The **Configuration Properties** wizard opens.

Title	Modified	Modified By
we-train		

8. Update the title to **We.Train**.
9. Click **Save & Close**. The green success message that confirms the changes are saved appears at the top of the page.

10. Click the **Adobe Experience Manager** logo from the header bar, click Tools, and then click **Templates**. The **Templates** console opens.

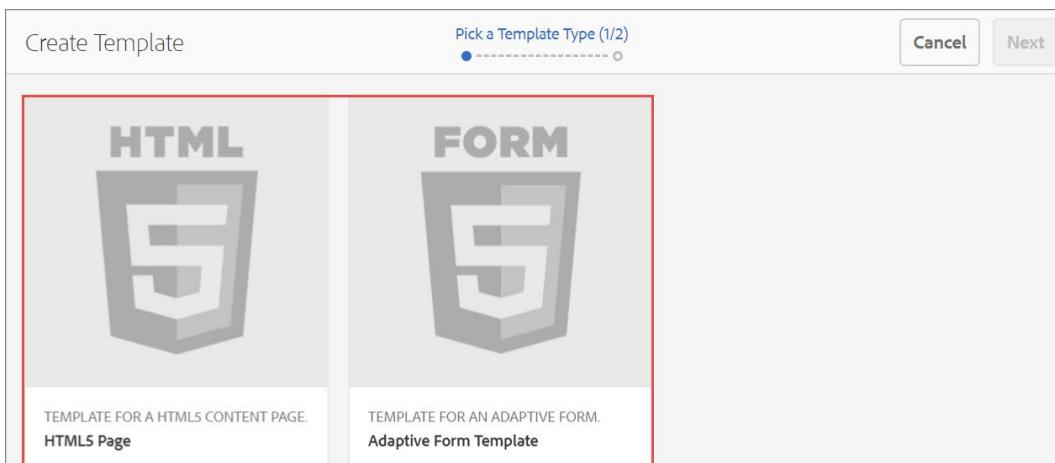
Notice, the **We.Train** folder is available as shown. You will use this folder in a later exercise to create an editable template.



Task 1.2: Create an empty page template type

Before you create a template type, let's check the template types that are available out of the box.

1. Click the **Adobe Experience Manager** from the header bar, click the Tools icon, and click **Templates**. The **Templates** console opens.
2. Navigate to the **We.Train** folder and click **Create** from the actions bar. Notice, when you try to create an editable template right now, there are only two out-of-the-box template types as shown:



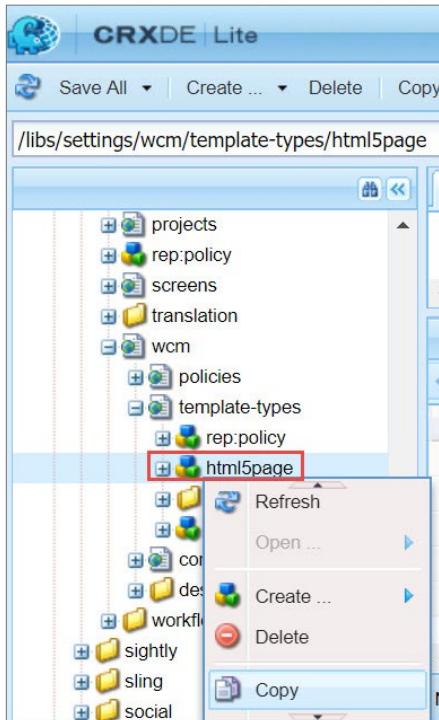
3. Click **Cancel** to close the **Create Template** wizard.

Before creating a custom template type, first you need to create a component that will be used as the base when creating pages. The base page is responsible for ensuring that global areas of the site are consistent. This includes loading of global CSS and Javascript, as well as the inclusion of code that will ensure the page can be edited using AEM authoring tools.

You will install the **adls-training-project-v2.0.zip** package that contains the training project structure along with the page component that was already created for you.

To create a custom template type:

4. Click **Adobe Experience Manager** from the header bar. The **Tools** console opens.
5. Click **CRXDE Lite**. The **CRXDE Lite** page opens.
6. Navigate to the **/libs/settings/wcm/template-types/html5page** node.
7. Right-click the **html5page** node and click **Copy** from the drop-down menu as shown:



8. Navigate to the **/conf/we-train/settings/wcm/template-types** folder.

- Select the **template-types** folder and click **Paste** from the actions bar. The **html5page** node is added to the template-types folder as shown:



- Click **Save All** from the actions bar.



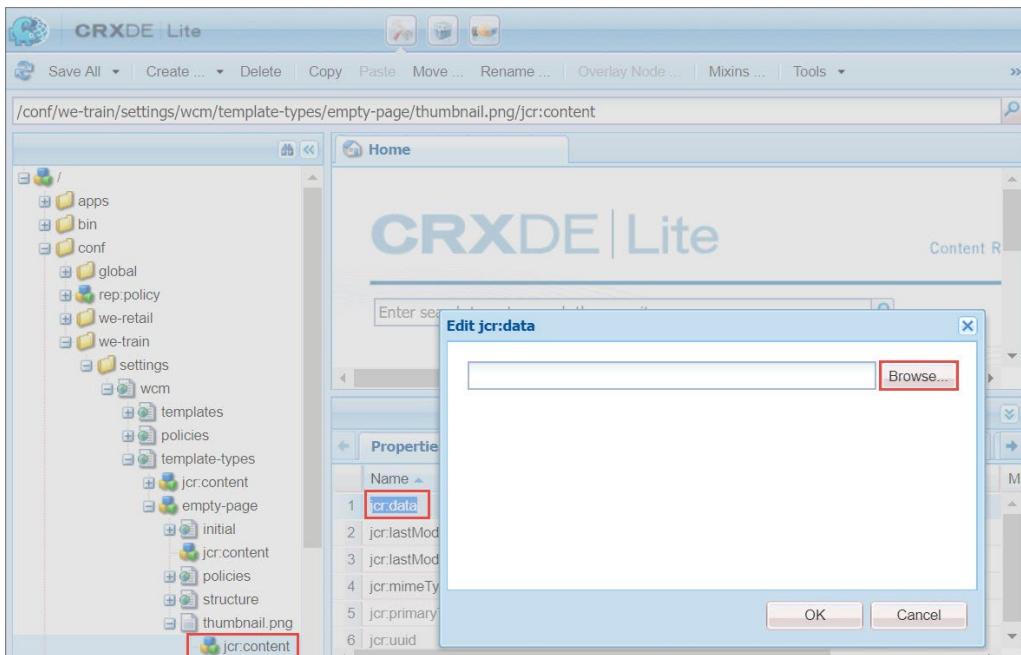
Note: you can also press Ctrl + S (Windows users) and Command + S (Mac users) to save your work in CRXDE Lite.

- Right-click the **html5page** node and click **Rename** from the shortcut menu.
- Rename the node to **empty-page**.
- Navigate to the **empty-page/jcr:content** node and modify the values of following properties:
 - jcr:title: We.Train Empty Page**
 - jcr:description: Empty Template for a We.Train page**
- Click **Save All** from the actions bar.
- Navigate to the **empty-page/initial** page, select the **jcr:content** node, and modify the value of the following property:
 - sling:resourceType: training/components/structure/page**
- Click **Save All** from the actions bar.
- Navigate to the **empty-page/structure** page, select the **jcr:content** node, and modify the value of the following property:
 - sling:resourceType: training/components/structure/page**
- Click **Save All** from the actions bar.
- Navigate to the **/conf/we-train/settings/wcm/template-types/empty-page/policies/jcr:content** node, select the **root** node, right-click and click **Delete** from the shortcut menu. This helps you start with an empty root layout container.
- Click **Save All** from the actions bar.

21. Navigate to the `/conf/we-train/settings/wcm/template-types/empty-page/thumbnail.png/jcr:content` node.

22. On the **Properties** tab, double-click the **jcr:data**. The **Edit jcr:data** dialog box opens.

23. Click **Browse** as shown:

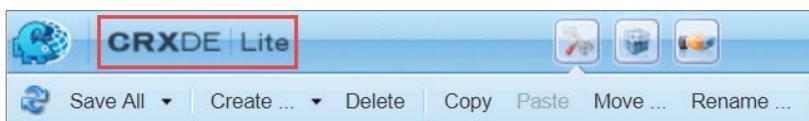


24. Navigate to the **Exercise_Files** folder on your file system, double-click to open the folder, select the **We.Train-thumbnail.png** image, and then click **Open**.

25. Click **OK**. The thumbnail image is added to template-type.

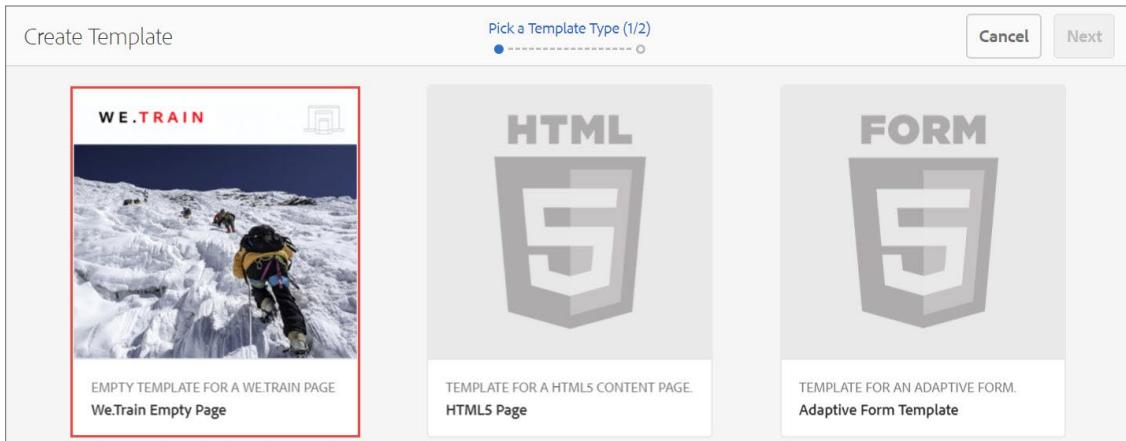
To test to whether the template type is available for creating editable templates:

26. Click **CRXDE Lite** from the header bar, as shown. You will be taken back to the **Navigation** page.



27. Click the Tools icon, and then click **Templates**. The **Templates** console opens.

28. Navigate to the **We.Train** folder, click **Create** from the actions bar. You should now see the **We.Train Empty Page** template type as shown:



29. Click **Cancel**.

Task 1.3: Create a development template

After you have created the template-type, you need to create a development template. The development template will let you use the template authoring environment and to setup intial template configurations. After the setup is ready in the development template, you will copy it back to the template type.

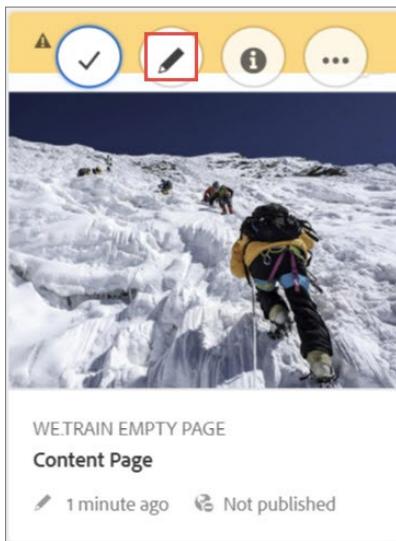
To create a development template:

1. Ensure you are in the **Templates** console or add the following URL <http://localhost:4502/libs/wcm/core/content/sites/templates.html/conf> in the address bar of the browser, and press Enter from the keyboard.
2. Navigate through the **We.Train** folder and click **Create** from the actions bar. The **Create Template** wizard opens.
3. Select the **We.Train Empty Page** template, and click **Next**.
4. Enter **Content Page** in the **Template Title** field, and then click **Create** as shown:

The screenshot shows the 'Create Template' interface with the title 'Create Template' at the top left. At the top center, it says 'Template Details (2/2)' with a progress bar showing two dots filled. On the right are 'Back' and 'Create' buttons. Below the title, there are two input fields:

- Template Title ***: A text input field containing 'Content Page' with a red border around it.
- Description**: A large text area for entering a description.

5. Click **Done** in the **Success** dialog box.
6. Hover over the **Content Page** template, and click the Edit icon as shown. The **Content Page** template opens in a new tab of the browser.



At this point, you will see the default page.html script that you installed in the previous task. To use the editable templates, you need to inherit the default script from the Core Page component. To do that:

7. Open a new tab of the browser and add the URL <http://localhost:4502/crx/de/index.jsp> in the address bar, and press Enter from the keyboard. The **CRXDE Lite** page opens.
8. Navigate to the `/apps/training/components/structure/page` node.
9. Select **page.html** and click **Delete** from the actions bar. The page is deleted.
10. Click **Save All** from the actions bar.
11. In your browser, navigate to the tab where the Content Page template is open, and refresh the page. The Template Editor helper wizard (Modes) dialog box opens.
12. Select the **Don't show this again** checkbox and click **Close**.

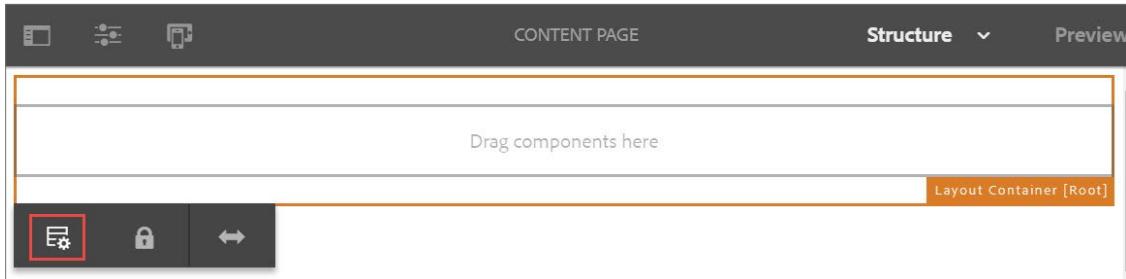
Notice that the `page.html` script does not appear and the **Layout Container [root]** is now visible, as shown. The Layout Container [root] helps configure the components that can be added to the template.



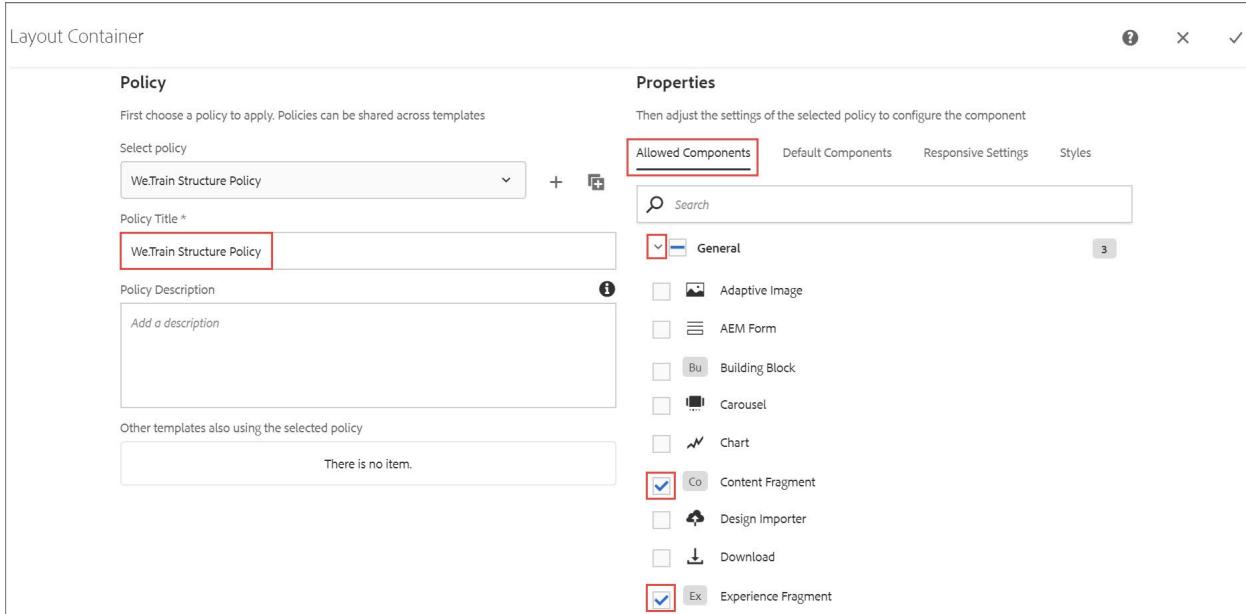
 Note: The root layout container is visible to template authors only when editing the template. This container will not be available on the page.

You now need to add few structure components to the root layout container so that you can start building the template.

13. Select the **Layout Container [root]**, and click the Policy icon as shown. The Layout Container wizard opens.



14. In the **Policy** section, enter **We.Train Structure Policy** in the **Policy Title** field.
15. In the **Properties** section, ensure you are in the **Allowed Components** tab, and expand the **General** group. The list of components available in the group appear.
16. Select the checkbox beside **Content Fragment** and **Experience Fragment** as shown:



17. Scroll down and select the checkbox beside **Layout Container**, and then click the checkmark icon (Done). The selected components are added to the policy. You do not have to do anything else, as your changes are automatically saved.

Proxy Components

Components are modular, reusable units that implement specific functionality or logic to render the content of your website.

Two sets of AEM components are available out-of-the-box:

- Core components - Offer flexible and feature-rich authoring functionality
- Foundation components - Are based on legacy technologies. It is still supported but no longer enhanced.

The Core components provide several basic building blocks for creating content. This includes Text, Image, and Title and several other components. These can be included in your project by creating a **cq:Component** for each one and using the **sling:resourceSuperType** property to point to the *Core Component*. This is known as creating *proxy components* and is the recommended way of using Core Components in your project.

Exercise 2: Create proxy components

In this exercise, you will create the following proxy content components and add them to the We.Train component group:

- Title
- List
- Text
- Breadcrumb
- Image

Later, you will add the We.Train group to your template through the policy.

To create proxy content components:

1. Click the **Adobe Experience Manager** icon, and then click the Tools icon.
2. Click **CRXDE Lite** or enter the <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
3. Navigate to the **/apps/training/components/content** folder.
4. Select the **content** folder, click **Create** from the actions bar, and then select **Create Node** from the drop-down menu. The **Create Node** dialog box opens.
5. Enter the following:
 - a. **Name: title**
 - b. **Type:** Select **cq:Component** from the drop-down menu
6. Click **OK**. The Title component is created.
7. Click **Save All** from the actions bar.

8. Select the **title** node, and add the following properties:

Field	Type	Value
sling:resourceSuperType	String	core/wcm/components/title/v2/title
jcr:title	String	Title
componentGroup	String	We.Train



Note: When adding the above properties, first you need to add the property, click Add, and then click **Save All**. Follow the same procedure for other properties.

9. Click **Save All** from the actions bar.

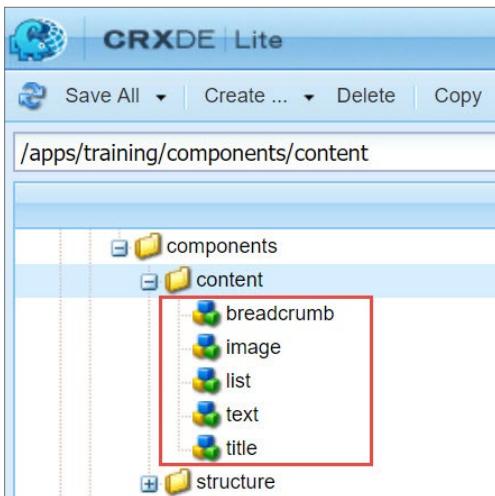
To create more proxy components:

10. Select the **title** proxy component, click **Copy** from the actions bar, select the **content** folder and then click **Paste** from the actions bar. The Copy of Title node is created.
11. Select **Copy of title** node, click **Rename** from the actions bar, and rename the node to **list**.
12. Select the **list** node and update the following values:

Field	Type	Value
sling:resourceSuperType	String	core/wcm/components/list/v2/list
jcr:title	String	List

13. Click **Save All** from the actions bar.

14. Perform the steps 10 through 12 and create text, breadcrumb, and image components.
15. Update the values of **sling:resourceSuperType** and **jcr:title** properties according to the proxy component name.
16. Ensure to click **Save All** after creating the proxy components. Your proxy components list should look similar to the list in the following screenshot:

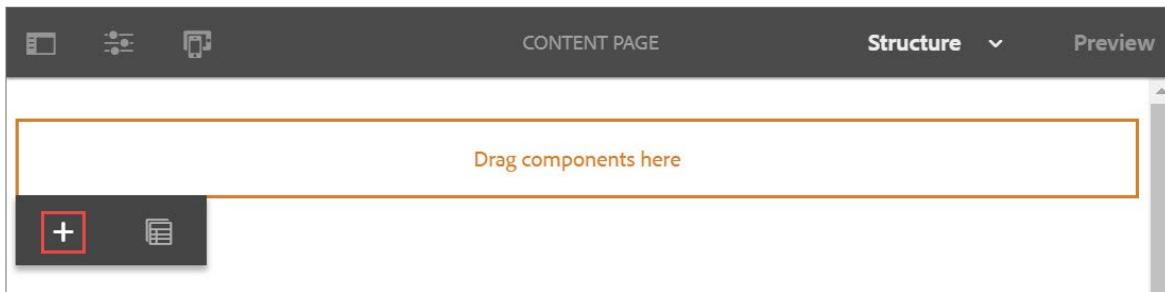


Exercise 3: Add content components to the template

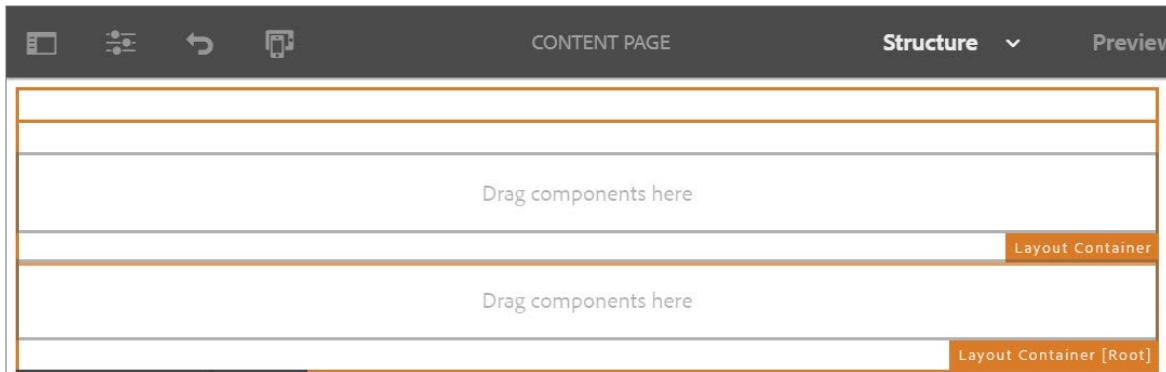
In this exercise, you will add the proxy components to the development template to help authors add components to a page. The page authors cannot edit the Layout Container [root] on the page, you will add a Layout Container to the template and specify the components that an author can use in the page through the Layout Container's content policy.

To add proxy components to the template:

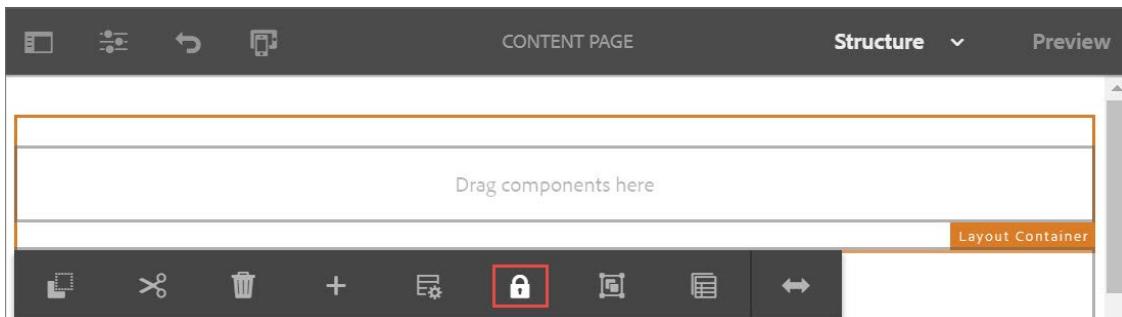
1. Open a new tab of your browser where the AEM author instance is running.
2. Click **Adobe Experience Manager** from the header bar, and then click the Tools icon.
3. Ensure you are in **General** section, click **Templates**. The **Templates** console opens.
4. Navigate to the **We.Train** folder, select the **Content Page** template, and then click **Edit** from the actions bar. The template opens in a new tab of the browser.
5. Ensure you are in **Structure** mode, click the **Drag components here**, and then click the Insert component icon as shown. The **Insert New Component** dialog box opens.



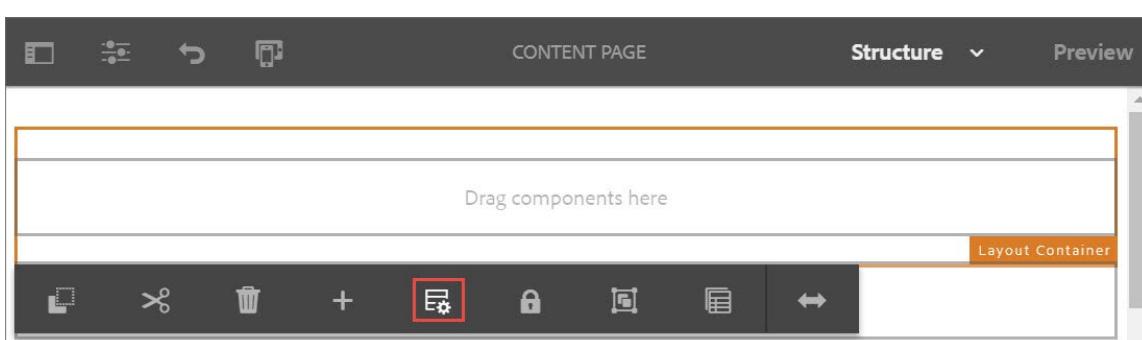
- Select **Layout Container** from the list of components. The component is added to the template as shown:



- Select the inner **Layout Container** (not the root Layout Container) and click the **Unlock Structure Component** icon from the component toolbar as shown. This allows you to add components to the container.



- Select the inner **Layout Container** and click the **Policy** icon from the component toolbar as shown. The **Layout Container** wizard opens.



- In the **Policy** section, enter **We.Train Content Policy** in the **Policy Title** field.
- In the **Properties** section, ensure you are in **Allowed Components** tab, and expand the **General** group. The list of components available in the group appear.

11. Select the checkbox beside **Content Fragment** and **Experience Fragment** as shown:

The screenshot shows the 'Layout Container' configuration dialog. On the left, there's a form with fields for 'Select policy' (set to 'WeTrain Content Policy'), 'Policy Title' (set to 'WeTrain Content Policy'), and 'Policy Description' (with placeholder 'Add a description'). Below these is a section for 'Other templates also using the selected policy' which says 'There is no item.' On the right, the 'Allowed Components' tab is selected, showing a list of components under the 'General' category. Two checkboxes are checked: 'Co Content Fragment' and 'Ex Experience Fragment'. Other components listed include Adaptive Image, AEM Form, Building Block, Carousel, Chart, Design Importer, Download, and Experience Fragment.

12. Scroll down and select the checkbox beside **We.Train** group, and then click checkmark icon (Done). This will add the newly created proxy components to the Layout Container from the We.Train group. Any new components with the **componentGroup=We.Train** property, will be added to the policy automatically.

Optional Exercise

Add two more Layout Containers to the template and use the Layout Mode to adjust the layout. Align the two Layout Containers to take up half the page into two columns. Remember to define the policies for both the Layout Containers!

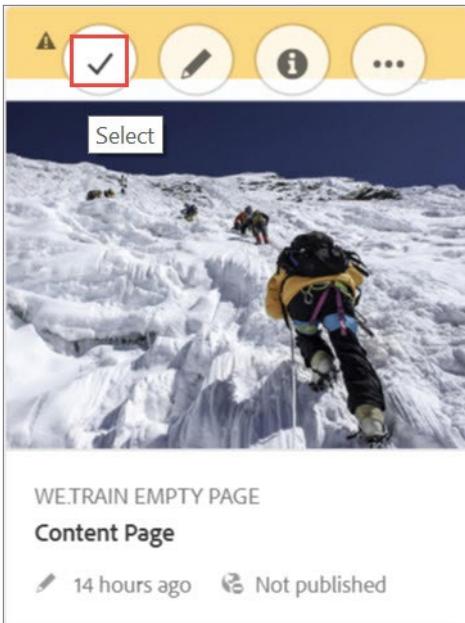
Hint: You can use the same policy for both the Layout Containers.

Exercise 4: Enable and publish the template

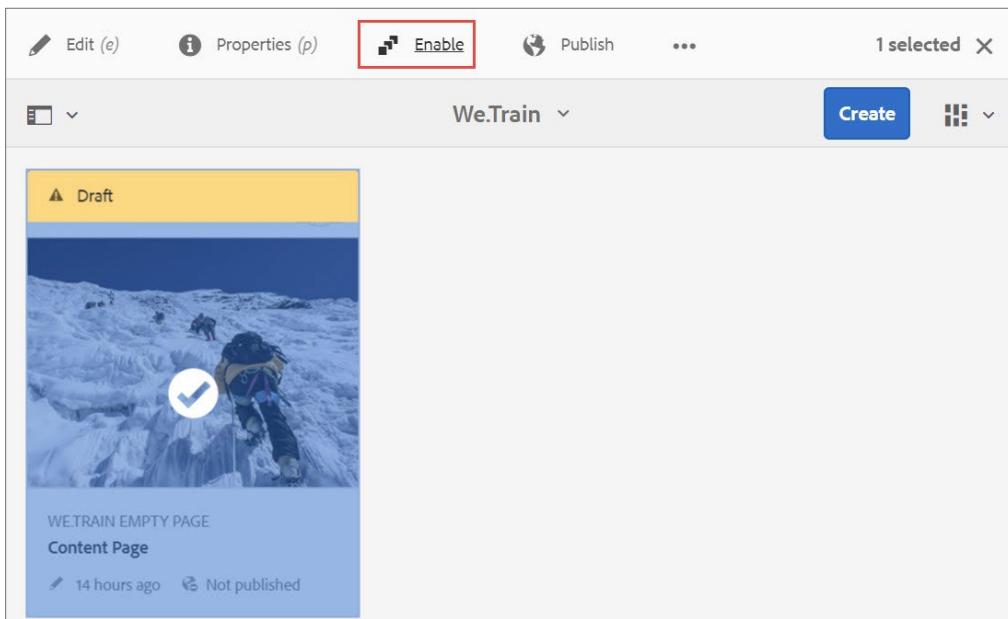
In this exercise, you will first enable the template to make it available for authors to create pages on your author instance. Later, publish the template to make it available on the publish instance.

To enable the template:

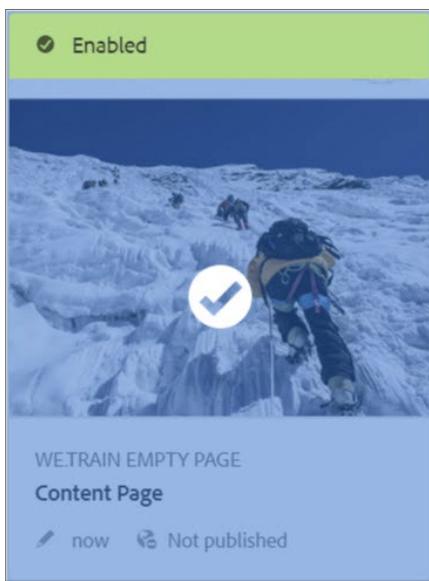
1. Open a new tab of your browser where the AEM author instance is running.
2. Click **Adobe Experience Manager** from the header bar, and then click the Tools icon.
3. Ensure you are in **General** section, click **Templates**. The **Templates** console opens.
4. Navigate to the **We.Train** folder, hover over the **Content Page** template, and click the checkmark (Select) icon as shown. The template is selected.



5. Click **Enable** from the actions bar as shown. The **Enable** dialog box opens.



6. Click **Enable**. Notice, the status of **Content Page** changes to **Enabled** as shown:



7. Ensure the **Content Page** template is selected, and click **Publish** from the actions bar. The **Publish** dialog box opens.
8. Ensure all checkboxes are selected, and click **Publish**. The message **INFO! The page has been published** appears.
9. Click **1 selected X** to deselect the template.

Creating Pages from Editable Templates

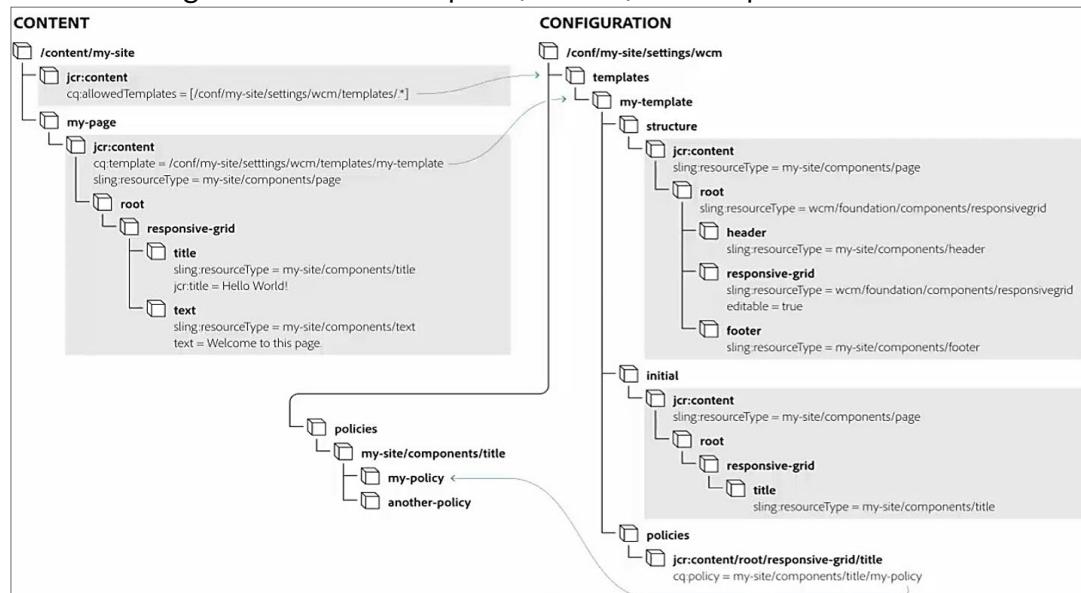
Before creating pages through the UI, a content root needs to be set up for your site. The content root will define the allowed templates for the site and is used to set other global configurations. By convention the content root is not intended to be the home page for the site and instead will redirect to the true home page.

Creating Pages

In the **Sites** console, you can create the pages of your website by using the template. The pages created from editable templates:

- Are created with a subtree that is merged from the structure and initial in the template
- Have references to the information held in the template and the template type. This is achieved with a jcr:content node with the following properties:
 - › **cq:template**: Provides the dynamic reference to the actual template, and enables changes to the template to be reflected on the actual pages.
 - › **cq:templateType**: Provides a reference to the template type.

The below diagram shows how templates, content, and components interrelate:



In the above diagram:

- Controller—The resultant page that references the template (`/content/<my-site>/<my-page>`)
 - The content controls the entire process. According to the definitions it accesses the appropriate template and components.
- Configuration—The template and related content policies define the page configuration (`/conf/<my-folder>/settings/wcm/templates/<my-template>`)
- Model—The OSGI bundles implement the functionality.
- View—On both the author environment and publish environment the content is rendered by components (`/apps/<my-site>/components`).

When rendering a page:

- The **cq:template** property of its **jcr:content** node will be referenced to access the template that corresponds to that page.
- The page component will merge the **structure/jcr:content** tree of the template with the **jcr:content** tree of the page.
- The page component will allow the author to edit only the nodes of the template structure that are flagged as editable.
- The relative path of the component will be taken from the jcr:content node when rendering a component on a page.
- The **cq:policy** property of the component:
 - Points to the actual content policy (holds the design configuration for that component).
 - Helps you have multiple templates that re-use the same content policy configurations.

Exercise 5: Create a content root

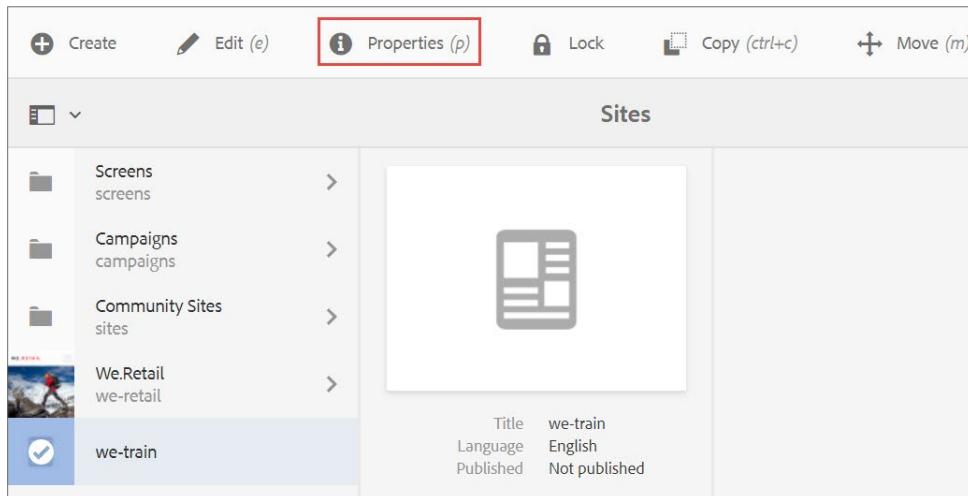
In this exercise, you will create a content root for the website. The content root will help you use the Content Page template when creating pages.

To create a content root:

1. Open a new tab of your browser where the AEM author instance is running.
2. Enter the URL <http://localhost:4502/crx/de/index.jsp> in the address bar of the browser, and then press Enter from the keyboard. The **CRXDE Lite** page opens.
3. Select the **/content** folder, click **Create** from the actions bar, and select **Create Node** from the drop-down menu. The **Create Node** dialog box opens.
4. Enter and select the following:
 - a. **Name:** **we-train**
 - b. **Type:** Select **cq:Page** from the drop-down menu.
5. Click **OK**.
6. Click **Save All** from the actions bar.
7. Select the **we-train** node, click **Create** from the actions bar, and select **Create Node** from the drop-down menu. The **Create Node** dialog box opens.
8. Enter and select the following:
 - a. **Name:** **jcr:content**
 - b. **Type:** Select **cq:PageContent** from the drop-down menu
9. Click **OK**.
10. Click **Save All** from the actions bar.
11. Select the **jcr:content** node that is below the **we-train** node, add the following property:

Field	Type	Value
sling:resourceType	String	training/components/structure/page

12. Open a new tab of your browser where the AEM author instance is running.
13. Click **Adobe Experience Manager > Navigation**, and then click **Sites**. The **Sites** console with **We.Train** page opens.
14. Click the thumbnail icon beside the **we-train** page and click **Properties** as shown. The **Properties** wizard opens.



15. Ensure you are in **Basics** tab and enter **We.Train** in the **Title** field.
16. Click the **Advanced** tab:
 - a. Enter **/content/we-train/en** in the **Redirect** field.
 - b. Scroll down and look for **Allowed Templates**, and click the **Add** button.
 - c. Enter **/conf/we-train/settings/wcm/templates/*** in the field.
17. Click **Save & Close**. The **form has been submitted successfully** message appears.

To view the properties that you added to content root in JCR:

18. Open a new tab of your browser where the AEM author instance is running.
19. Enter the URL <http://localhost:4502/crx/de/index.jsp> in the address bar in the browser, and then press Enter from the keyboard. The CRXDE lite page opens.
20. Navigate to the **/content/we-train/jcr:content** node and observe the properties that you just added to the content root.

Exercise 6: Create a site structure

You have enabled and published the Content Page template, and created a content root. Next, you can create your site structure.

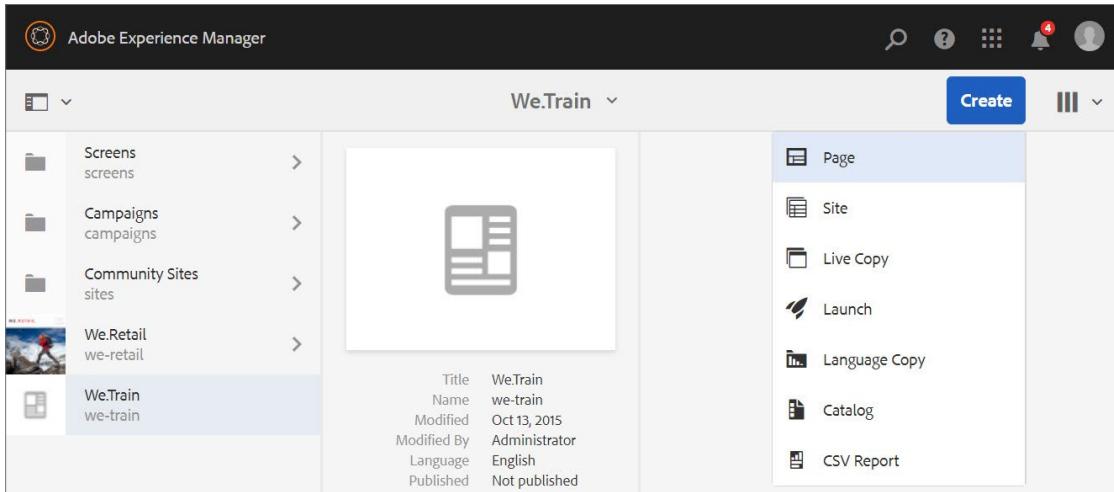
In this exercise, you will create the following website structure:

```
--> We.Train  
-----> English  
-----> About Us  
-----> Experiences  
-----> Equipment  
-----> Activities  
-----> Hiking  
-----> Biking  
-----> Running  
-----> Skiing  
-----> Climbing  
-----> French
```

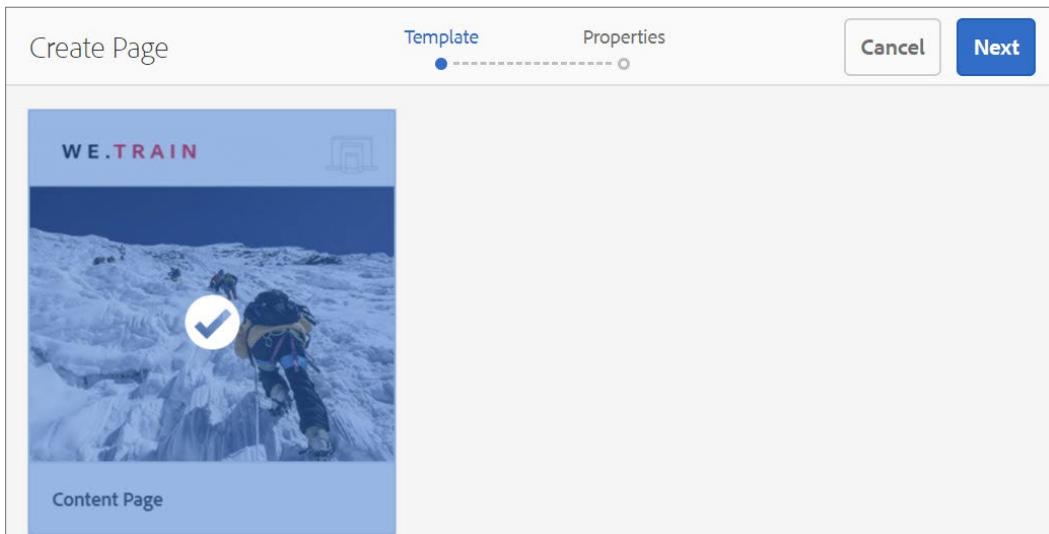
To create a page:

1. Open a new tab of your browser where the AEM author instance is running.
2. Enter the URL <http://localhost:4502/sites.html/content> in the address bar, in the browser and then press Enter from the keyboard. The **Sites** console with **We.Train** page opens.

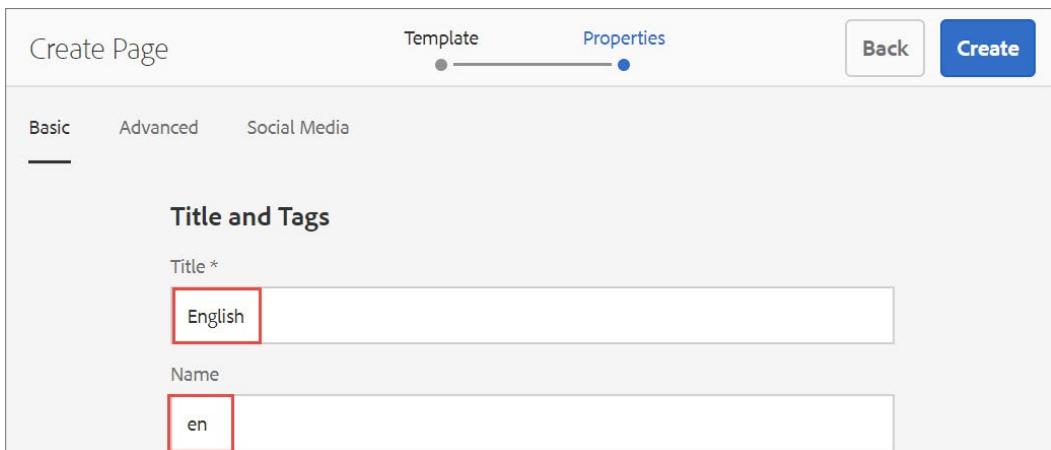
3. Select the **We.Train** page, click **Create** from the actions bar, and select **Page** from the drop-down menu as shown. The **Create Page** dialog box opens.



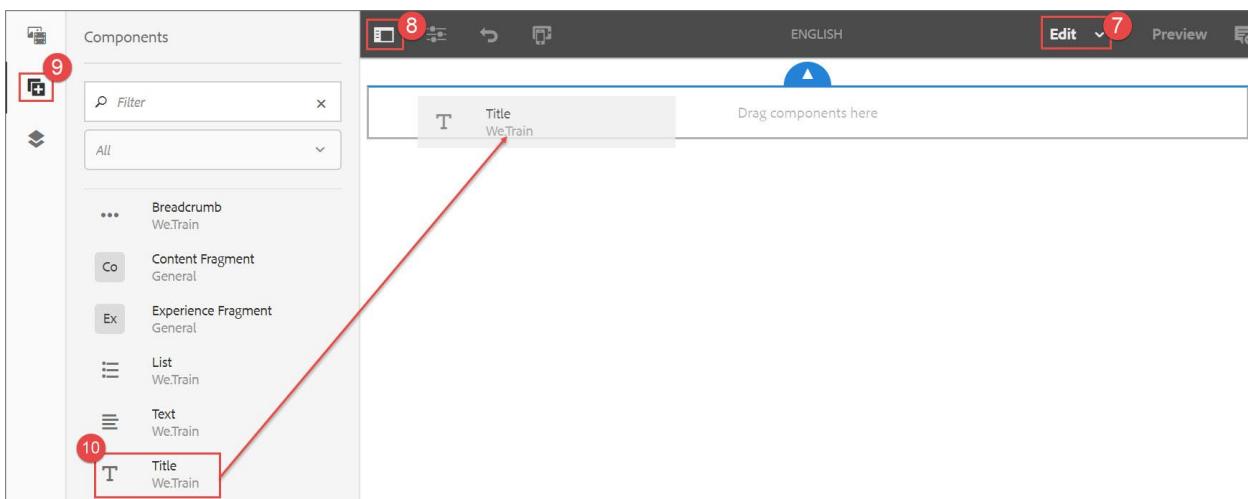
4. Select the **Content Page** template and click **Next** as shown. The **Properties** wizard opens.



5. Enter **English** in the **Title** field and **en** in the **Name** field, and then click **Create** as shown. The **Success** dialog box appears.



6. Click **Open**. The page opens in a new tab of your browser.
7. Ensure the **Edit** button is selected in the page toolbar.
8. Click the **Toggle Side Panel** icon from the page toolbar. The side panel opens.
9. Click the **Components** icon, all available components for the page appear in Components browser.
10. Drag the **Title** component onto the **Drag components here** area on the page as shown. The **Title** component is added to the page.



To create a subpage:

11. Ensure you are in **Sites** console
12. Navigate through the **We.Train > English** page.
13. Click **Create** from the actions bar and select **Page** from the drop-down menu as shown. The **Create Page** dialog box opens.

The screenshot shows the Adobe Experience Manager (AEM) interface. At the top, there's a navigation bar with icons for search, help, and user profile. Below it is a header bar with the text "Adobe Experience Manager" and "English". On the left, there's a sidebar with links to "Screens", "Campaigns", "Community Sites", "We.Retail", and "We.Train". The main area shows a tree structure under "English" with a thumbnail for "English en". To the right of the tree, there's a "Create" button followed by a dropdown menu. The dropdown menu has several options: "Page" (selected), "Site", "Live Copy", "Launch", "Language Copy", "Catalog", and "CSV Report". A tooltip on the "Page" option indicates it was created 4 minutes ago by a system administrator.

14. Select the **Content Page** template and click **Next**, as shown. The **Properties** wizard opens.
15. Enter **About Us** in the **Title** field and **about-us** in the **Name** field, and then click **Create**. The **Success** dialog box opens.
16. Click **Done**. The subpage of **English** is created as shown:

This screenshot shows the same AEM interface as the previous one, but now with a new subpage. The "English" site structure now includes a node for "About Us" with the name "about-us". The thumbnail for this subpage shows a person in a dynamic pose. The rest of the interface remains the same, with the sidebar, header, and "Create" dropdown visible.

17. Perform steps 11 through 16 and create the below website structure with the following naming conventions (where, T stands for Title and N for Name):

```
-----> English (T) (N: en)
-----> About Us (T) (N: about-us)
-----> Experiences (T) (N: experiences)
-----> Equipment (T) (N: equipment)
-----> Activities (T) (N: activities)
-----> Hiking (T) (N: hiking)
-----> Biking (T) (N: biking)
-----> Running (T) (N: running)
-----> Skiing (T) (N: skiing)
-----> Climbing (T) (N: climbing)
-----> French (T) (N: fr)
```



Note: It is a best practice to name all your pages and subpages in lowercase and use hyphens for two or more words.

After, your site structure is complete, it should look similar to the site shown in the following screenshot:

The screenshot shows the Adobe Experience Manager (AEM) Sites console. The left sidebar lists several items: Screens (screens), Campaigns (campaigns), Community Sites (sites), We.Retail (we-retail), and We.Train (we-train). The main content area is titled "Activities" and shows a tree structure of pages. At the top level, there are links for "English en" and "French fr". Under "English en", there are links for "About Us about-us", "Experiences experiences", "Equipment equipment", and "Activities activities". Under "French fr", there are links for "Hiking hiking", "Biking biking", "Running running", "Skiing skiing", and "Climbing climbing". Each page item has a small thumbnail image to its left.

Optional Exercise

The content root page (/content/we-train) does not have a thumbnail in the Sites console. Add a thumbnail to the page.

Hint: Select the We.Train page, click **Properties**, and upload an image from the **Thumbnail** tab.



Creating Client-Side Libraries

Introduction

Modern websites rely heavily on client-side processing driven by complex JavaScript (JS) and the Cascading Style Sheets (CSS) code. Organizing and optimizing the code can be a complicated task. Adobe Experience Manager (AEM) provides client-side library folders to store client-side code in the repository, organize them into categories, and define when and how each category of code can be served to the client. The client-side library system helps produce the correct links on the final webpage to load the correct code.

Objectives

After completing this module, you should be able to:

- Explain how the client-side libraries work in AEM
- Explain the client libraries' structure
- Organize client-side libraries
- Explain how to reference client-side libraries
- Create a client-side library
- Add a client-side library to a page component
- Add a client-side library to a template

Client-Side Libraries

The standard way to include a client-side library (a JS or a CSS file) in the HTML of a page is to include a <script> or <link> tag in the Java Server Page (JSP), containing the path to the JS/CSS file. Although this approach works in AEM, it can lead to problems when pages and their constituent components become complex. In such cases, multiple copies of the same JS library may be included in the final HTML output.

To avoid this issue, AEM uses client-side library folders to organize the client-side libraries logically. The basic goals of client-side libraries are to:

- Store CSS/JS in small discrete files for easier development and maintenance
- Manage dependencies on third-party frameworks in an organized fashion
- Minimize the number of client-side requests by concatenating CSS/JS into one or two requests
- Minimize CSS/JS that is delivered to optimize speed/performance of a site

Structure of Client-Side Libraries

A client-side library folder is a repository node of type **cq:ClientLibraryFolder**. Its definition in Compact Namespace and Node Type Definition (CND) notation is:

```
[cq:ClientLibraryFolder] - jcr:primaryType
  - categories (String[])
  - embed (String[])
  - dependencies (String[])
  + css.txt (nt:file)
  + js.txt (nt:file)
```

Each cq:ClientLibraryFolder is populated with a set of JS and/or CSS files, along with a few supporting files. You can configure cq:ClientLibraryFolder through the following properties:

- categories
- dependencies
- embed
- allowProxy

categories

This property helps identify the categories into which the set of JS and/or CSS files within this cq:ClientLibraryFolder belong. The categories property is multi-valued and allows a library folder to be part of more than one category.

dependencies

This property provides a list of other client library categories on which this library folder depends. For example, given two cq:ClientLibraryFolder nodes F and G, if a file in F requires another file in G in order to function properly, then at least one of the categories of G should be among the dependencies of F.

embed

This property is used to embed code from other libraries. If node F embeds nodes G and H, the resulting HTML will be a concatenation of content from nodes G and H.

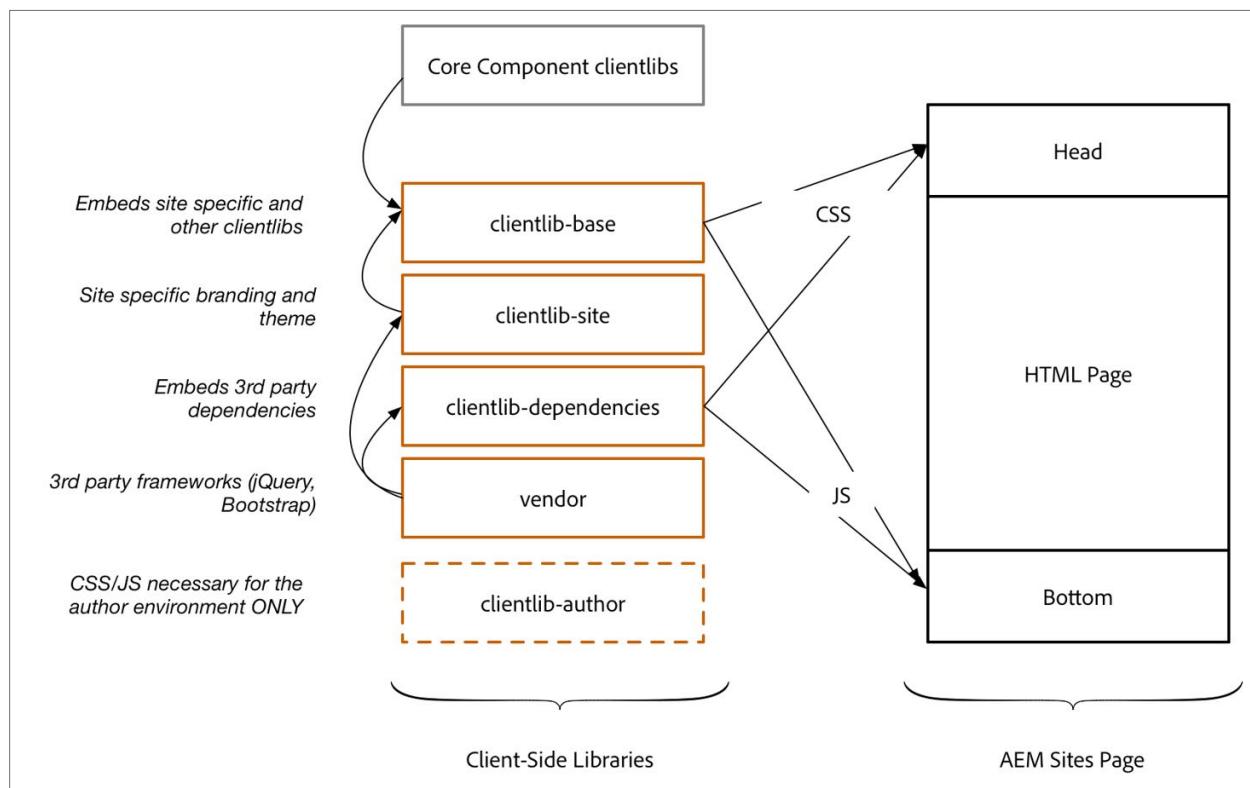
allowProxy

If a client library is located under /apps, this property allows access to it through the proxy servlet.

js.txt/css.txt

This file helps identify the source files to merge in the generated JS and/or CSS files.

The below diagram explains the client-side libraries' convention followed in the We.Retail site (that is available out-of-the-box in AEM) and can be applied to most sites' implementations:



Organizing Client-Side Libraries

By default, the cq:ClientLibraryFolder nodes can exist anywhere within the /apps and /libs subtrees of the repository. The common locations for clientlibs are:

- Site-level (/apps/<your-project>/clientlibs)
- Component-level (/apps/<your-project>/components)

Site-Level

The clientlibs in site-level:

- Are used for CSS/JS for site specific design elements
- Have CSS at the top of page and JS at the bottom of page
 - Examples include responsive design, dependencies, embedding structure components, and vendor code

Component-Level

The clientlibs in the component-level:

- Are component-specific CSS/JS
- Can be embedded into the site design

Referencing Client-Side Libraries

You can include the client-side libraries in HTML Template Language (HTL), which is the preferred technology for developing AEM sites. However, you can also use JSP.

In HTL, client-side libraries are loaded through a helper template provided by AEM. You can access the template through `data-sly-use`. Three templates are available, these templates `css`, `js`, and `all` can be called through `data-sly-call`:

- `css`: Loads only the CSS files of the referenced client libraries.
- `js`: Loads only the JavaScript files of the referenced client libraries.
- `all`: Loads all files of the referenced client libraries (both CSS and JavaScript).

Each helper template expects a `categories` option for referencing the desired client libraries. This option can be either an array of string values, or a string containing a comma-separated values list.

Exercise 1: Create a client-side library

In this exercise, you will create a base client-side library. This base library will help compile the sites css/js into a single client library.

You will install the **adls-training-project-v3.0** package on your AEM author instance. The package contains a project structure with components, templates, and a site structure that has already been created for you.

To create a base client-side library:

1. Click **Adobe Experience Manager** from the header bar, and click the Tools icon. The **Tools** console opens.
2. Click **CRXDE Lite**. The CRXDE Lite page opens.
3. Navigate to the **/apps/training** folder.
4. Select the **training** folder, click **Create** from the actions bar, and then click **Create Folder** from the drop-down menu. The **Create Folder** dialog box opens.
5. Enter **clientlibs** in the **Name** field, and click **OK**. The folder is created under the training folder.
6. Click **Save All** from the actions bar.



Note: You can also press Ctrl+S (Windows users) and Command+S (Mac users) to save your work in CRXDE Lite.

7. Select the **clientlibs** folder, click **Create** from the actions bar, and click **Create Node** from the drop-down menu. The **Create Node** dialog box opens.
8. Perform the following:
 - a. **Name:** **clientlib-base**
 - b. **Type:** Select **cq:ClientLibraryFolder** from the drop-down menu.
9. Click **OK**. The folder is created.
10. Click **Save All** from the actions bar.

11. Select the **client-base** folder, and add the following properties:

Field	Type	Value
categories	String[]	we.train.base
embed	String[]	[we.train.site, we.train.font-awesome]
allowProxy	Boolean	true



Note: To enter a String Array (String[]), select **String** from **Type** field, and then click the **Multi** button.

Remember, the **Multi** button will be selected until you deselect it. If a dialog box appears with the string array information, click **OK**.

12. Click **Save All** from the actions bar.
13. Select the **clientlibs** folder, click **Create** from the actions bar, and select **Create File** from the drop-down menu. The **Create File** dialog box opens.
14. Enter **css.txt** in the **Name** field, and click **OK**. The file is created under the clientlibs folder.
15. Click **Save All** from the actions bar.
16. Perform step 13 through 15 and create the **js.txt** file.
17. Click **Save All** from the actions bar.

Next, you will upload the **we-train-design-package** to JCR. This package contains the css/js that you will use in the We.Train site structure.

To upload and install the we-train-design-package:

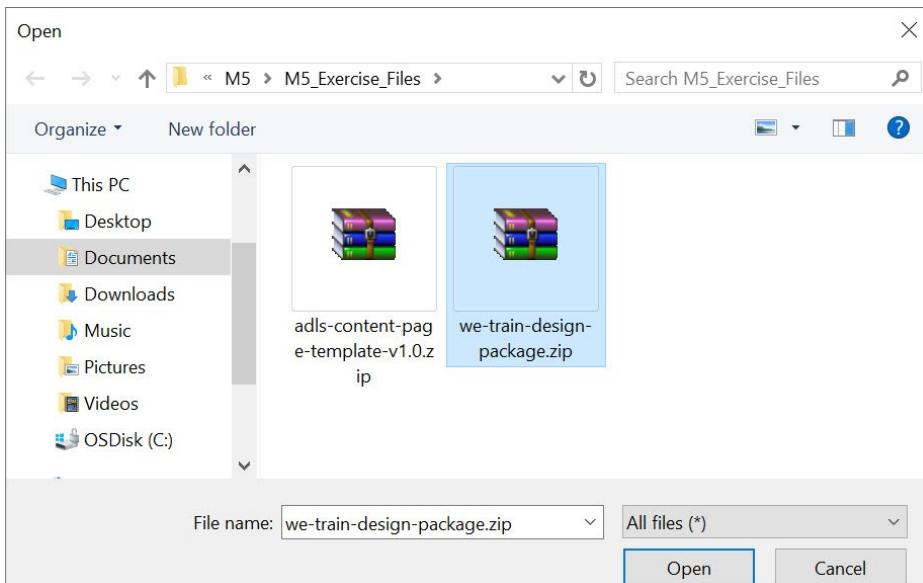
18. From CRXDE Lite, click the Package icon from the header bar as shown. The **Package Manager** page opens.



19. Click **Upload Package** from the actions bar, as shown. The **Upload Package** dialog box opens.

20. Click **Browse**.

21. Navigate to the **Exercise_Files** folder on your file system, double-click to open the folder, select the **we-train-design-package.zip** package, and then click **Open** as shown. The **Upload Package** dialog box opens.



22. Click **OK**. The package is uploaded.

23. In the **we-train-design-package.zip** section, click **Install**, as shown. The **Install Package** dialog box opens. Notice the filters on the package. This shows the files and folders that will be installed into your **/apps/training/clientlibs** folder on JCR.

Package:	we-train-design-package
Download:	we-train-design-package.zip (1.2 MB)
Group:	my_packages
Filters:	/apps/training/clientlibs/clientlib-site /apps/training/clientlibs/custom-lib /apps/training/clientlibs/vendor

24. Click **Install**. The package is installed on your instance.

25. Click the Develop icon from the header bar as shown. The **CRXDE Lite** page opens.



26. Navigate to the **/apps/training/clientlibs** folder and observe the client libraries that are installed by the package.

Exercise 2: Add a client-side library to a page component

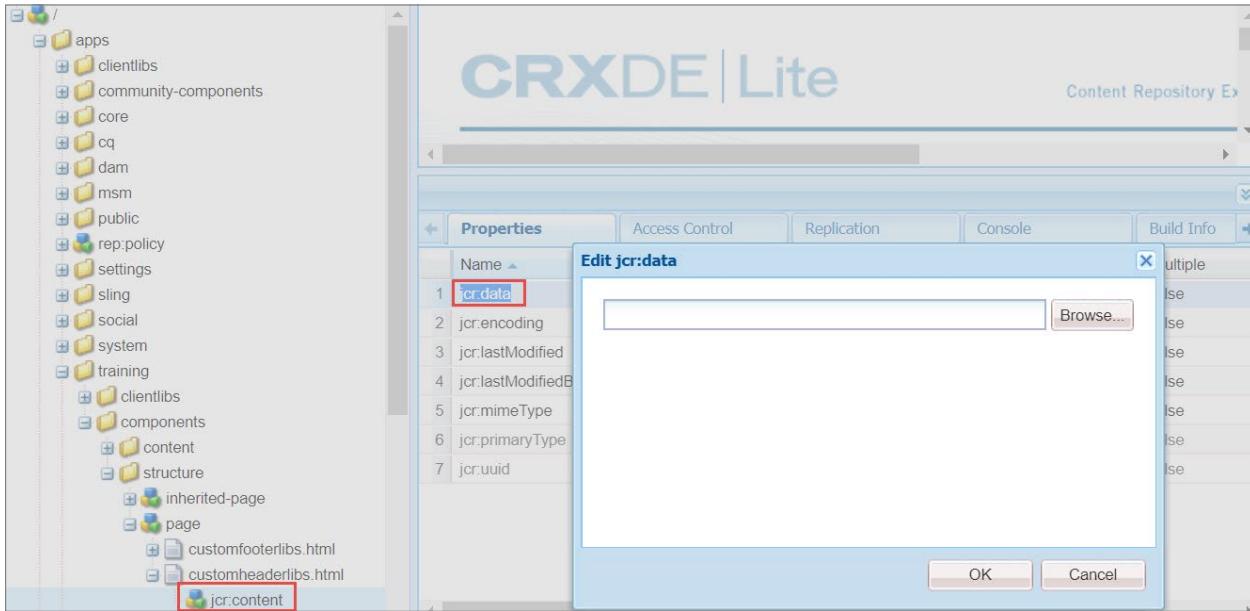
To include the client libraries on the page, you will overlay the `customheaderlibs.html` and `customfooterlibs.html` from the core page component.

To add the above files to JCR:

1. Open a new tab of your browser where the AEM author instance is running.
2. Add the URL <http://localhost:4502/crx/de/index.jsp> in the address bar, and then press Enter from the keyboard. The **CRXDE Lite** page opens.
3. Navigate to the `/apps/training/clientlibs/clienlib-base` node. Observe the property `categories=we.train.base`. You will use this category name when adding the clientlib to a page component.
4. Navigate to the `/apps/training/components/structure` folder.
5. Select the **page** node, click **Create** from the actions bar, and select **Create File** from the drop-down menu. The **Create File** dialog box opens.
6. Enter **customheaderlibs.html** in the **Name** field, and click **OK**. The file is created.
7. Click **Save All** from the actions bar.
8. Perform step 4 and 7 and create a **customfooterlibs.html** file.
9. Click **Save All** from the actions bar.

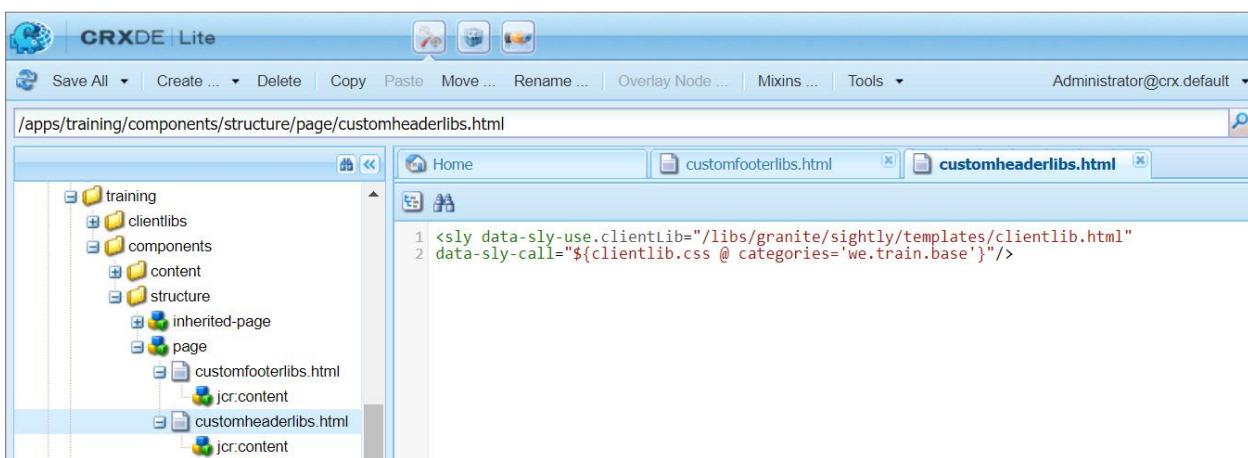
To add code to the .html files:

10. Select the **jcr:content** node that is below **customheaderlibs.html**. The **Properties** tab opens on the left side panel.
11. Double-click the **jcr:data** property as shown. The **Edit jcr:data** dialog box opens.



12. Click **Browse**, as shown. The **Open** dialog box opens.
13. Navigate to the **Exercise_Files** folder on your file system.
14. Select **customheaderlibs.html**, and then click **Open**.
15. Click **OK** in the **Edit jcr:data** dialog box. The code is added to **customheaderlibs.html**.

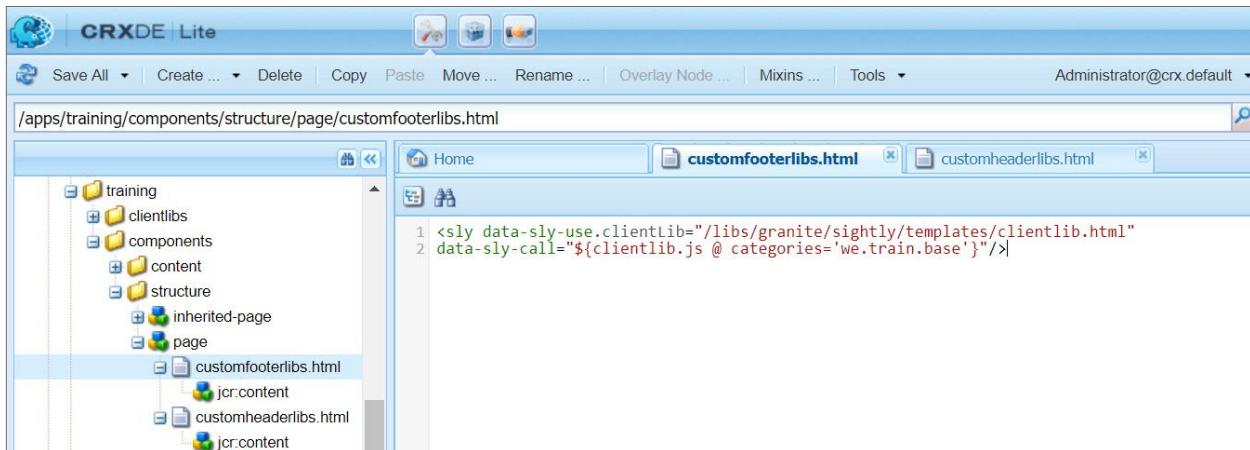
16. Open the **customheaderlibs.html** to view new code that you added as shown:



17. Perform steps 10 through 16 and add code to the **customfooterlibs.html** file, making sure to select the corresponding **customfooterlibs.html** (the **Exercise_Files** folder on your file system) for the **jcr:data**.

18. Click **Save All**.

19. Open the **customfooterlibs.html** to view new code that you added as shown:



You can verify if the css and javascript client libraries are added to the top and bottom of the page respectively, from the HTML source of a page.

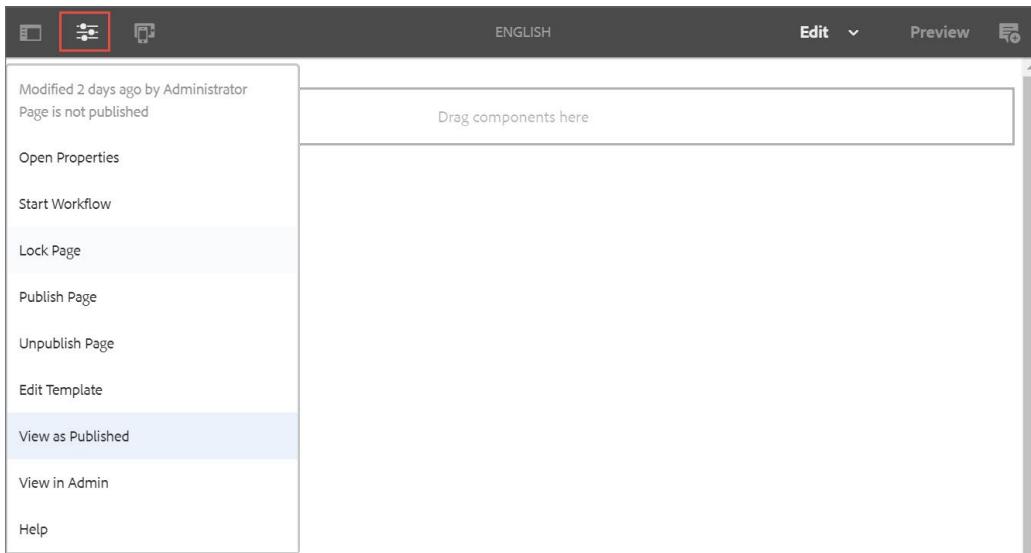
20. Click **CRXDE Lite** from the header bar. You will be taken to the **Navigation** pane.

21. Click **Sites**. This will open the **Sites** console.

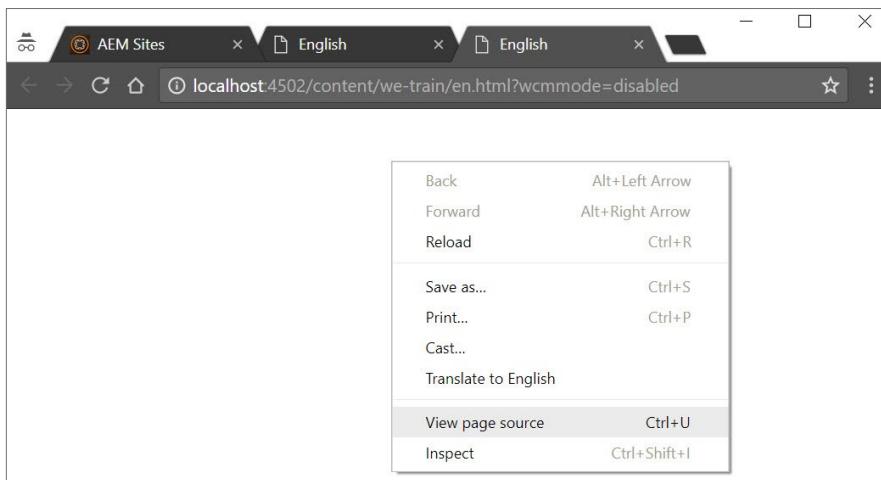
22. Navigate through the **We.Train** site, click the thumbnail on the **English** page, and then click **Edit (e)** from the actions bar. The English page opens in a new tab.

You will not see any visual changes in the English page. However, you can see the clientlibs.css (added by the customheaderlibs.html) and clientlibs.js (added by the customfooterlibs.html) are referenced in the page by viewing the source code of the page.

23. Click the Page Information icon from the page toolbar, and click **View as Published** from the drop-down menu as shown. The page opens in a new tab of the browser.



24. Right-click in the English page, and click **View page source** from the list as shown. The source code opens in a new tab of the browser.



25. Look for the line, `<link rel="stylesheet" href="/etc.clientlibs/training/clientlibs/client-base.css" type="text/css">` in the source code. Notice how the **href** tag points to the clientlibs path in JCR that confirms the customheaderlibs.html is referenced in the page as shown:

```
1 <!DOCTYPE HTML>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <title>English</title>
6
7
8
9     <meta name="template" content="content-page"/>
10
11
12
13 <link rel="stylesheet" href="/etc.clientlibs/training/clientlibs/client-base.css" type="text/css">
14
15
```

26. Look for the line, `<script type="text/javascript" src="/etc.clientlibs/training/clientlibs/client-base.js"></script>` in the source code. Notice how the **href** tag points to the clientlibs path in JCR that confirms the customfooterlibs.html referenced in the page as shown:

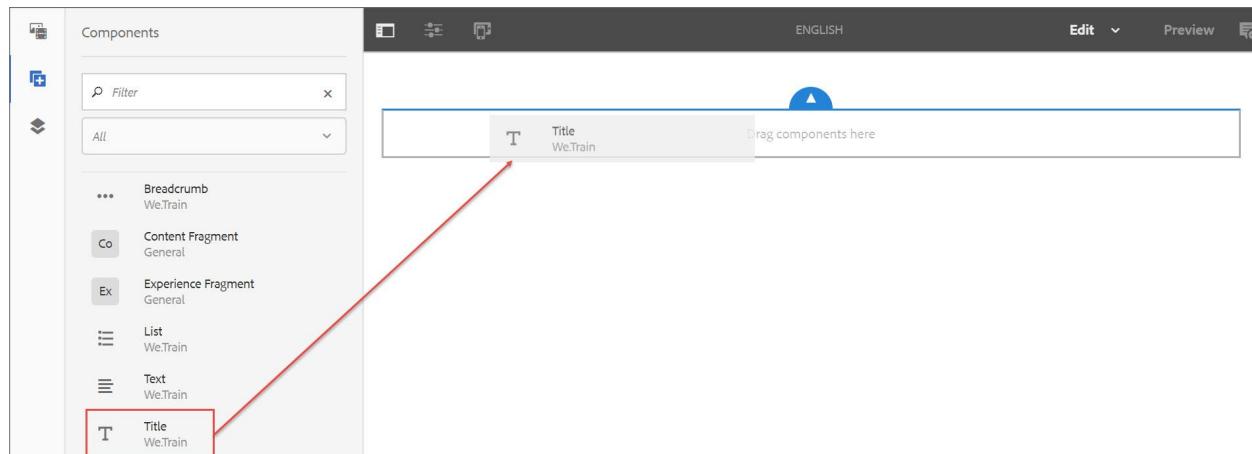
```
66
67
68
69
70
71
72
73
74 <script type="text/javascript" src="/etc.clientlibs/training/clientlibs/client-base.js"></script>
75
76
77
78
79
80
81
82
83
84
85
86     </body>
87 </html>
88
```

Exercise 3: Add a client-side library to a template

In the previous exercise, you learned how a developer can code in the client libraries directly in the header and footer of a page. If the template editor is adding client libraries to the page, they can do this through the Template Editor. This approach can be useful when page templates use the exact same components and layout, but templates need to have a different design for microsites and multiple tenants with a single development team.

To add clientlibs to the template:

1. Open a new tab of your browser where the AEM author instance is running.
2. Click **Adobe Experience Manager** from the header bar. The **Navigation** pane opens.
3. Click **Sites**. The **Sites** console opens.
4. Navigate to the **We.Train** site, click the thumbnail on the **English** page.
5. Click **Edit** from the actions bar. The **English** opens on a new tab.
6. Click the Toggle Side Panel icon from the page toolbar.
7. Click Components icon from the panel. The components list appears.
8. Drag the **Title** component from the panel, drop it onto **Drag components here** placeholder on the page as shown. The component displays the title of the page.

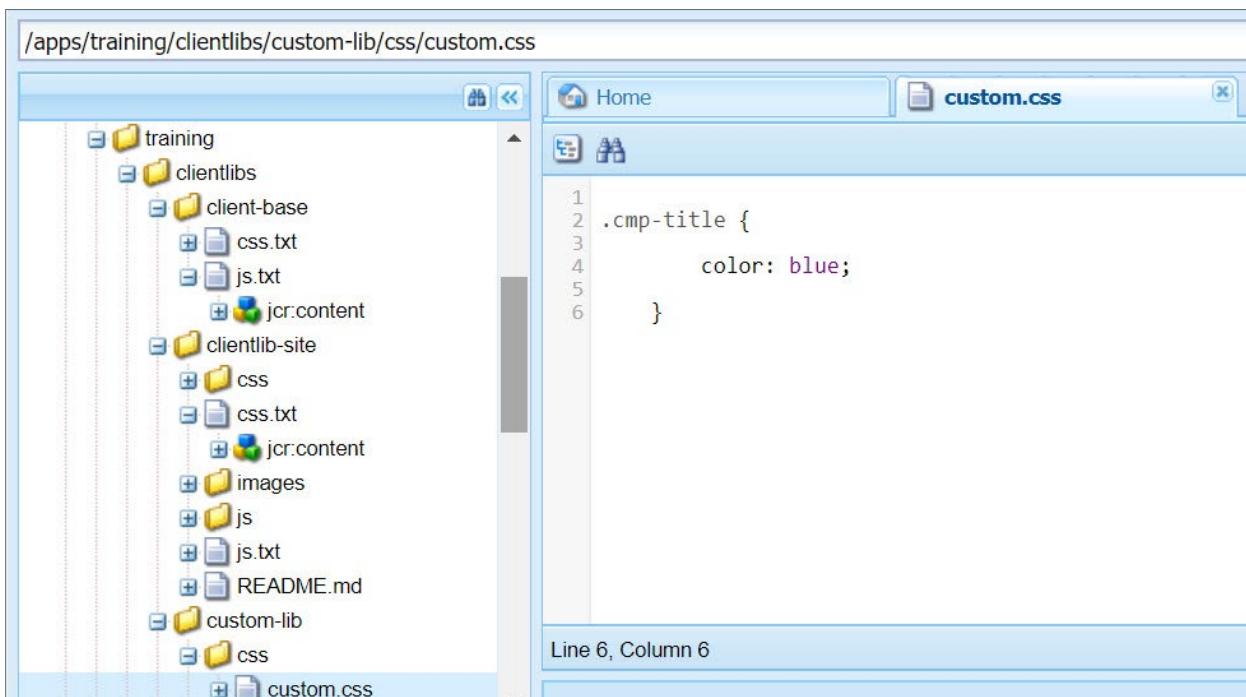


Notice, how the title is currently black as shown:



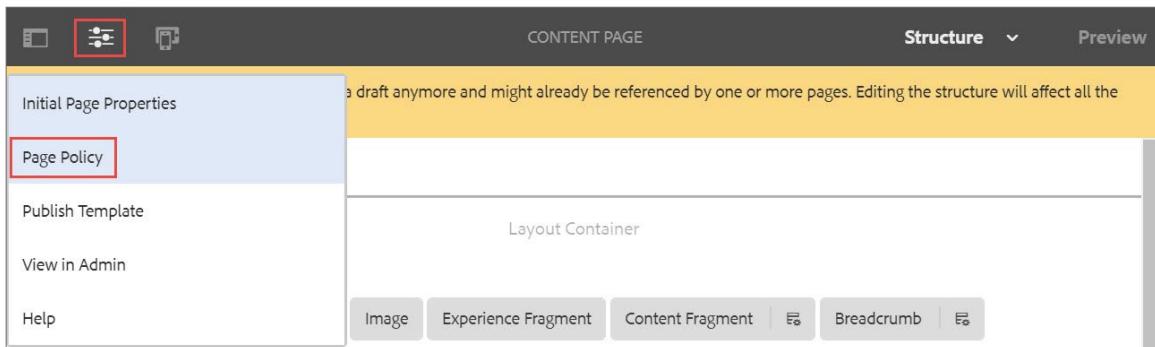
In the We.Train website, let's change the title color to blue by adding the clientlibs.

9. Open a new tab of your browser where the AEM author instance is running.
10. Add the URL <http://localhost:4502/crx/de/index.jsp> in the address bar, and then press Enter from the keyboard. The CRXDE Lite page opens.
11. Navigate to the `/apps/training/clientlibs/custom-lib` node. Observe the property `categories=we.train.custom.lib`. You will use this category when adding the clientlib to the template.
12. Navigate to the `/apps/training/clientlibs/custom-lib/css/custom.css` folder. Notice, the `custom.css` has the css to change the title color to blue, as shown. This client library was added when you installed the `we-train-design-package` earlier.



To add a client library to a template:

13. Navigate to the tab of the browser where the **English** page of We.Train site is open.
14. Click the Page Information icon from the page toolbar, and click **Edit Template** from the drop-down menu. The template opens on a new tab.
15. Ensure you are in the content page template, click the Page Information icon from the toolbar, and click **Page Policy** from the drop-down menu as shown. The **Page Content Policy** opens.



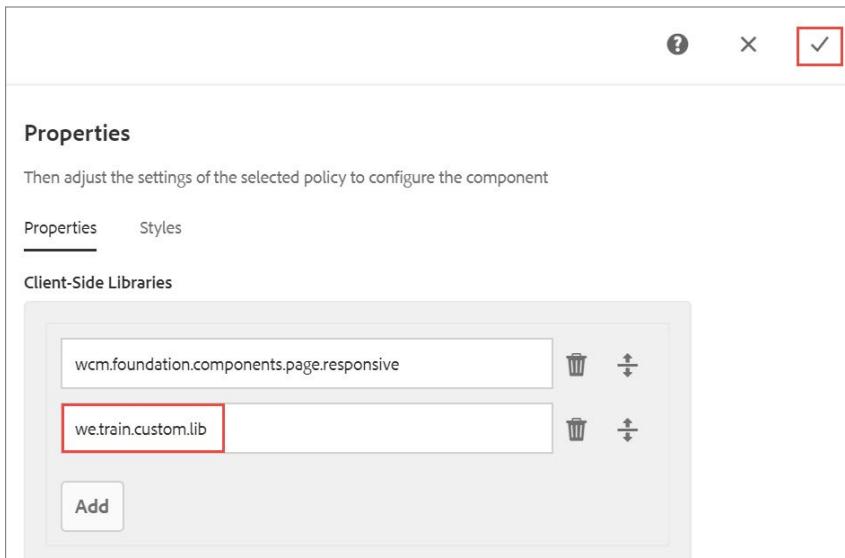
16. In **Policy** section, enter **Custom Clientlibs** in the **Policy Title** field.
17. On the right side, in the **Properties** section, notice the clientlib, **wcm.foundation.components.page.responsive** is already added to the template, as shown. It is the default, AEM responsive grid. Notice, you can add new client libraries to the page template

A screenshot of the 'Page Content Policy' dialog. The left panel, titled 'Policy', shows a dropdown for 'Select policy' set to 'New policy', a 'Policy Title' input field containing 'New policy', and a 'Policy Description' input field containing 'Add a description'. The right panel, titled 'Properties', has a 'Properties' tab selected. Under 'Client-Side Libraries', there is a list box containing 'wcm.foundation.components.page.responsive' with a delete icon. Below this is a 'Web Resources Client Library' section with 'org.example.myapp.resources'. A red box highlights the 'Client-Side Libraries' list box.

Now, let's add the custom clientlib to the template.

18. Click **Add**, and enter **we.train.custom.lib** in the field.

19. Click the checkmark (Done) icon as shown. The custom clientlib is added to the template.

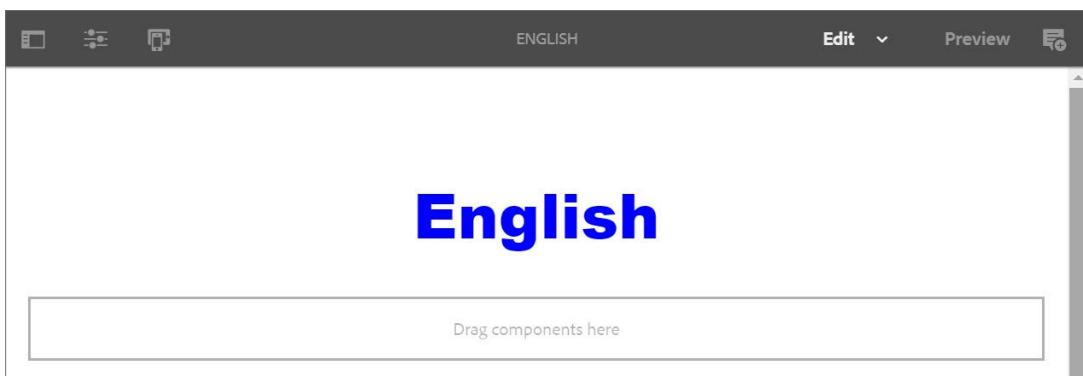


 **Note:** The Content Page template is already enabled, which means the changes you make affect all pages that were built from the template.

 **Caution:** The design changes you made are only on the author instance. If you want this change to be replicated to the publish instance, you must republish this template.

20. Navigate to the tab of the browser where the **English** page of the **We.Train** site is open.

21. Refresh the browser. Notice how the title color changed to blue as shown:



To verify that the client library is added to the page, you can view the source of the page.

22. Click the Page Information icon from the page toolbar, and click **View as Published** from the drop-down menu, as shown. The page opens in a new tab of the browser.
23. Right-click in the **English** page, and click **View page source** from the list, as shown. The source code opens in a new tab of the browser.
24. Look for the line, `<link rel="stylesheet" href="/etc.clientlibs/training/clientlibs/custom-lib.css" type="text/css">` in the source code. Notice how the **href** tag points to the **clientlibs** path in JCR that confirms the custom clientlibs is added to the page as shown:

```
21  
22  
23  
24  
25  
26 <link rel="stylesheet" href="/libs/wcm/foundation/components/page/responsive.css" type="text/css">  
27 <link rel="stylesheet" href="/etc.clientlibs/training/clientlibs/custom-lib.css" type="text/css">  
28  
29  
30  
31  
32  
33  
34 </head>  
35   <body class="page basicpage">  
36
```

The screenshot shows the raw HTML source code of a page. Lines 26 and 27 contain the following code:
<link rel="stylesheet" href="/libs/wcm/foundation/components/page/responsive.css" type="text/css">
<link rel="stylesheet" href="/etc.clientlibs/training/clientlibs/custom-lib.css" type="text/css">
A red callout bubble points to the second line with the text "We.Train custom clientlib". Another red callout bubble points to the first line with the text "Default responsive grid clientlib".



Working with Components

Introduction

Adobe Experience Manager (AEM) provides a wide range of component implementations on a standard instance by default. Before developing components in AEM, you need to understand the key concepts of a component, its structure, and the configuration.

Objectives

After completing this module, you should be able to:

- Explain the basics of a component
- Explain the HTL business logic
- Create a basic header component
- Add JavaScript business logic to a header component
- Add Sling Model business logic to a header component
- Explain AEM dialogs
- Create an edit dialog for the component
- Add an editconfig node to the component
- Explain component client-side library
- Add a client-side library to the component
- Add a cq:htmlTag to the component
- Explain AEM design dialogs
- Create a content policy for a proxy component
- Create a design dialog for a component

Component Basics

In AEM, a component is a resource type. It contains a collection of scripts that implement a specific functionality to present the content on your website.

Components:

- Are modular and reusable units
- Are developed as self-contained units within one folder of the repository
- Have no hidden configuration files
- Can contain other components
- Can run anywhere within any AEM system
- Have a standardized user interface
- Have configurable edit behavior
- Use dialogs that are built using sub-elements such as Granite UI components for the touch UI and widgets
- Can be developed using HTML Template Language (HTL) (recommended) or Java Server Pages (JSP)

AEM provides the core components that offer flexible and feature-rich authoring functionality.

Component List

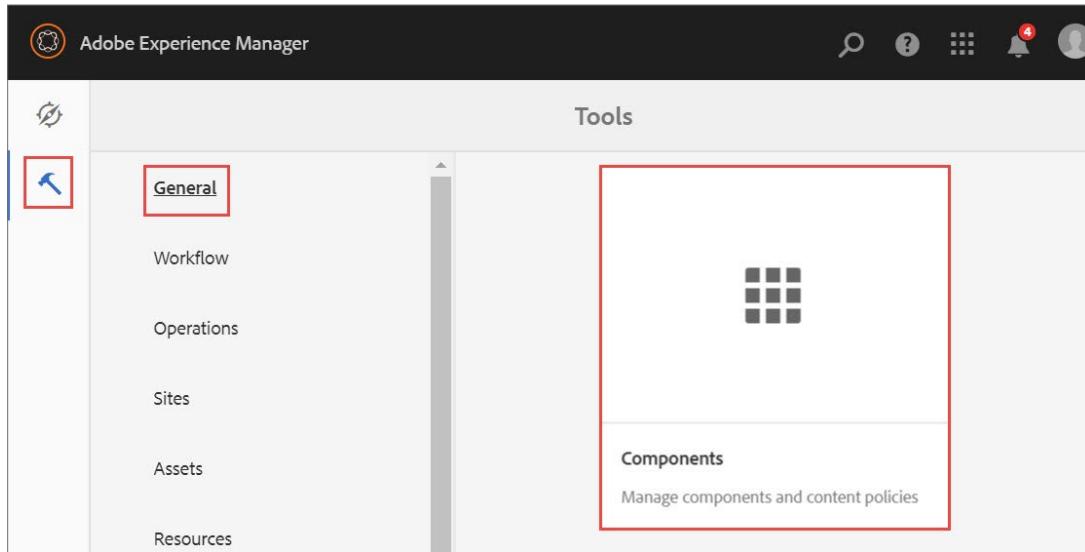
The following table describes the available core components:

Core Component	Description
Page	Responsive page working with the template editor
Breadcrumb	Page hierarchy navigation
Title	H1-H6 title
Text	Rich text
Image	Smart and lazy loading of optimal rendition size
List	List of pages
Social Media Sharing	Facebook- and Pinterest- sharing widget
Form Container	Responsive form paragraph system
Form Text	Text input field
Form Options	Multiple options input field
Form Hidden	Hidden input field
Form Button	Submit or custom button
Navigation	A site navigation component that lists the nested page hierarchy
Language Navigation	A language and country switcher that lists the global language structure
Quick Search	A search component that displays the results as real-time suggestions in a drop-down menu

 **Note:** Core components are not immediately available to authors. The development team must first integrate them into the AEM author environment, and then preconfigure the core components from the template editor to make them available for authors.

Components Console

You can view all available components on your AEM instance from the **Components** console. You can access the **Components** console from **Tools > General** section as shown:



In the Components console, each component has an icon, a title, the group to which the component belongs, and its resource type as shown:

Components			
	Title	Group	Resource Type
3r	3rdparty	.hidden	/libs/cq/gui/components/cloudservices/admin/3rdparty
Gm	Gmail	.hidden	/libs/mcm/components/newsletter/emailclient/Gmail
Ho	Hotmail	.hidden	/libs/mcm/components/newsletter/emailclient/Hotmail
Ou	Outlook 2007	.hidden	/libs/mcm/components/newsletter/emailclient/Outlook2007
Ya	YahooMail	.hidden	/libs/mcm/components/newsletter/emailclient/YahooMail
Ab	Abandoned Product Call To Action	Commerce	/libs/commerce/components/calltoaction/abandoned

Structure of a Component

The most important elements of the structure are:

- Root node
- Vital properties
- Vital child nodes

Root Node

It is the hierarchy node of the component and represented as node <mycomponent> (**type cq:Component**).

Vital Properties

The vital properties of a component are:

- jcr:title: Is the component title; for example, this property is used as a label when the component is listed in the components browser.
- jcr:description: Is the description for the component. You can use this property as a mouse-over hint in the components browser.
- cq:icon: Is the string property pointing to a standard icon in the Coral UI library to display in the component browser.

Vital Child Nodes

The vital child nodes of a component are:

- cq:editConfig (cq:EditConfig): Defines the edit properties of the component and enables the component to appear in the Components browser
- cq:childEditConfig (cq:EditConfig): Controls the author UI aspects for child components that do not define their own cq:editConfig
- cq:dialog (nt:unstructured): Is the dialog for this component and defines the interface allowing the user to configure the component and/or edit content.
- cq:design_dialog (nt:unstructured): Helps in editing the design of a component



Note: Refer to the following link for more information on the structure of the component:

<https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/components-basics.html#Structure>

Structure Components versus Content Components

The content of a page is distributed into layout and page-specific categories. You can design the page layout with a set of components called *structure components*. The content that is unique to the page is added through *content components*.

Authors must follow a few guidelines when working with structure and content components:

- Template authors can add and remove the structure components to the template.
- Authors cannot remove structure components from the page.
- Authors can edit and configure structure components of the page.
- Authors can add and remove content components from the page.

HTL Business Logic

HTML Template Language (HTL) is HTML5 that helps develop and design business logic in HTL completely. HTL helps add more complex business logic to the code through the following options:

- Sling Models
- Server-Side JavaScript
- WCMUsePojo Java Class

Each method exposes global objects that can be used in the business logic. You can call the getters [get() method] in HTL to expose the output of the above methods. The following HTL statement is used for invoking the business logic:

```
<div data-sly-use.myObj="com.my.package.MyClass">${myObj.myGetter}</div>
```

Sling Models

Sling Models is the preferred method for implementing the business logic of a component. Sling Models are Java classes used to develop components. Sling Model supports both HTML components and a headless Content Management System (CMS).

Sling Models are Java classes that can be mapped to Sling resources. To support a headless CMS, you can also use the Sling Model Exporter to export the resource (the node representing the component) as JavaScript Object Notation (JSON). Apache Sling provides a Jackson JSON exporter to cover the most common case of exporting Sling Models as JSON objects. The JSON objects are used by programmatic web consumers such as other Web services and JavaScript applications.

Server-Side JavaScript

Server-side JavaScript is made available through the Use-API. This API is from the HTL specification and helps developers write quick business logic for their components. Generally, business logic is built in Java. When the component needs some view-specific changes from what the Java API provides, it can be convenient to use some server-side executed JavaScript to do that.

```
use(function () {
    var curTitle = currentPage.getTitle();
    return {
        link: "#my link's safe",
        title: curTitle,
        text: "my text's safe"
    };
});
```

WCMUsePojo Java Class

WCMUsePojo is another method to expose the business logic by using the Use-API. Writing the business logic within a Java class helps implement the heavier processes easily and efficiently. With the introduction of Sling Models for components, Java code must be written with Sling Models rather than WCMUsePojo. WCMUsePojo will continue to be supported for backwards compatibility.

Exercise 1: Create a basic header component

In this exercise, you will create a basic header component and add it to the template.

You need to install the **adls-training-project-v4.0.zip** package on your AEM author instance. The package contains the training project structure, page component with clientlibs, template, and site structure that was already created for you.

To create the basic header component:

1. Click **Adobe Experience Manager** from the header bar. You will be in the **Tools** console.
2. Click **CRXDE Lite**. The CRXDE Lite page opens.
3. Navigate to the **/apps/training/components/structure** folder.
4. Select the **structure** folder, click **Create** from the actions bar, and then click **Create Component** from the drop-down menu. The **Create Component** dialog box opens.
5. Enter the following in the dialog box:
 - a. **Label: header**
 - b. **Title: We.Train Header**
 - c. **Group: We.Train.Structure**
6. Click **Next**, and then click **OK**. The component is created.
7. Click **Save All** from the actions bar.



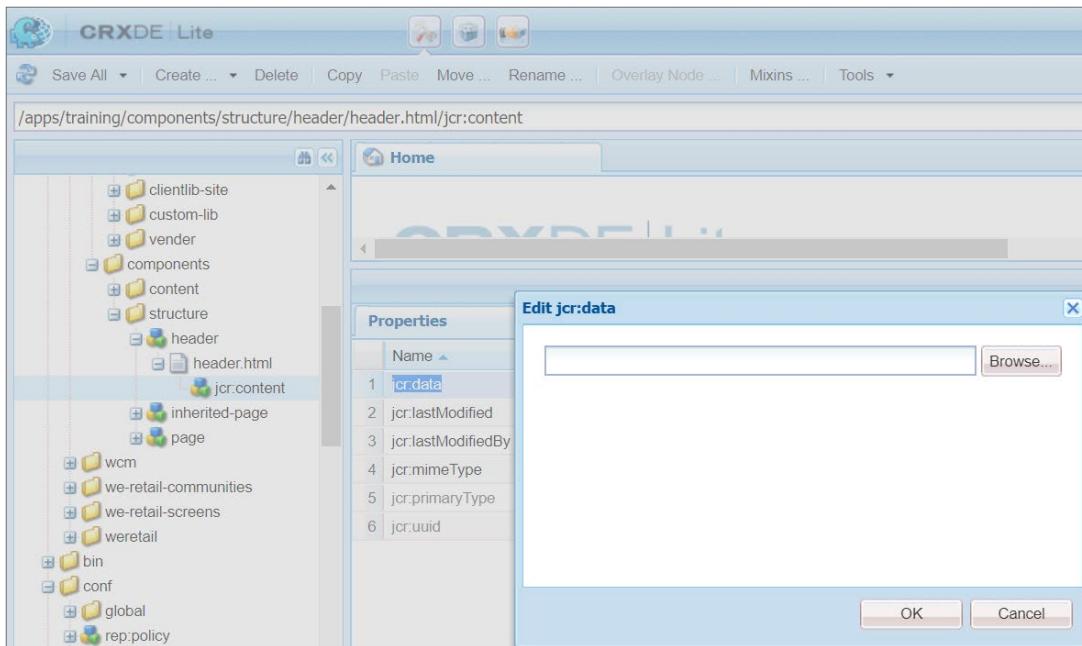
Note: You can also press Ctrl+S (Windows users) and Command+S (Mac users) to save your work in CRXDE Lite.

8. Navigate to the header component, select the **header.jsp**, click **Rename** from the actions bar, and change the file name to **header.html**.

9. Click **Save All** from the actions bar.
10. Double-click the **header.html** to open the script for editing.

To add code to the header.html page:

11. Select the **jcr:content** node that is below **header.html**. The **Properties** tab opens.
12. Double-click the **jcr:data** property. The **Edit jcr:data** dialog box opens.
13. Click **Browse** as shown. The **Open** dialog box opens.



14. Navigate to the **Exercise_Files** folder on your file system.
15. Select **basic_header.html**, and then click **Open**.
16. Click **OK** in the **Edit jcr:data** dialog box. The default sample code is replaced with the code in **header.html**.
17. Open the **header.html** to view the code that you added. Notice how the following code snippet of **header.html** has a simple navigation that lists the child pages of the current page.

```

01. <h3>Header Component</h3>
02.
03. <ul class="nav navbar-nav navbar-center">
04.   <li class="nav navbar-nav navbar-left" data-sly-repeat="${currentPage.listchildren}">
05.     <a href="${item.path}.html">${item.title}</a>
06.   </li>
07. </ul>

```

18. Click **Save All**.

Before you add the header component to the template, you need to add a cq:dialog node to the header component.

19. Select the **header** component, click **Create** from the actions bar, and then click **Create Node**.

The **Create Node** dialog box opens.

20. Enter the following:

- Name:** cq:dialog
- Type:** nt:unstructured

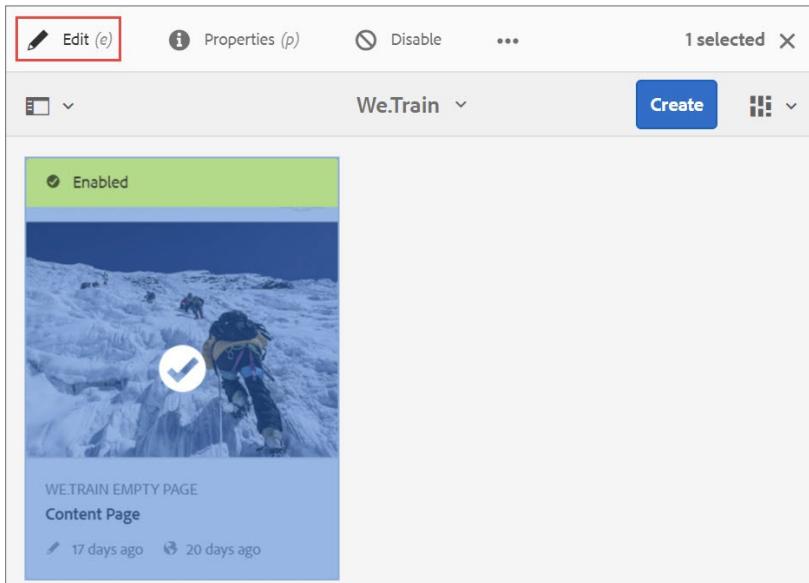
21. Click **OK**, and then click **Save All** from the actions bar.

22. Click **CRXDE Lite** from the header bar to navigate back to the **Tools** console in AEM.

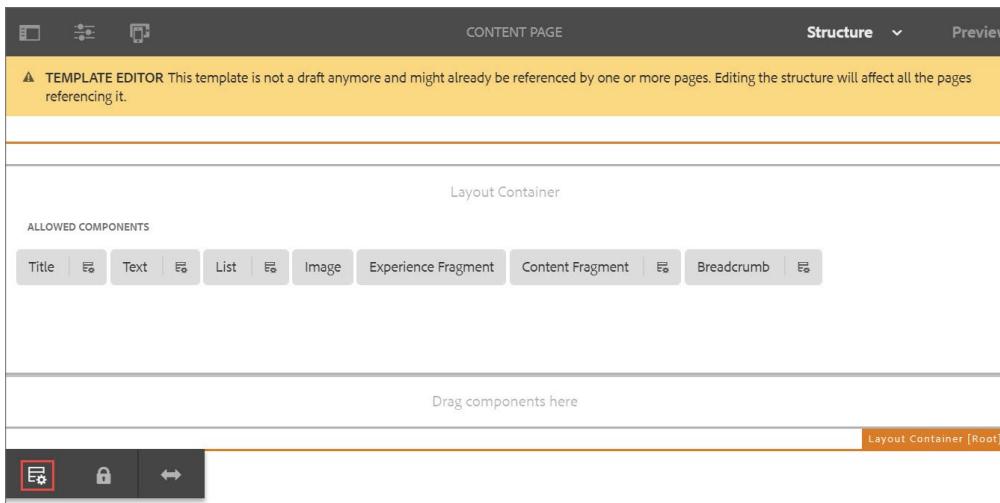
To add the header component to the template:

23. Ensure you are in the **Tools** console, and click **Templates**.

24. Navigate to the **We.Train** folder, select the Content Page template, and click **Edit** from the actions bar as shown. The template editor opens.



25. Select the **Layout Container [Root]**, and click the Policy icon from the component toolbar as shown. The **Layout Container** policy wizard opens.



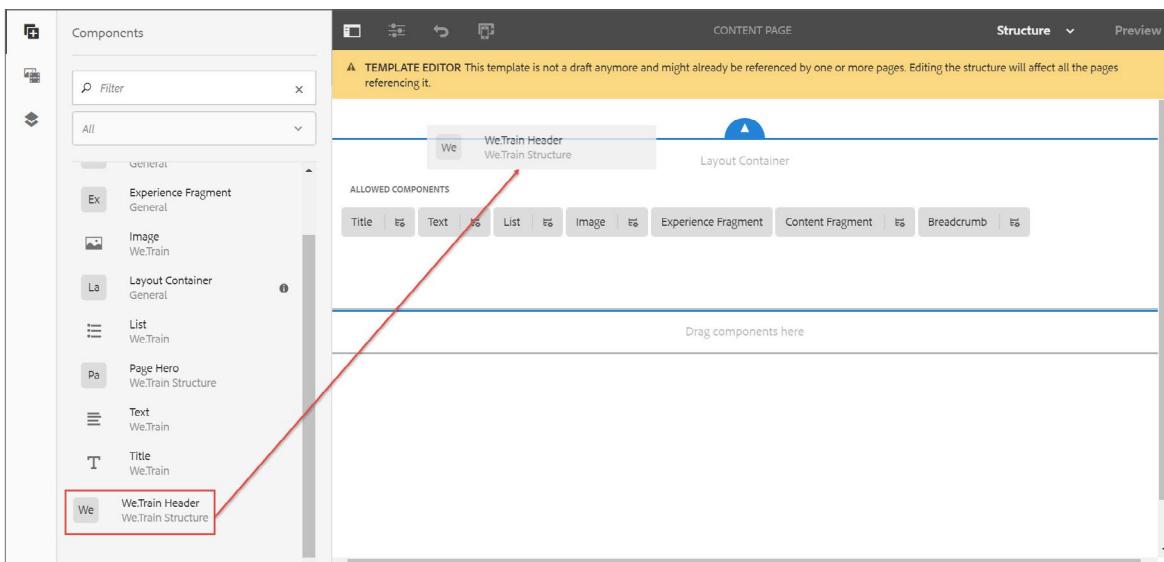
26. Under the **Properties** section, on the **Allowed Components** tab, scroll down the components list and select the checkbox beside the **We.Train.Structure** group that contains the header component.

27. Click the Done (checkmark) icon in the the upper-right corner.

 **Note:** When adding the header component to the Content Page template, you did not create a new policy; instead , you used the existing We.Train Structure Policy. The next template that uses the We.Train. Structure Policy will automatically have access to the structure components (header) that you have built.

28. Click the Toggle Side Panel icon, if not already selected, and then click the Components icon from the panel. The Components browser opens.

29. Drag the **We.Train Header** from the panel, drop it above the **Layout Container** as shown. The **Header Component** is added to the template.



30. Open a new tab in your browser, enter <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The **English** page of the We.Train site with the header component (with its child pages) opens as shown:



You have created the component, added it to the template, and verified it on the page. Now, you can begin a deeper development with the header.html script.

31. Ensure you are in CRXDE Lite or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
32. Navigate to the **/apps/training/components/structure/header** node.
33. Double-click the **header.html** to open the script for editing.
34. Update the **header.html** script by replacing the existing code with new code (**basic_header_2.html**) from the **Exercise_Files** folder.
35. Click **Save All**.
36. Open the **header.html** to view the code that you added. Notice in the following code snippet of **header.html**, there is an account navigation bar and a responsive site navigation bar. Most anchors are `href="#"` that help mock the page for the business logic. Notice, `${header.rootPage.path}` and `${!header.anonymous}`, these are the variables that you get from the business logic. Because the header variable is not defined, it is not evaluated.

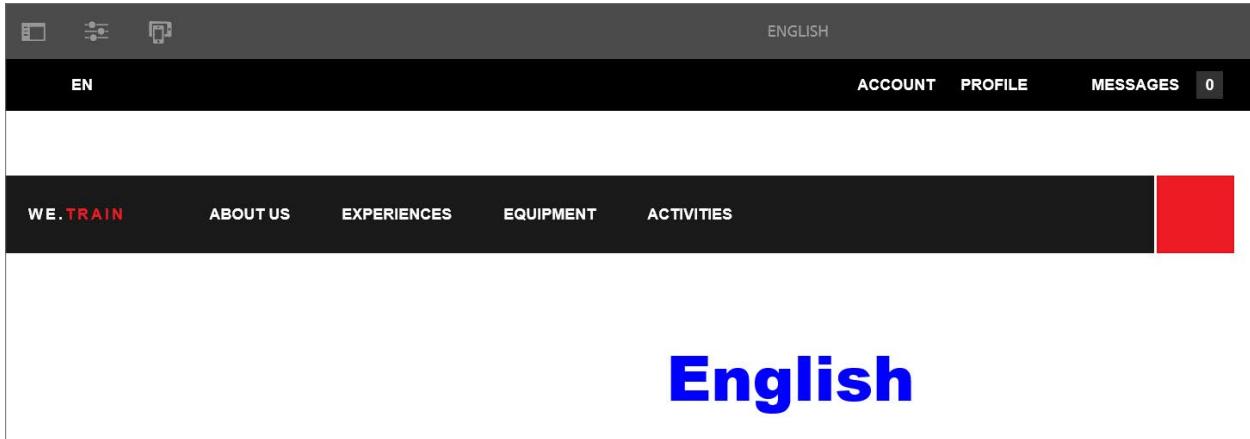
```

01. <!--/* -----Account Navigation Bar----- */-->
02. <div
03.     class="navbar navbar-inverse navbar-fixed-top hidden-xs hidden-sm">
04.         <div class="container-fluid">
05.             <ul class="nav navbar-nav navbar-left">
06.                 <li><a href="${header.rootPage.path || '#' @ extension = 'html'}" style="text-transform: uppercase;">
07.                     <i class="fa fa-globe" aria-hidden="true"></i>
08.                     ${header.rootPage.name || 'en'}
09.                 </a></li>
10.             </ul>
11.             <ul class="nav navbar-nav navbar-right">
12.                 <li data-sly-test="${!header.anonymous}">
13.                     <a href="${'#' @ extension = 'html'}">${ 'Account' @ i18n }</a></li>
14.                     <li data-sly-test="${!header.anonymous}">
15.                         <a href="${'#' @ extension = 'html'}">${ 'Profile' @ i18n }</a></li>
16.                         <li data-sly-test="${!header.anonymous}"><a href="${'#' @ extension = 'html'}">
17.                             <i class="fa fa-envelope-o"></i> ${'Messages' @ i18n }
18.                             <span class="badge" id="we-retail-message-count" data-siteurl="#">0</span></a>
19.                         </li>
20.                         <li data-sly-test="${!header.anonymous}"><a href="${'#' @ extension = 'html'}">
21.                             <i class="fa fa-flag-o"></i> ${'Notifications' @ i18n }
22.                             <span class="badge" id="we-retail-notification-count" data-siteurl="#">0</span>
23.                         </a></li>
24.                     </ul>
25.                 </div>
26.             </div>

```

37. Navigate to the tab where the English page of the We.Train site is open and refresh the page.

Notice how the header is completely responsive as shown:



Exercise 2: Add JavaScript business logic to the header component

In this exercise, you will use JavaScript to implement the business logic of the header component.

1. Ensure you are in CRXDE Lite or enter <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components/structure/header** node.
3. Select the **header** node, click **Create** from the actions bar, and then click **Create File** from the drop-down menu. The **Create File** dialog box opens.
4. Enter **header.js** in the **Name** field, and then click **OK**. The file is created.

To add code to the header.js file:

5. Double-click the **header.js** file to open the script for editing.
6. Update the **header.js** script by replacing the existing code with new code (**header.js**) from the **Exercise_Files** folder.

7. Click **Save All**.
8. Open **header.js** to view the code that you added. In the following code snippet:
 - a. Line 4, helps get the language root of the website to display the child pages in navigation.
 - b. Line 12, helps log a message in the server-side Javascript.
 - c. Line 14, helps filter the child pages, only if **Hide in navigation** page property is selected.

```

01. "use strict";
02. use(function() {
03.   var items = [];
04.   var root = currentPage.getAbsoluteParent(2);
05.
06.   //make sure that we always have a valid set of returned items
07.   //if navigation root is null, use the currentPage as the navigation root
08.   if(root == null){
09.     root = currentPage;
10.   }
11.
12.   log.info("#####NavRoot Page: " + root.title);
13.
14.   var it = root.listChildren(new Packages.com.day.cq.wcm.api.PageFilter());
15.   while (it.hasNext()) {
16.     var page = it.next();
17.     items.push(page);
18.   }
19.
20. /* Change anonymousUser=false to see the Account Navigation change with account links */
21. var anonymousUser = true;
22.
23. return {
24.   items: items,
25.   rootPage: root,
26.   anonymous: anonymousUser
27. }
28. });

```

To update the code in the header.html page:

9. Double-click the **header.html** to open the script for editing.
10. Update the **header.html** script by replacing the existing code with new code (**js_header.html**) from the **Exercise_Files** folder.
11. Click **Save All**.

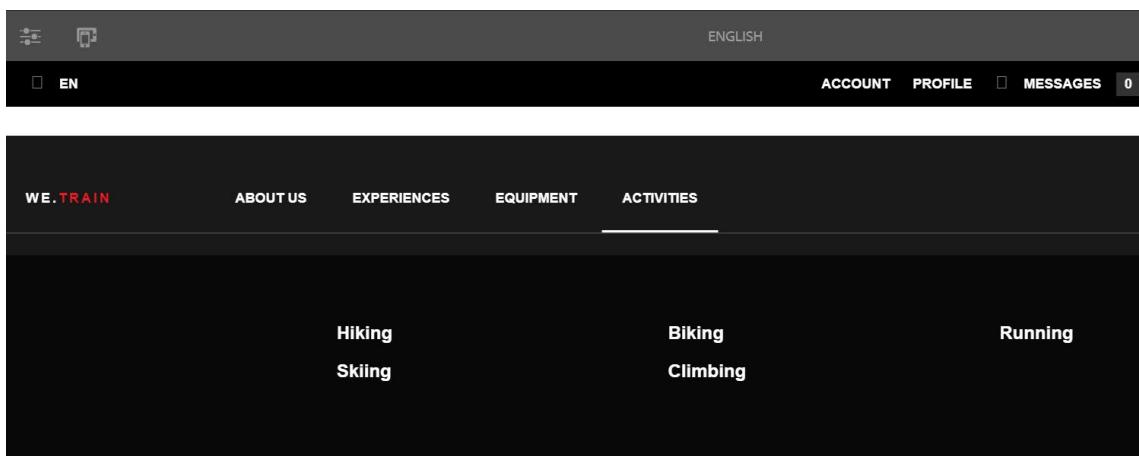
12. Open the **header.html** to view the code that you added. Notice the following lines in the below code snippet:

- a. Line 2, `data-sly-use.header="header.js"`. helps define the header variable with a HTL Javascript Use-script.
 - b. Notice how `${header.rootPage.path}` and `${!header.anonymous}` are now able to render based on the JavaScript.
 - c. Line 53, `data-sly-repeat="${header.items}"`, instead of using the child pages of the current page, the HTL statement requests the items object from the header.js script.

```
01. <!--/* -----HTML with server-side business logic added----- */-->
02. <sly data-sly-use.header="header.js" />
03. <div class="navbar navbar-inverse navbar-fixed-top hidden-xs hidden-sm">
04.     <div class="container-fluid">
05.         <ul class="nav navbar-nav navbar-left">
06.             <li><a href="${header.rootPage.path || '#' @ extension = 'html'}" style="text-transform: uppercase;">
07.                 <i class="fa fa-globe" aria-hidden="true"></i>
08.                 ${header.rootPage.name || 'en'}
09.             </a></li>
10.         </ul>
11.
12.         <!--/* -----Navigation Business Logic----- */-->
13.
14.         <li data-sly-repeat="${header.rootPage.listChildren}">
15.             <a href="${ item.path @ extension = 'html'}">${item.title}</a>
16.             <!--/* -----Subnavigation Logic----- */-->
17.             <ul data-sly-list.subitem="${item.listChildren}" class="navbar-nav-subitems">
18.                 <li>
19.                     <a href="${ subitem.path @ extension = 'html'}">${subitem.title}</a>
20.                 </li>
21.             </ul>
22.
23.         </li>
```

To view the changes on the English page:

13. Navigate to the tab where the **English** page of the We.Train site is open and refresh the page.
 14. If the **English** page is not open, open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The English page opens.
 15. Click **Preview** from the page toolbar, and then hover over each child page to view its subpages as shown. Notice how the header component has the same functionality as seen in the previous exerciseas shown:



Exercise 3: Add a Sling Model business logic to the header component

In this exercise, you will upload a Sling Model **we.train.core-0.0.1-SNAPSHOT.jar** file (that contains Java classes created by Maven and Eclipse, and built into the OSGi Bundle) into your AEM author instance that implements the business logic of the header component.

Before installing the bundle, examine the **header.java** class that implements the Sling Model business logic.

1. Navigate to your file system, open the **Exercise_Files** folder, and then open **header.java** file in text editor (Notepad++). Notice in the following code snippet, the Java class has getters [`get()` method] that are similar to the Javascript business logic, which you added in the previous exercise.

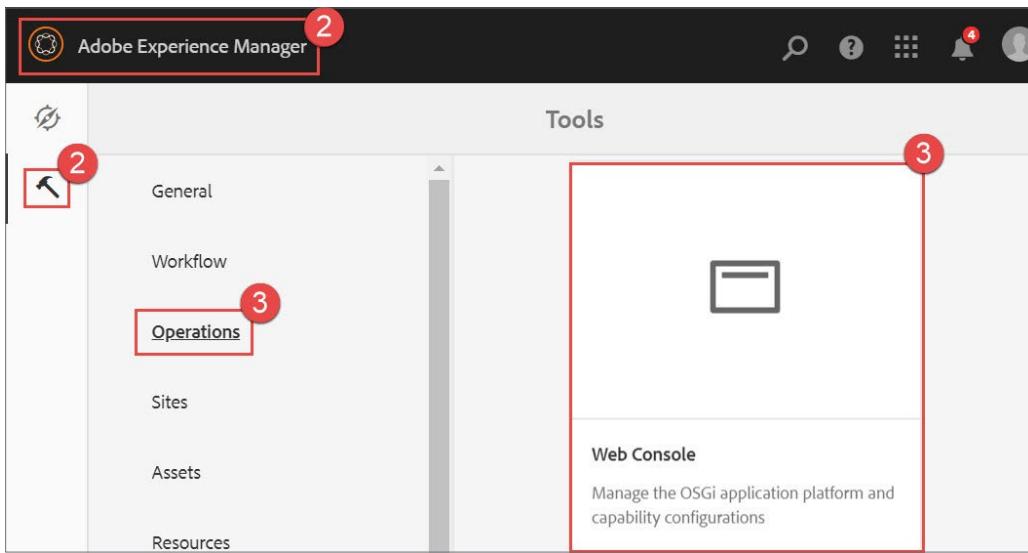
```

01. @Model(adaptables=SlingHttpServletRequest.class,
02.         resourceType = "training/components/structure/header",
03.         defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL)
04. public class Header {
05.
06.     @ScriptVariable
07.     private Page currentPage;
08.
09.     @ScriptVariable
10.     private SlingHttpServletRequest request;
11.
12.     @PostConstruct
13.     private Page root() {
14.         if (currentPage.getAbsoluteParent(2) != null) {
15.             return currentPage.getAbsoluteParent(2);
16.         }
17.         return currentPage;
18.     }
19.
20.
21.     private List<Page> rootChildren() {
22.         List<Page> items = new ArrayList<Page>();
23.         Iterator<Page> it = root().listchildren(new PageFilter());
24.         while (it.hasNext()) {
25.             items.add(it.next());
26.         }
27.         return items;
28.     }
29.
30.     //getters
31.
32.     public String getAnonymous() {
33.         return (request.getRequestParameter("anonymous") != null) ?
34.                 request.getRequestParameter("anonymous").toString() : "false";
35.
36.     }

```

To install the bundle:

2. Click **Adobe Experience Manager** from the header bar, and then click the Tools icon.
3. Click **Operations > Web Console** as shown. The **Adobe Experience Manager Web Console Configuration** page opens.



4. Click **OSGi** from the header bar, and then select **Bundles** from the drop-down menu as shown. The **Adobe Experience Manager Web Console Bundles** page opens.

5. Click **Install/Update...** from the actions bar as shown. The **Upload / Install Bundles** dialog box opens.

The screenshot shows the 'Bundles' section of the AEM Web Console. At the top, there's a navigation bar with links for Main, OSGi, Sling, Status, and Web Console, along with a Log out button. Below the navigation is a message stating 'Bundle information: 544 bundles in total - all 544 bundles active'. The main area is a table with columns for ID, Name, Version, and Category. Two rows are visible: 'System Bundle (org.apache.felix.framework)' with Version 5.6.10 and 'Abdera Client (org.apache.abdera.client)' with Version 1.0.0.R783018. At the bottom of the table is a 'Install/Update...' button, which is highlighted with a red box.

6. Select the **Start Bundle** and **Refresh Packages** checkbox, enter **1** in the **Start Level** field, and then click **Browse** as shown. The **Open** dialog box appears.

This is a screenshot of the 'Upload / Install Bundles' dialog. It has three checkboxes at the top: 'Start Bundle' (checked), 'Refresh Packages' (checked), and 'Start Level' with the value '1'. Below these is a 'Browse...' button. At the bottom is an 'Install or Update' button.

7. Navigate to the **Exercise_Files** folder on your file system, select the **we.train.core-0.0.1-SNAPSHOT.jar** file, and then click **Open** as shown. You will be taken back to the **Bundles** page.

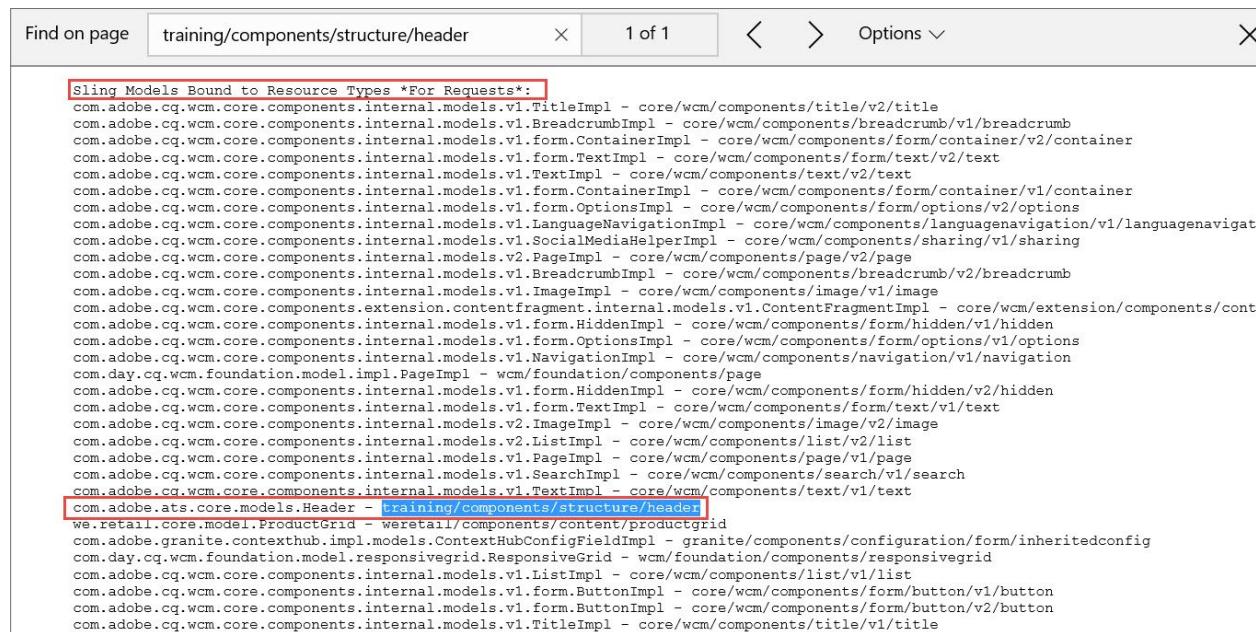
This is a screenshot of a Windows 'Open' file dialog. The path 'M6 > M6_Exercise_Files' is selected. In the list, the file 'we.train.core-0.0.1-SNAPSHOT.jar' is highlighted. At the bottom, the 'File name:' dropdown shows 'we.train.core-0.0.1-SNAPSHOT.jar', and the 'Open' button is highlighted with a blue border.

- Click **Install or Update** in the **Upload / Install Bundles** dialog box. The **Adobe Experience Manager Web Console Bundles** page refreshes when the install is complete.

To verify if the bundle is installed successfully on your instance:

- Click **Status** from the header bar, and then select **Sling Models** from the drop-down menu. The **Adobe Experience Manager Web Console Sling Models** page opens.
- Press **Ctrl + F** to open the **Find on page** dialog box.

Enter **training/components/structure/header** in the field and press Enter. Notice that the **com.adobe.ats.core.models.Header - training/components/structure/header** is under **Sling Models Bound to Resource Types *For Requests** (if you cannot find this, use **Ctrl+F**). This confirms the bundle is installed successfully as shown:



- Click **X (Close)** icon to close the **Find on page** dialog box.

To update the code on the header.html page:

- Ensure the CRXDE Lite tab is open or enter <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
- Navigate to the **/apps/training/components/structure/header** node.

14. Double-click the **header.html** to open the script for editing.
15. Update the **header.html** script by replacing the existing code with new code (**model_header.html**) from the **Exercise_Files** folder.
16. Open the **header.html** to view the code that you added. Notice that the **model_header.html** script is the exact same **js_header.html**, except you are requesting the Sling Model rather than the Javascript.

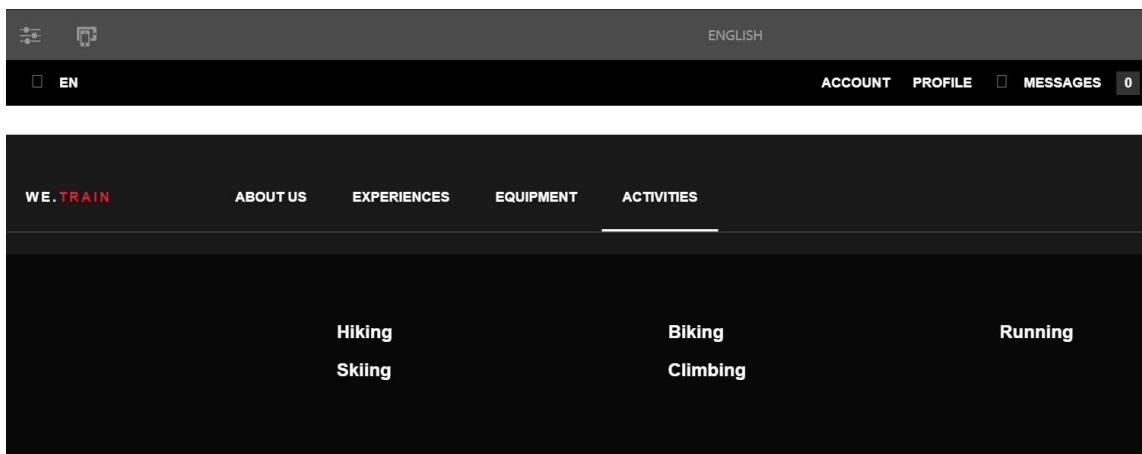
```

01. <!--/* -----HTL with Sling Model business logic added----- */-->
02. <sly data-sly-use.header="com.adobe.ats.core.models.Header" />
03. <div class="navbar navbar-inverse navbar-fixed-top hidden-xs hidden-sm">
04.   <div class="container-fluid">
05.     <ul class="nav navbar-nav navbar-left">
06.       <li><a href="${header.rootPage.path || '#' @ extension = 'html'}" style="text-transform: uppercase;">
07.         <i class="fa fa-globe" aria-hidden="true"></i>
08.         ${header.rootPage.name || 'en'}
09.       </a></li>
10.     </ul>
11.     <!--/* -----Navigation Business Logic----- */-->
12.
13.     <li data-sly-repeat="${header.rootPage.listChildren}">
14.       <a href="${ item.path @ extension = 'html'}">${item.title}</a>
15.       <!--/* -----Subnavigation Logic----- */-->
16.       <ul data-sly-list.subitem="${item.listChildren}" class="navbar-nav-subitems">
17.         <li >
18.           <a href="${ subitem.path @ extension = 'html'}">${subitem.title}</a>
19.         </li>
20.       </ul>
21.
22.     </li>
23.   </ul>
24.
25. </div>
26.
27. <!--/* -----Footer Business Logic----- */-->
28.
29. <div class="container">
30.   <div class="row">
31.     <div class="col-md-4">
32.       <h2>Activities</h2>
33.       <ul class="list-group">
34.         <li>Hiking</li>
35.         <li>Skiing</li>
36.         <li>Climbing</li>
37.         <li>Biking</li>
38.       </ul>
39.     </div>
40.     <div class="col-md-4">
41.       <h2>Equipment</h2>
42.       <ul class="list-group">
43.         <li>Running</li>
44.         <li>Swimming</li>
45.         <li>Cycling</li>
46.         <li>Hiking</li>
47.       </ul>
48.     </div>
49.     <div class="col-md-4">
50.       <h2>Experiences</h2>
51.       <ul class="list-group">
52.         <li>Mountain Climbing</li>
53.         <li>Rock Climbing</li>
54.         <li>Snowboarding</li>
55.         <li>Hiking</li>
56.       </ul>
57.     </div>
58.   </div>
59. </div>
60.
61.
62.

```

To view the changes on the English page:

17. Navigate to the tab where the **English** page of the We.Train site is open and refresh the page.
18. If the **English** page is not open, open a new tab in your browser, enter <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The English page opens.
19. Click **Preview** from the page toolbar, and then hover over each child page to view its subpages, as shown. Notice that the header component has the same functionality as seen in the previous exercise as shown:



Dialogs

Dialogs are a key element of a component as they provide an interface for authors to configure and provide input to the component.

Depending on the complexity of the component, the dialogs can have one or more tabs. The tabs help the dialog to be short and sort the input fields.

Coral UI and Granite UI

The Coral UI and Granite UI define the look and feel of AEM. The Granite UI provides a large range of the basic components (widgets) needed to create your dialog in the authoring environment. When necessary, you can extend this selection and create your own widget.

cq:dialog

The `cq:dialog` (nt:unstructured) node type is used to create a dialog.

The `cq:dialog` node:

- Defines the dialog for editing the content of this component
- Is specific to the touch-enabled UI
- Is defined by using Granite UI components
- Has a property `sling:resourceType`, as the standard Sling content structure
- Can have a property `helpPath` to define the context-sensitive help resource (absolute or relative path) that is accessed when the Help icon (the ? icon) is selected.
 - › For out-of-the box components, `helpPath` often references a page in the documentation.
 - › If no `helpPath` is specified, the default URL (documentation overview page) is displayed.

Configuring the Edit Behavior

You can configure the edit behavior of a component by adding a cq:editConfig node of type cq:EditConfig below the component node (of type cq:Component) and by adding the specific properties and child nodes. The following properties and child nodes are available:

- The cq:editConfig node properties are:
 - › cq:actions (String array): Defines the actions that can be performed on the component.
 - › cq:layout (String): Defines how the component is edited in the classic UI.
 - › cq:dialogMode (String): Defines how the component dialog is opened in the classic UI
 - › cq:emptyText (String): Defines text that is displayed when no visual content is present.
 - › cq:inherit (Boolean): Defines if missing values are inherited from the component that it inherits from.
 - › dialogLayout (String): Defines how the dialog should open.
- The cq:editConfig child nodes are:
 - › cq:dropTargets (node type nt:unstructured): Defines a list of drop targets that can accept a drop from an asset of the content finder
 - › cq:actionConfigs (node type nt:unstructured): Defines a list of new actions that are appended to the cq:actions list.
 - › cq:formParameters (node type nt:unstructured): Defines additional parameters that are added to the dialog form.
 - › cq:inplaceEditing (node type cq:InplaceEditingConfig): Defines an in-place editing configuration for the component.
 - › cq:listeners (node type cq>EditListenersConfig): Defines what happens before or after an action occurs on the component.

Exercise 4: Create an edit dialog for the component

To create an edit dialog:

1. Ensure the CRXDE Lite tab is open or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components/structure** folder.
3. Select the **structure** folder, click **Create** from the actions bar, and then click **Create Component** from the drop-down menu. The **Create Component** dialog box opens.
4. Enter the following values in the dialog box:
 - a. **Label:** `pagehero`
 - b. **Title:** `Page Hero`
 - c. **Group:** `We.Train.Structure`
5. Click **Next**, and then click **OK**. The component is created.
6. Click **Save All** from the actions bar.
7. Navigate to the `pagehero` node in CRXDE Lite, select the `pagehero.jsp`, click **Rename** from the actions bar, and change the file name to `pagehero.html`.
8. Click **Save All** from the actions bar.
9. Double-click the `pagehero.html` to open the script for editing.

To add code to the pagehero.html page:

10. Double-click the **pageheader.html** to open the script for editing.
11. Update the **pageheader.html** script by replacing the existing code with new code (**basic_pagehero.html**) from the **Exercise_Files** folder.
12. Open the **pagehero.html** to view the code that you added. Notice that the following code snippet, the HTL script uses the global variables properties and resource, which helps return values persisted by the dialog box. Since you do not have a dialog box yet, the values will not return anything.

```

01. <sly data-sly-use.hero="pagehero.js">
02.   <div class="we-HeroImage width-full jumbotron"
03.     style="${hero.style @ context='styleString'}">
04.       <div class="container cq-dd-image">
05.         <div class="we-HeroImage-wrapper">
06.           <p class="h3">${properties.heading}</p>
07.           <strong class="we-HeroImage-title h1">${properties.jcr:title}</strong>
08.
09.
10.          <p data-sly-test="${properties.buttonLabel && properties.buttonLinkTo}">
11.            <a class="btn btn-
12. primary" href="${properties.buttonLinkTo @ extension = 'html'}" role="button">
13.              ${properties.buttonLabel}
14.              <i class="fa fa-arrow-right" aria-hidden="true"></i>
15.            </a>
16.          </p>
17.        </div>
18.      </div>
19.    </sly>
```

13. Click **Save All**.
14. Select the **pagehero** node, click **Create** from the actions bar, and then click **Create File** from the drop-down menu. The **Create File** dialog box opens.
15. Enter **pagehero.js** in the **Name** field, and then click **OK**. The file is created.

To add code to the pagehero.js file:

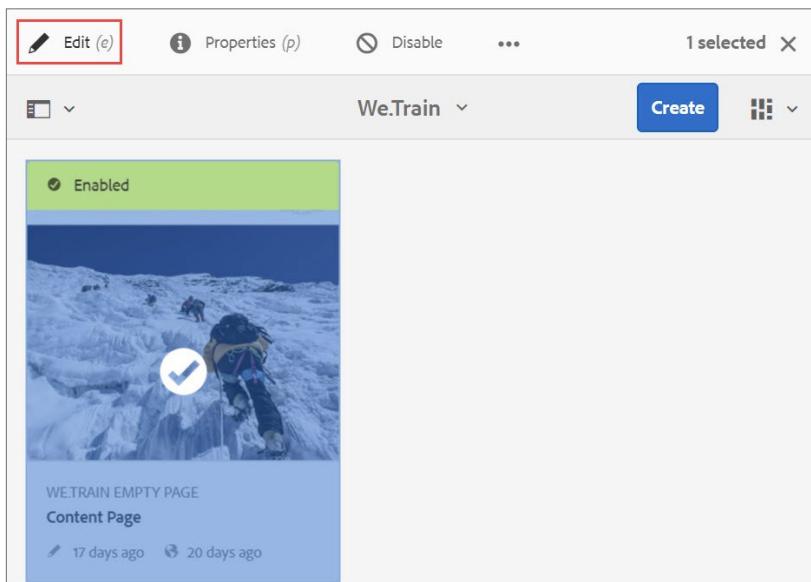
16. Double-click the **pagehero.js** to open the script for editing.
17. Update the **pagehero.js** script by replacing the existing code with new code (**hero.js**) from the **Exercise_Files** folder.

Before, you add the pagehero component to the template, you need to add a cq:dialog node to the pagehero component.

18. Select the **pagehero** node, click **Create** from the actions bar, and then click **Create Node**. The **Create Node** dialog box opens.
19. Enter and select the following values:
 - a. **Name:** cq:dialog
 - b. **Type:** nt:unstructured
20. Click **OK**, and then click **Save All** from the actions bar.
21. Click **CRXDE Lite** from the header bar to navigate back to the **Tools** console.

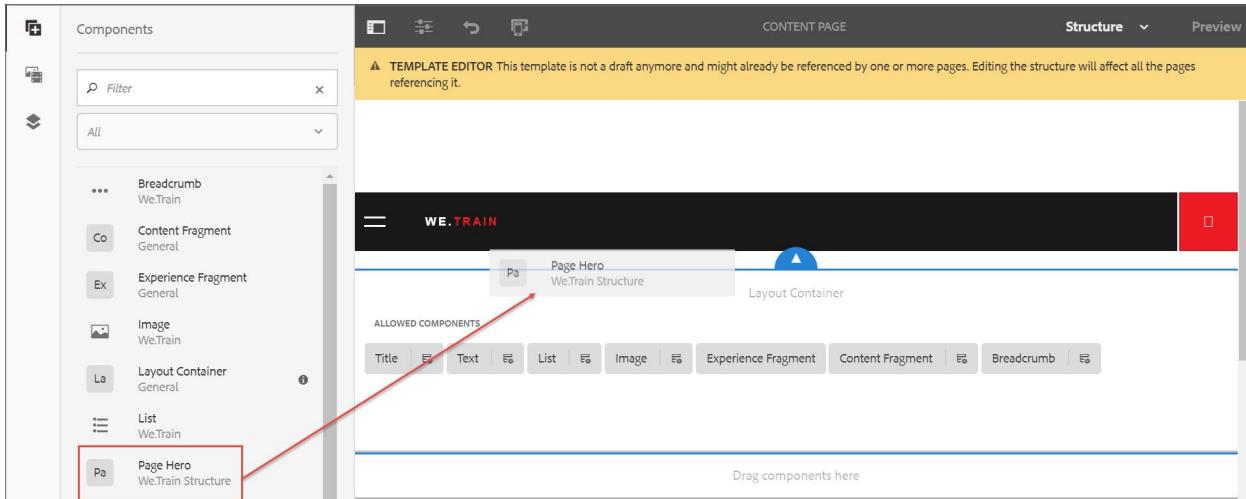
To add the pagehero component to the template:

22. Ensure you are in the **Tools** console, and click **Templates**.
23. Navigate to the **We.Train** folder, select the **Content Page** template, and click **Edit** from the actions bar as shown. The template editor opens.

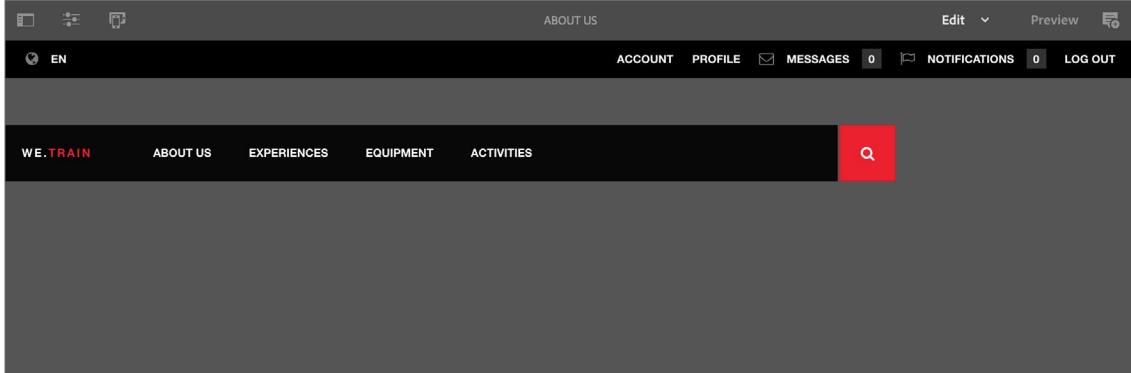


24. Click Toggle Side Panel, and then click Components icon from the panel. Notice, The Page Hero component is available in the Components browser.

25. Drag the **Page Hero Component** from the panel, drop it above the **Layout Container** below the **Header** component as shown. The **Page Hero** component is added to the template.



26. Open a new tab in your browser, enter <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The English page of the We.Train site with the page hero component opens as shown:

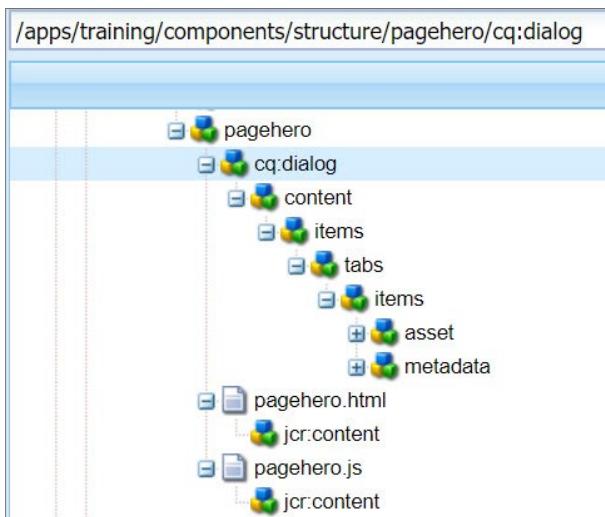


Now that you have added the Page Hero component to the template, you will start to build and test the dialog box.

27. Ensure you are in CRXDE Lite or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.

In AEM, it is a best practice to reuse a node structure rather than recreating it. You will copy the Core Image dialog box and then modify it according to your need.

28. Navigate to the **/apps/training/components/structure/pagehero** node.
29. Delete the **cq:dialog** node below the pagehero node.
30. Click **Save All**.
31. Navigate to the **/apps/core/wcm/components/image/v2/image** node, select the **cq:dialog** node, and then click **Copy** from the actions bar.
32. Navigate to the **/apps/training/components/structure** folder, select the **pagehero** component, and then click **Paste**. The cq:dialog node of Core Image is added to the pagehero component as shown:



33. Click **Save All**.

34. Select the **cq:dialog** node, and on the **Properties** tab, double-click **jcr:title**, and update the value as **Page Hero** as shown:

Name	Type	Value
extraClientlibs	String[]	core.wcm.components.image.v2....
helpPath	String	https://www.adobe.com/go/aem_c...
jcr:primaryType	Name	nt:unstructured
jcr:title	String	Page Hero
sling:resourceType	String	cq/gui/components/authoring/dialog

35. Right-click the **extraClientlibs** property, and select **Delete** from the drop-down menu as shown:

36. Similarly, delete the **helpPath** property, and click **Save All**.

37. Navigate to the **/apps/training/components/structure/pagehero/cq:dialog/content/items/tabs/items/metadata/items** node, select the **columns** node, and then click **Delete** from the actions bar to create own formfields.

38. Click **Save All**.

39. Navigate to the **/apps/training/components/structure/pagehero/cq:dialog/content/items/tabs/items**, select the **metadata** node, click **Rename** from the actions bar, and rename it to **hero**.

40. Select the above **hero** node, in the **Properties** tab, double-click **jcr:title**, and update the value as **Hero**.

41. Select the **items** node that is below the **hero** node, click **Create** from the actions bar, and then click **Create Node**. The **Create Node** dialog box opens.

42. Enter the following:

- a. Name: title
- b. Type: nt:unstructured

43. Click **OK**, and then click **Save All** from the actions bar.

44. Select the **title** node, and add the following properties as shown:

Name	Type	Value
fieldLabel	String	Title
name	String	./jcr:title
sling:resourceType	String	granite/ui/components/foundation/form/textfield

Name	Type	Value
1 fieldLabel	String	Title
2 jcr:primaryType	Name	nt:unstructured
3 name	String	./jcr:title
4 sling:resourceType	String	granite/ui/components/found...

45. Click **Save All**.

Similar to the **title** node, you will create **heading**, **buttonLabel**, and **buttonLink** form fields below the **/hero/items** node. Instead of creating the formfields, you will copy/paste the **title** node and update the required values accordingly.

46. Select the **title** node, and then click **Copy** from the actions bar.

47. Ensure you have selected the **/hero/items** node, and then click **Paste**. The **Copy of title** node is created.

48. Select the **Copy of title** node, and rename the node to **heading**.

49. Select the **heading** node and update the values of the following properties:

Name	Type	Value
fieldLabel	String	Heading
name	String	./heading
sling:resourceType	String	granite/ui/components/foundation/form/textfield

50. Click **Save All**.

51. Perform steps 54 and 55, and create **buttonLabel** and **buttonLink** nodes below the **/hero/items** node.

52. Select the **buttonLabel** node and update the values of the following properties:

Name	Type	Value
fieldLabel	String	Button Label
name	String	./buttonLabel
sling:resourceType	String	granite/ui/components/foundation/form/textfield

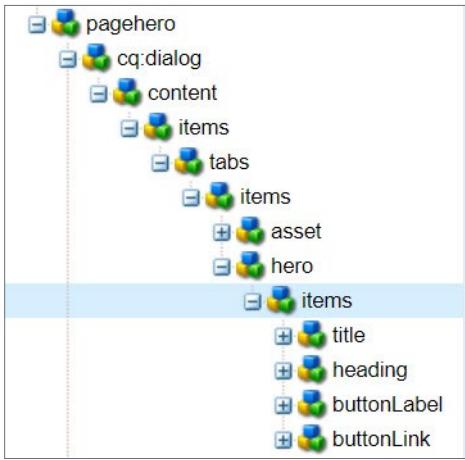
53. Click **Save All**.

54. Select the **buttonLink** node and update the values of the following properties:

Name	Type	Value
fieldLabel	String	Button Link
name	String	./buttonLinkTo
rootPath	String	/content/we-train
sling:resourceType	String	granite/ui/components/foundation/form/pathfield

55. Click **Save All**.

Your form fields structure should look similar to the one given in the below screenshot:

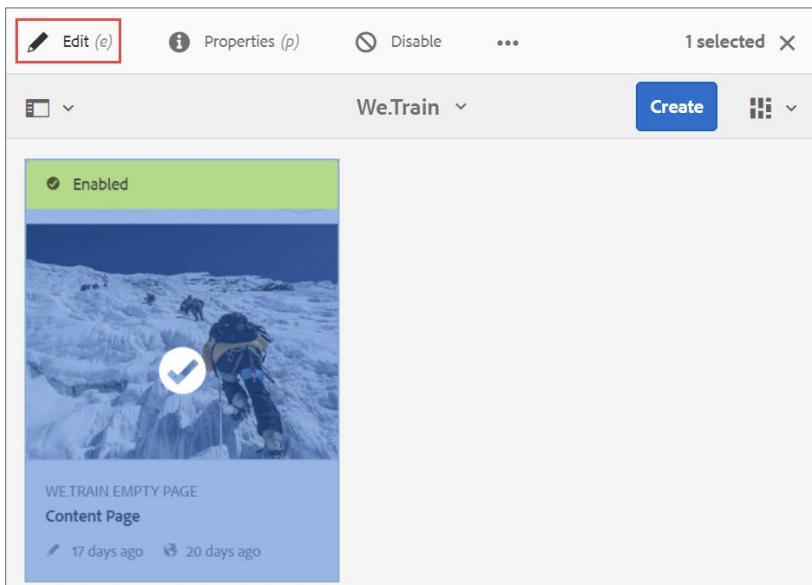


To enable the dialog box on the page, first you need to unlock the Page Hero component in the template.

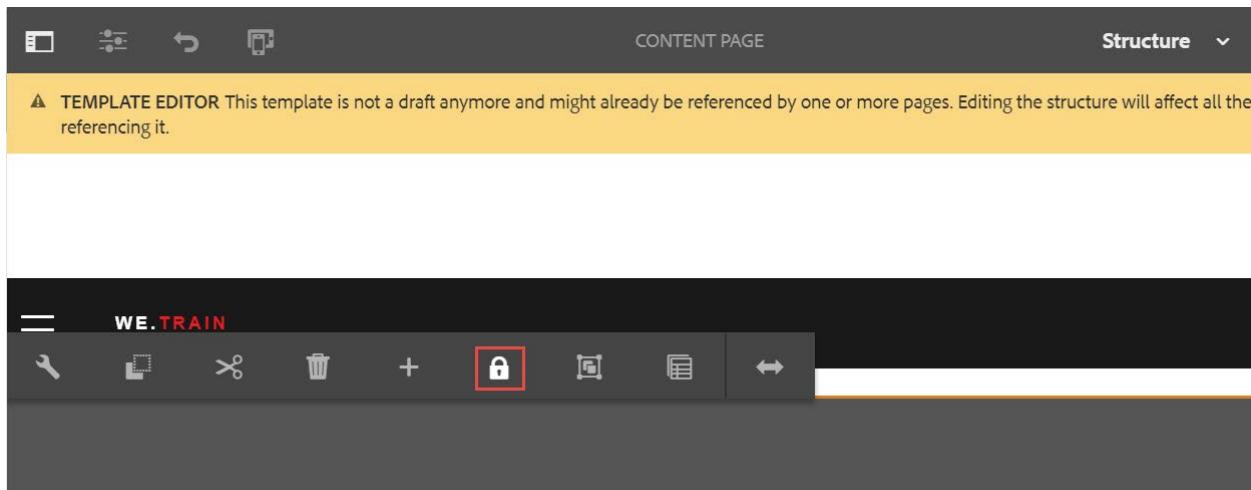
56. Ensure the **Content Page** tab is open in your browser or click **CRXDE Lite** from the header bar to navigate back to the Navigation pane.

57. Click the Tools icon, and then click **Templates**. The **Templates** console opens.

58. Navigate to the **We.Train** folder, select the **Content Page** template, and click **Edit** from the actions bar as shown. The template editor opens.



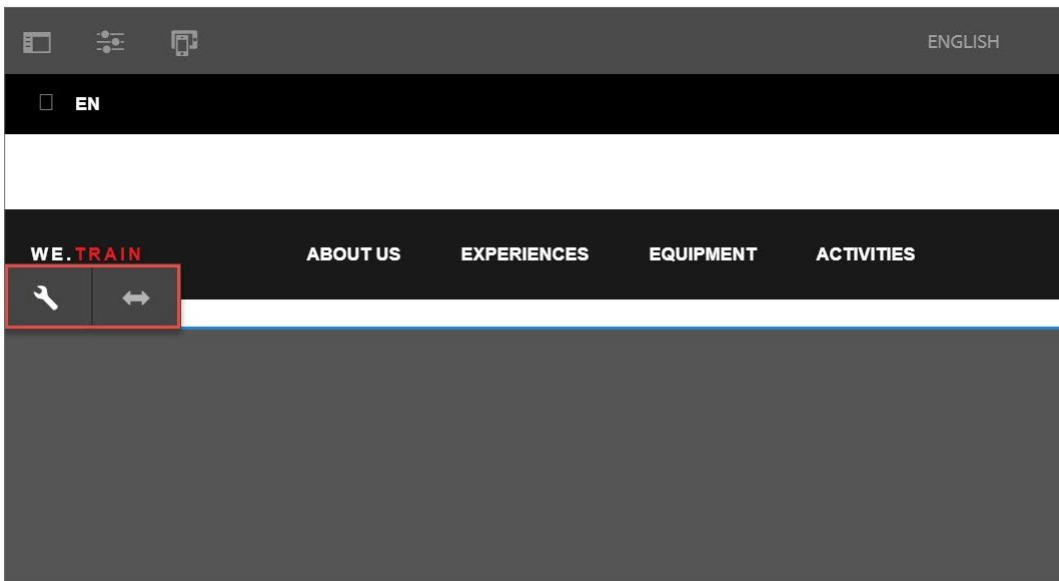
59. Select the **Page Hero** component, at the top of the page, and click the **Unlock** structure component icon from the component toolbar as shown. The **Unlock** dialog box opens.



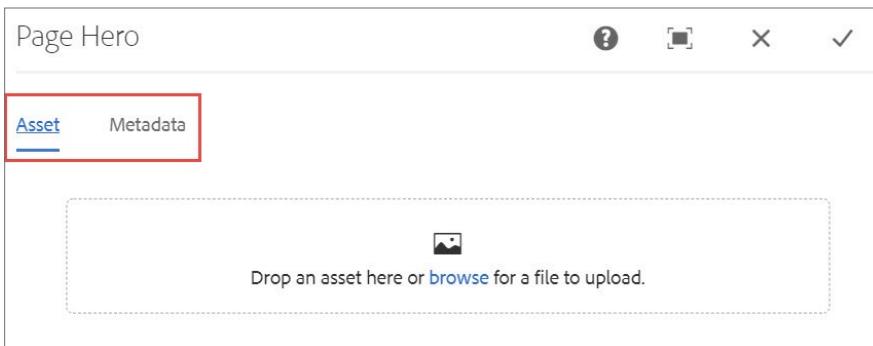
60. Click **Unlock**.

61. Navigate to the tab where the English page of the We.Train site is open and refresh the page or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter.

62. Hover over the **Page Hero[Root]** component and notice the component toolbar with Configure (wrench icon) and Layout (double-headed arrow) appears as shown:



63. Click the Configure (wrench) icon. The **Page Hero** dialog box with the **Assets** tab and the **Metadata** tab opens as shown:



Exercise 5: Add an editconfig node to the component

In this exercise, you will add an cq:editConfig node to the Page Hero component. Instead of recreating an existing node structure, you will copy the Core Image component, add it to the Page Hero component, and update the required values accordingly.

To add an editConfig node:

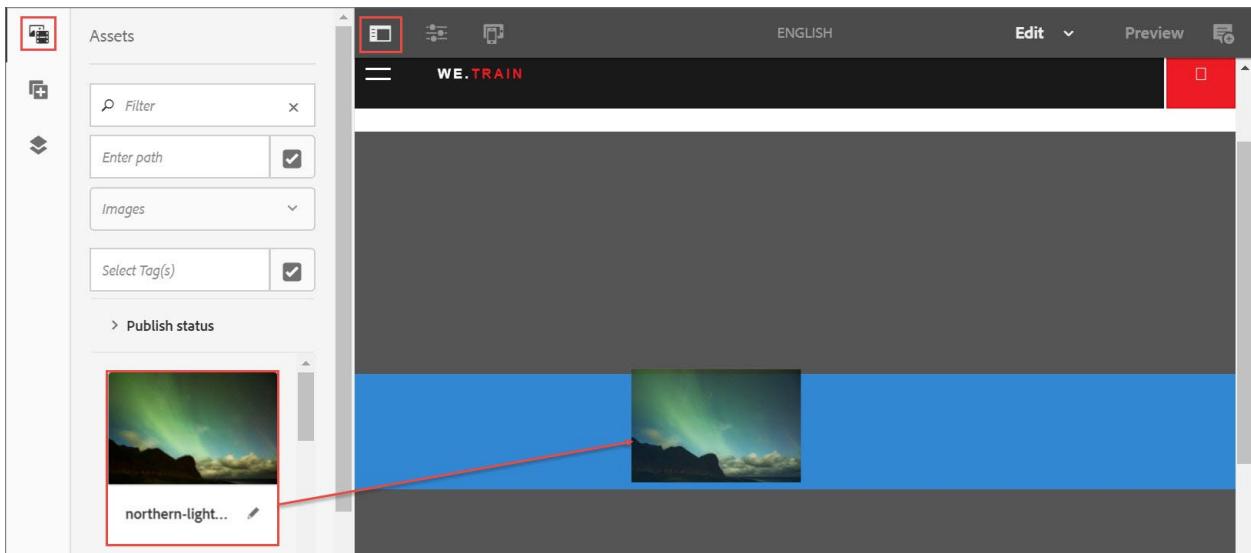
1. Ensure the CRXDE Lite tab is open or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/core/wcm/components/image/v2/image** node, select the **cq:editConfig** node, and then click **Copy** from the actions bar.
3. Navigate to the **/apps/training/components/structure** folder, select the **pagehero** component, and then click **Paste** from the actions bar. The cq:editConfig node is added.
4. Select the **cq:inplaceEditing** node that is below the **cq:editConfig** node, and then click **Delete** from the actions bar.
5. Click **Save All**.
6. Navigate to the **/apps/training/components/structure/pagehero/cq:editConfig/cq:dropTargets/image** node, select the **parameters** node, and update the following property:

Name	Type	Value
sling:resourceType	String	training/components/structure/pagehero

7. Click **Save All**.

To test if the cq:editConfig is added to the Page Hero component:

8. Navigate to the tab where the English page of the We.Train site is open and refresh the page or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The English page opens.
9. Click the Toggle Side Panel icon from the page toolbar, and then click the Assets icon. The Assets browser opens.
10. Drag an image from the Assets browser, and drop it onto the **Page Hero** component as shown. The image is added to the component.



Design Dialogs

Design dialogs are very similar to the dialogs used to edit and configure content, but they provide the interface for authors to configure and provide design details for the component.

The `cq:design_dialog (nt:unstructured)` node type is used to create a design dialog.

Design Configuration through Content Policies

In the template editor, policies are used to configure component design. For example, policies specify a component's design configurations, allowed components for a container, and mapping asset into components.

Configuring a template-editor's policy is similar to a Static template's design dialog. To define and access a new policy:

1. Create a policy configuration dialog - You can define the component's policy dialog by adding a `cq:design_dialog` to the component.
2. Configure the policy - After adding the design dialog, when you open the template in **Structure** mode, and click the component. The Policy icon will be available to configure the design properties in the component toolbar.
3. Access the policy - You can access the component's policy configuration through `ContentPolicyManager`.

Policy Storage

Policies are stored in following location by default:

/conf/project_name/settings/wcm/policies/component_name/policy_randomNumber

You can also store policies in the /apps or /libs folder. In this case the resource will be resolved in the following order: /conf, /apps, /libs.

The policies are stored centrally for a project. This gives the authors the ability to share a design policy among multiple templates. Template -> policy mapping is done by referring policy through:

cq:policy=/conf/project_name/settings/wcm/policies/component_name/policy_randomNumber

Exercise 6: Add a component client library to the component

To add a component clientlib:

1. Ensure the CRXDE Lite tab is open or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/component/structure** folder, select the **pagehero** component, click **Create** from the actions bar, and then select **Create Node** from the drop-down menu. The **Create Node** dialog box opens.
3. Enter and select the following:
 - a. **Name:** **clientlibs**
 - b. **Type:** **cq:ClientLibraryFolder**
4. Click **OK**, and then click **Save All** from the actions bar.
5. Select the **clientlibs** node and add the following properties:

Name	Type	Value
allowProxy	Boolean	true
categories	String[]	we.train.pagehero

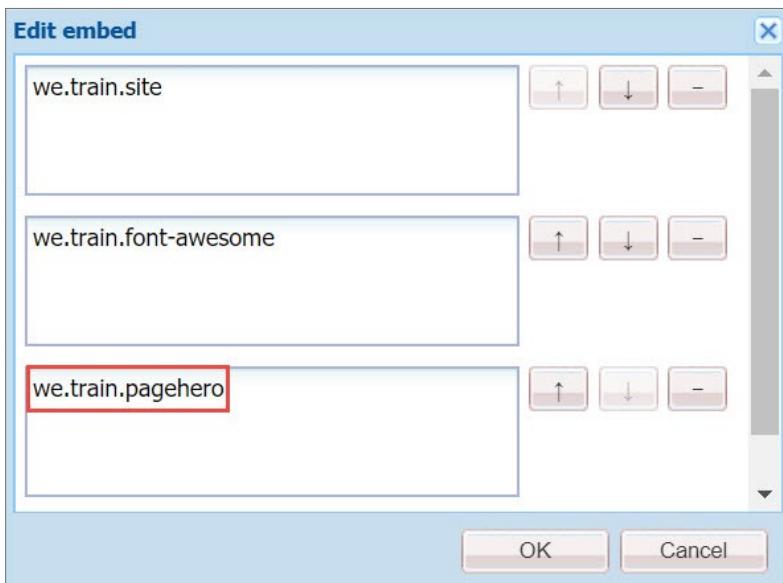
6. Click **Save All**.
7. Select the **clientlibs** folder, click **Create** from the actions bar, and then select **Create Folder** from the drop-down menu. The **Create Folder** dialog box opens.
8. Enter **css** in the **Name** field, click **OK**, and then click **Save All**.
9. Select the **css** folder, click **Create** from the actions bar, and then select **Create File** from the drop-down menu. The **Create File** dialog box opens.
10. Enter **pagehero.css** in the **Name** field, click **OK**, and then click **Save All**.

To add code to the pagehero.css file:

11. Double-click the **pagehero.css** to open the script for editing.
12. Update the **pagehero.css** script by replacing the existing code with new code (**pagehero.css**) from the **Exercise_Files** folder.
13. Select the **clientlibs** folder, click **Create** from the actions bar, and then select **Create File** from the drop-down menu. The **Create File** dialog box opens.
14. Enter **css.txt** in the **Name** field, click **OK**, and then click **Save All**.
15. Double-click the **css.txt** to open the file for editing.
16. Add `css/pagehero.css` code to the **css.txt** file:

To include the component client library at the top of the page, you need to embed it to the **clientlibs-base**.

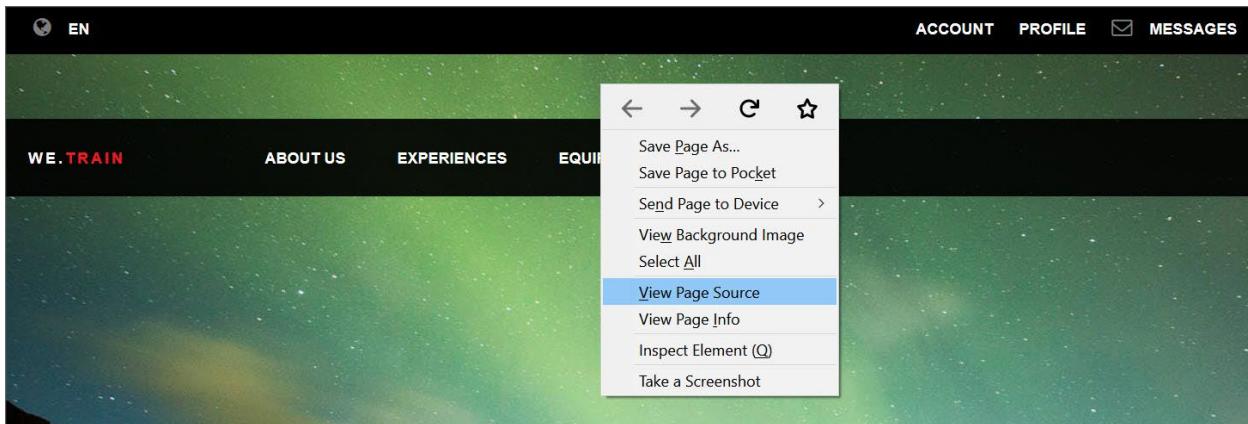
17. Navigate to the **/apps/training/clientlibs** folder, and select **client-base** folder.
18. In the **Properties** tab, double-click the **embed** property to edit its value. The **Edit embed** dialog box opens.
19. Click **+**, add **we.train.pagehero** in the field, and then click **OK**, as shown. This will add the css for the pagehero component at the top of the page.



20. Click **Save All**.

To observe the changes from the clientlib on the Page Hero component:

21. Navigate to the tab where the English page of the We.Train site is open and refresh the page or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter.
22. Click the Page Information icon from the page toolbar, and select **View as Published** from the drop-down menu. The page opens in a new tab.
23. Right-click on the English page, and click **View page source** from the list as shown. The source code opens in a new tab of the browser.



24. Click the clientlib-base.css line, as shown. Observe the css from the pagehero was added.

```
1 1<!DOCTYPE HTML>
2 2<html lang="en">
3 3  <head>
4 4    <meta charset="UTF-8">
5 5    <title>English</title>
6 6
7 7
8 8
9 9    <meta name="template" content="content-page"/>
10 10
11 11
12 12
13 13
14 14<link rel="stylesheet" href="/etc.clientlibs/training/clientlibs/client-base.css" type="text/css">
15 15
```

Exercise 7: Add a cq:htmlTag to the component

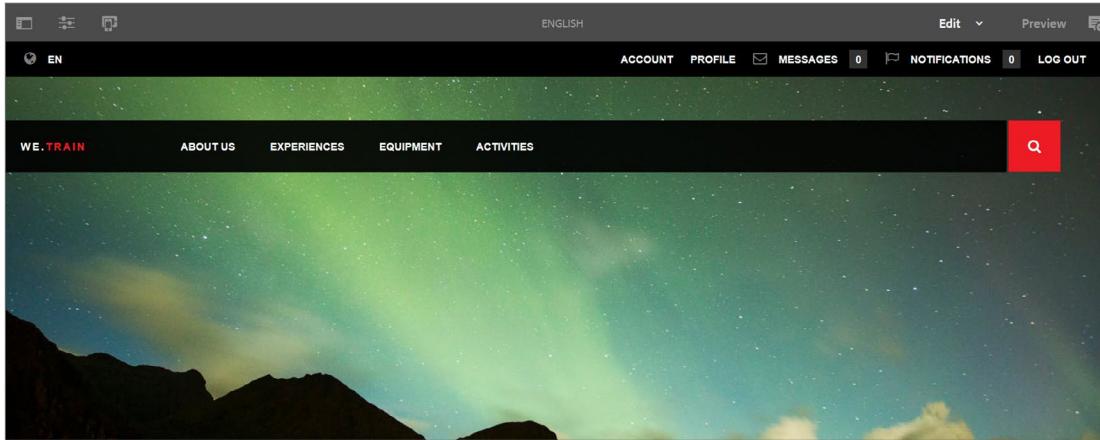
In this exercise, you will add a cq:htmlTag node to the pagehero component. This will wrap the HTML code of the component with a div element and a special css class that can be used to identify this component. This can be useful for future development of proxy components by allowing unique class names as per the extended component, allowing for unique client libraries for the same HTL code.

1. Ensure the **CRXDE Lite** tab is open or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components/structure** folder, select the **pagehero** component, click **Create** from the actions bar, and select **Create Node**. The **Create Node** dialog box opens.
3. Enter **cq:htmlTag** in the **Name** field, ensure the type **nt:unstructured** is selected, and then click **OK**. The node is created.
4. Select the **cq:htmlTag** node, and add the following properties:

Name	Type	Value
class	String	cmp cmp-pagehero
cq:tagName	String	div

5. Click **Save All**.

6. Navigate to the tab where the English page of the We.Train site is open and refresh the page or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The English page opens. Notice how the client library is applied to the Page Hero component.



Optional Exercise

You have a pagehero component built for your template so you can reuse this component as a content component for authors to drag and drop onto the page.

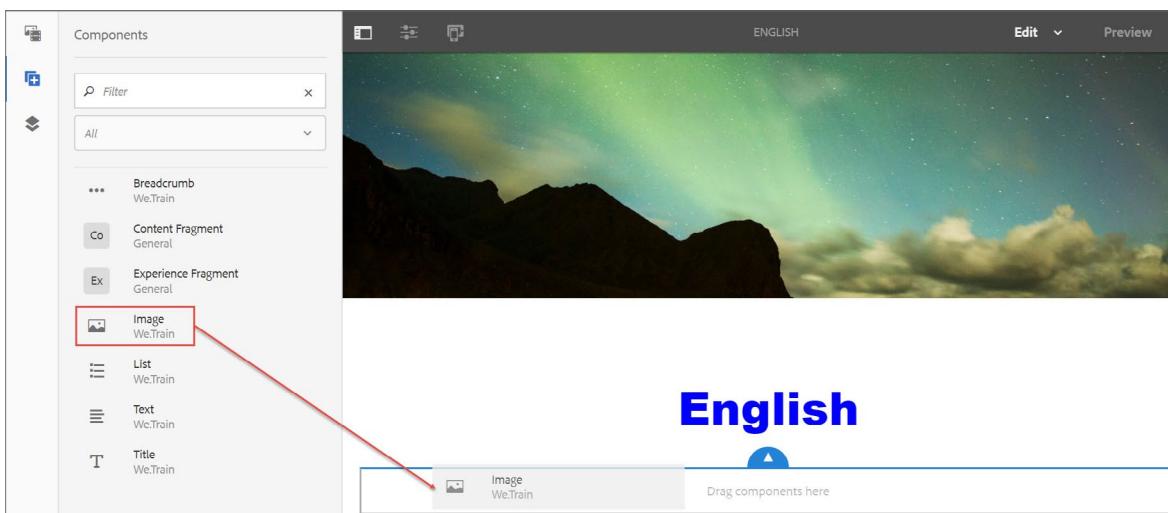
Create a proxy component called imagehero under /apps/training/components/content. This component should not have the css fix that moves it up 170px.

Hint: Create a cq:htmlTag node under the proxy component with a class name of cmp-imagehero to remove the css fix from the pagehero base component.

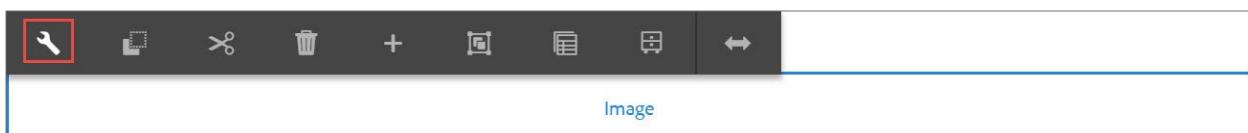
Exercise 8: Create a content policy for a proxy component

In this exercise, you will create a content policy to restrict authors from uploading the assets from their local computer on the for the proxy image component. To do that:

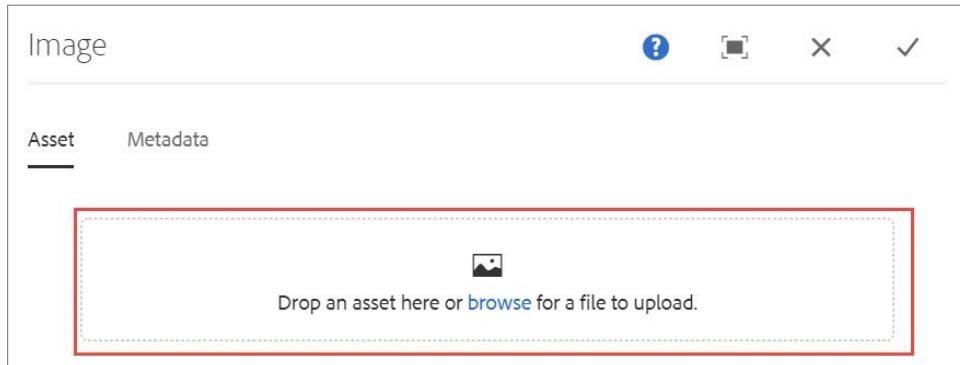
1. Navigate to the tab where the English page of the We.Train site is open or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The **English** page opens.
2. Ensure you are in **Edit** mode or click **Edit** from the page toolbar.
3. Click the Toggle Side Panel icon from the page toolbar. The panel opens.
4. Click the Components icon. The Components browser opens.
5. Drag the **Image** component and drop it below the title component on the page as shown. The component is added.



6. Select the **Image** component and click the Configure icon (wrench) from the component toolbar as shown. The **Image** dialog box opens.

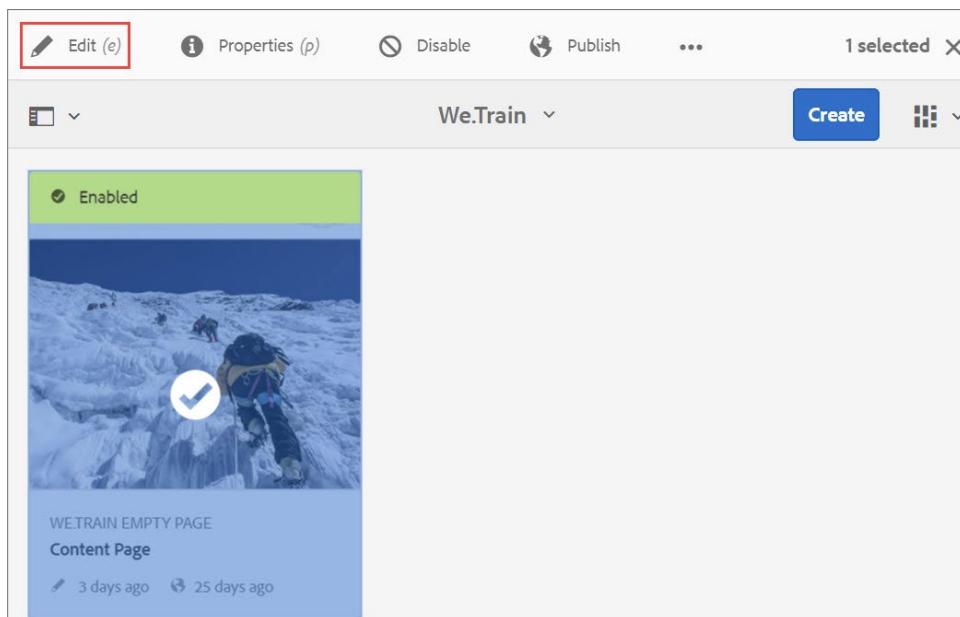


- Notice that on the **Asset** tab, you can add an asset from the AEM Assets console or upload an asset from your local computer as shown:

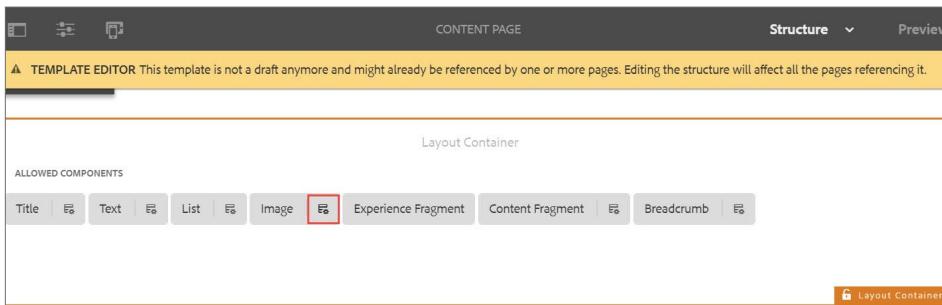


You will create a policy at the template level to restrict authors from uploading assets from their local computer. Authors need to use only the assets from the AEM Assets console on the Image component.

- Navigate to the tab where the AEM author instance is running.
- Click **Adobe Experience Manager** from the header bar, click the Tools icon, and then click **Templates**. The Templates console opens.
- Navigate to the **We.Train** folder, select the **Content Page** template, and then click **Edit** from the actions bar as shown.



11. Within the **Layout Container**, and click the Policy icon on the Image component as shown. The **Image** wizard opens.

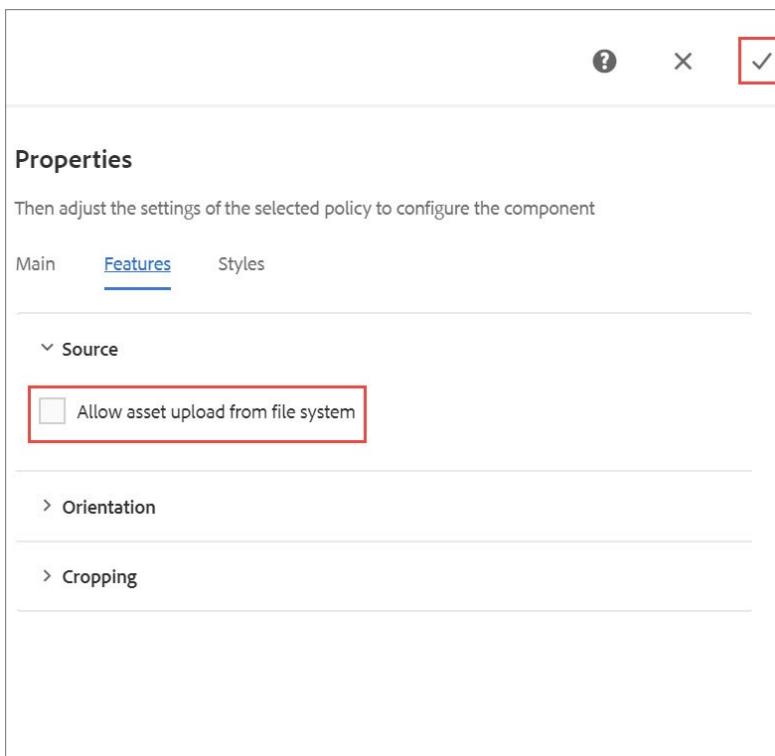


12. Enter **We.Train Image Policy** in the **Policy Title** field as shown. This will be the policy for the Image component, which helps create and retain a consistent policy across all templates.

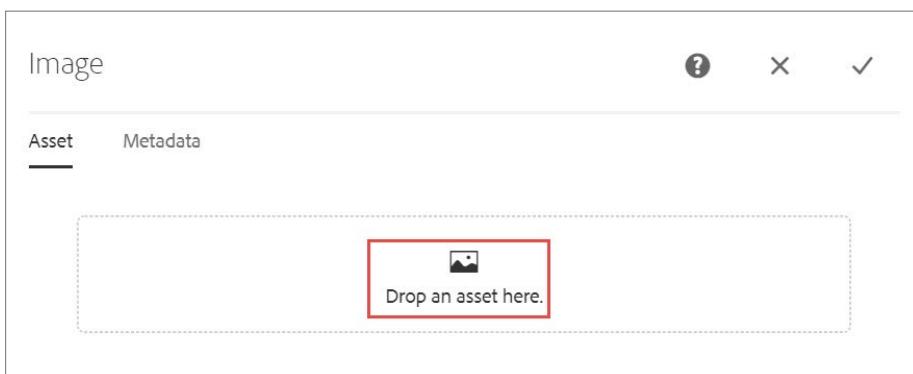
A screenshot of the 'Policy' dialog. It has a title 'Policy' and a sub-instruction 'First choose a policy to apply. Policies can be shared across templates'. Below this is a 'Select policy' section with a dropdown menu containing 'We.Train Image Policy'. To the right of the dropdown are a '+' button and a trash bin icon. Below the dropdown is a 'Policy Title *' field, which contains the text 'We.Train Image Policy'. This entire 'Policy Title *' field is highlighted with a red border.

13. In the **Properties** section, click the **Features** tab, ensure the **Allow asset upload from file system** checkbox is clear.

14. Click Done (checkmark) icon as shown. You will be taken back to the Content Page template.



15. Navigate to the tab where the English page of the We.Train site is open and refresh the page or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The English page opens.
16. Select the **Image** component that you added before.
17. Click Configure (wrench) icon from the component toolbar, as shown. The **Image** dialog box opens.
18. Notice that the Asset tab no longer has the option to upload the asset from your local computer. You can still upload an asset from the AEM Assets console:



19. Click X (Cancel) to close the dialog box.

Exercise 9: Create a design dialog for a component

In this exercise, you will create a new component and add a design dialog to it.

1. Ensure the **CRXDE Lite** tab is open or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components** folder, select the **structure** folder, click **Create** from the actions bar, and select **Create Component**. The **Create Component** dialog box opens.
3. Enter the following:
 - a. **Label:** footer
 - b. **Title:** We.Train Footer
 - c. **Group:** We.Train.Structure
4. Click **Next** and click **OK**. The component is created.
5. Click **Save All**.
6. Navigate to the **footer** component, select the **footer.jsp**, click **Rename** from the actions bar, and change the file name to **footer.html**.
7. Click **Save All** from the actions bar.

To add code to the footer.html page:

8. Double-click the **footer.html** to open the script for editing.
9. Add new code from **basic_footer.html** in the **Exercise_Files** folder.
10. Select the **footer** component, click **Create** from the actions bar, and then click **Create File**. The **Create File** dialog box opens.
11. Enter **footer.js** in the **Name** field, click **OK**, and then click **Save All**.
12. Double-click the **footer.js** to open the script for editing.

13. Add new code from **basicfooter.js** in the **Exercise_Files** folder. Notice that the business logic is almost identical to the business logic that you added to the header.js code except that you are adding a few pages to display on the footer as shown:

```

01. "use strict";
02. use(function() {
03.   var pages = [];
04.   var root = currentPage.getAbsoluteParent(2);
05.
06.   //make sure that we always have a valid set of returned items
07.   //if navigation root is null, use the currentPage as the navigation root
08.   if(root == null){
09.     root = currentPage;
10.   }
11.
12.   log.info("#####NavRoot Page: " + root.title);
13.
14.   var it = root.listChildren(new Packages.com.day.cq.wcm.api.PageFilter());
15.   while (it.hasNext()) {
16.     var pageObject = it.next();
17.     pages.push(pageObject);
18.   }
19.
20.   //Get the current year
21.   var curYear = (new Date()).getFullYear();
22.
23.   return {
24.     items: pages,
25.     currentYear: curYear
26.   }
27. });

```

14. Click **Save All**.

You need to add a cq:dialog node to add the footer component to the template.

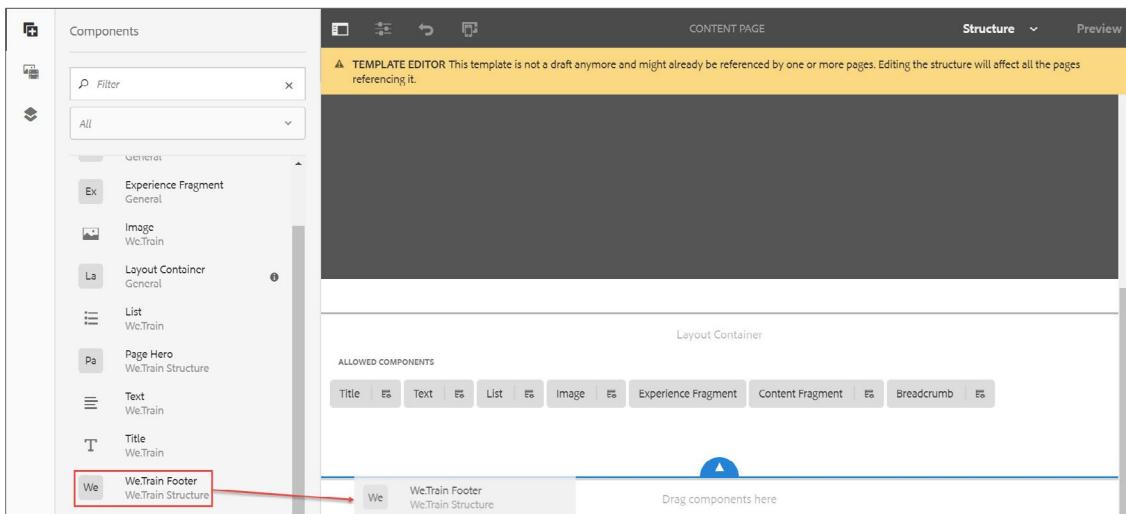
15. Select the **footer** component, click **Create** from the actions bar, and then click **Create Node**. The **Create Node** dialog box opens.
16. Enter the following:
- Name: cq:dialog**
 - Ensure **nt:unstructured** is selected
17. Click **OK** and click **Save All**.

To add the footer component to the template:

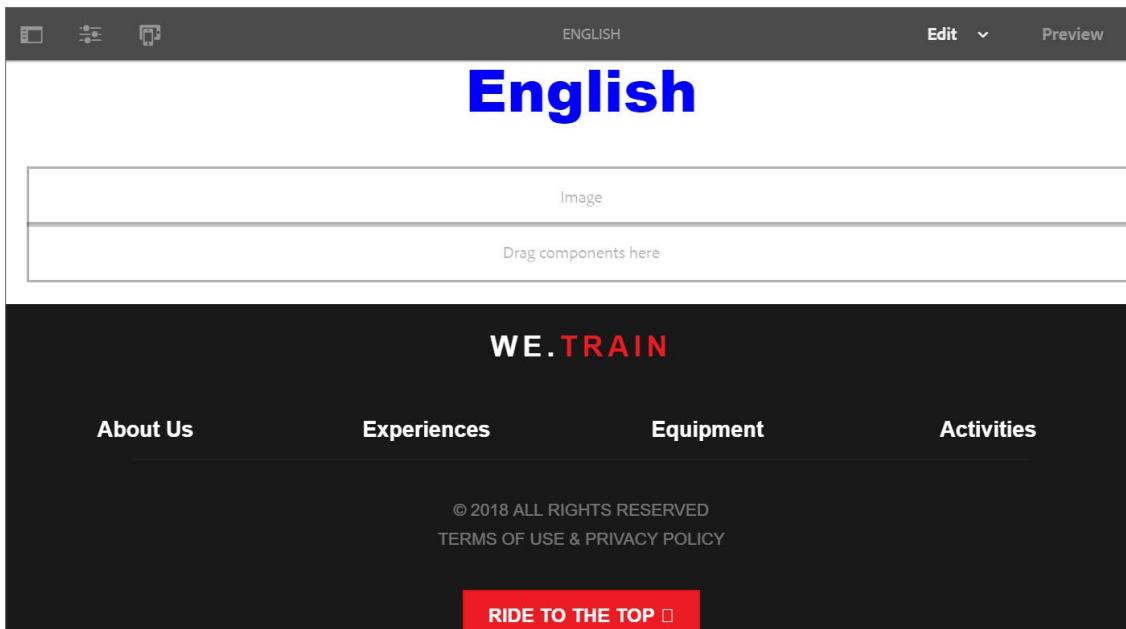
18. Navigate to the tab where the AEM author instance is running.
19. Click **Adobe Experience Manager** from the header bar, click the Tools icon, and then click **Templates**. The Templates console opens.
20. Navigate to the **We.Train** folder, select the **Content Page** template, and then click **Edit** from the actions bar.
21. Click the Toggle Side Panel icon from the page toolbar, and then click the Components icon. The components browser opens.

22. Notice that the We.Train Footer is available.

23. Drag the **We.Train Footer** from Components browser onto the **Drag components here** placeholder as shown. The component is added.



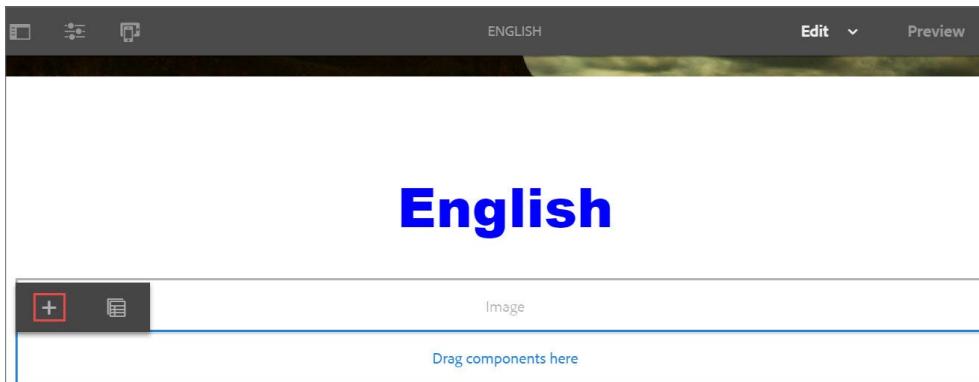
24. Navigate to the tab where the English page of the We.Train site is open and refresh the page or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The English page opens. Notice how the footer is rendering the basic navigation pages as shown:



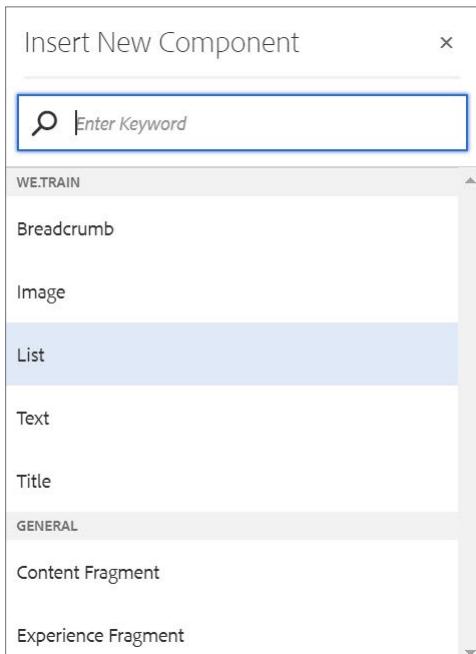
Sometimes, organizations may want to able to update and change the links on their footers without involving the development team. This can be achieved by creating a custom content policy for the footer.

To create a content policy, you will build a design dialog that helps the template authors specify the links for the footer. Rather than building the design dialog from scratch, you will build it based on the list component dialog. To see this dialog:

25. Ensure the **English** page is open.
26. Select the **Drag components here** area and click the + icon from the component toolbar as shown. The **Insert New Component** dialog box opens.



27. Select the **List** component as shown. The component is added to the page.

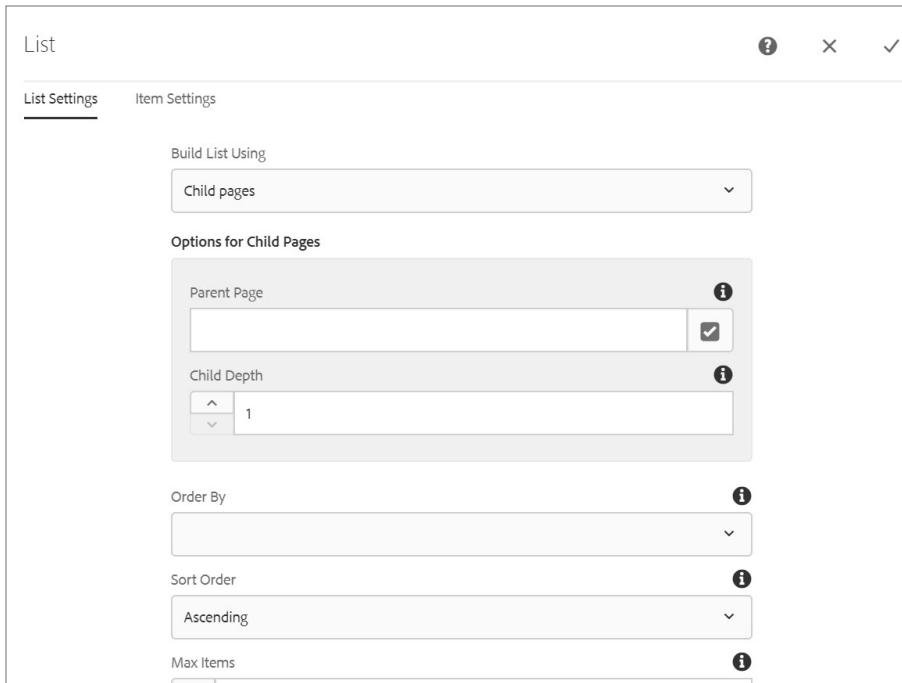


To view the design dialog:

28. Select the **List** component and click the Configure (wrench) icon from the component toolbar.

The **List** dialog box opens.

29. Observe the dialog box and the options it provides:



30. Click X (Cancel) to close the dialog box.

To create a design dialog:

31. Ensure the **CRXDE Lite** tab is open or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.

32. Navigate to the **/apps/training/components/structure** folder, select the **footer** node, click **Create** from the actions bar, and select **Create Node**. The **Create Node** dialog box opens.

33. Enter the following:

a. **Name:** cq:design_dialog

34. Ensure the **Type** is **nt:unstructured**.

35. Click **OK**. The node is created.

36. Click **Save All**.

37. Select the **cq:design_dialog** node and add the following properties:

Name	Type	Value
sling:resourceType	String	cq/gui/components/authoring/dialog
jcr:title	String	Footer Policy

38. Click **Save All**.

39. Navigate to the **/apps/core/wcm/components/list/v2/list/cq:dialog** node, select the **content** node, click **Copy** from the actions bar.

40. Navigate to the **/apps/training/components/structure/footer** node, select the **cq:design_dialog** node, and click **Paste** from the actions bar.

41. Click **Save All**.

Next, remove all the extra fields from the dialog and customize it to your footer component.

42. Navigate to the **/apps/training/components/structure/footer/cq:design_dialog/content/items/tabs/items** node, select the **itemSettings** node, and click **Delete** from the actions bar. The node is deleted.

43. Click **Save All**.

44. Navigate to the **/apps/training/components/structure/footer/cq:design_dialog/content/items/tabs/items/listSettings/items/columns/items/column/items** node, select the **setStatic** node, and click **Copy** from the actions bar.

45. Navigate to the **/apps/training/components/structure/footer/cq:design_dialog/content/items** node and click **Paste** from the actions bar.

46. Click **Save All**.

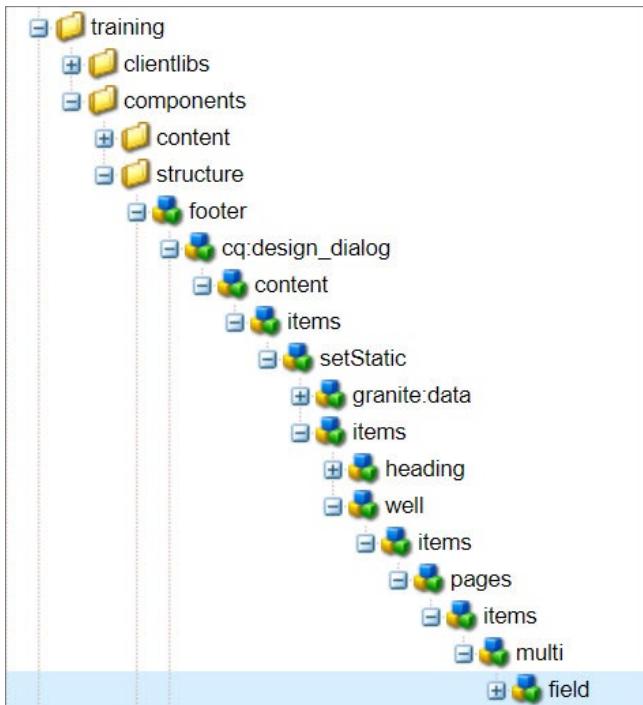
47. Navigate to the **/apps/training/components/structure/footer/cq:design_dialog/content/items/setStatic** node.

48. In the **Properties** tab, select the **granite:class** property, right-click, and then click **Delete**. The property is deleted from the node.

49. Click **Save All**.

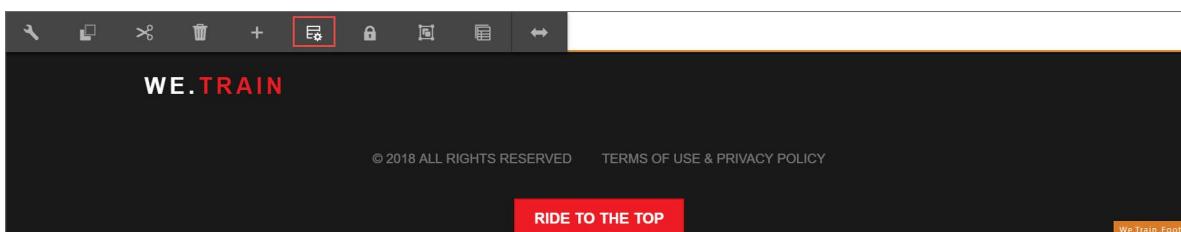
50. Navigate to the **/apps/training/components/structure/footer/cq:design_dialog/content/items** node, select the **tabs** node, and then click **Delete** from the actions bar.

51. Navigate to the `/apps/training/components/structure/footer/cq:design_dialog/content/items/setStatic/items/well/items/pages/items/multi/field` node. Notice that the property name= `./pages` holds the fixed list. Your folder structure should look similar to the one shown in the below screenshot:



Now that you have built the design dialog, you need to update the business logic to support it.

52. Navigate to the `/apps/training/components/structure/footer` node, double-click the **footer.js** to open the script for editing.
53. Update the **footer.js** with **policyfooter.js** script that is available in the **Exercise_Files** folder.
54. Click **Save All**.
55. Navigate to the tab where the **Content Page** template is open.
56. Select the **Footer** component and click the **Policy** icon from the component toolbar as shown.
The **Footer Policy** dialog box opens.



57. In the **Policy** section, enter the title as **We.Train Footer Policy** as shown:

Policy

First choose a policy to apply. Policies can be shared across templates

Select policy

We.Train Footer Policy

Policy Title *

We.Train Footer Policy

Policy Description

Add a description

58. In the **Properties** section, notice the design dialog that you built is available:

59. Under **Options for Fixed List**, click **Add**, and enter **/content/we-retail/us/en/experience** path in the field.

60. Click Done (checkmark) icon, as shown:

Properties

Then adjust the settings of the selected policy to configure the component

Options for Fixed List

/content/we-retail/us/en/experience

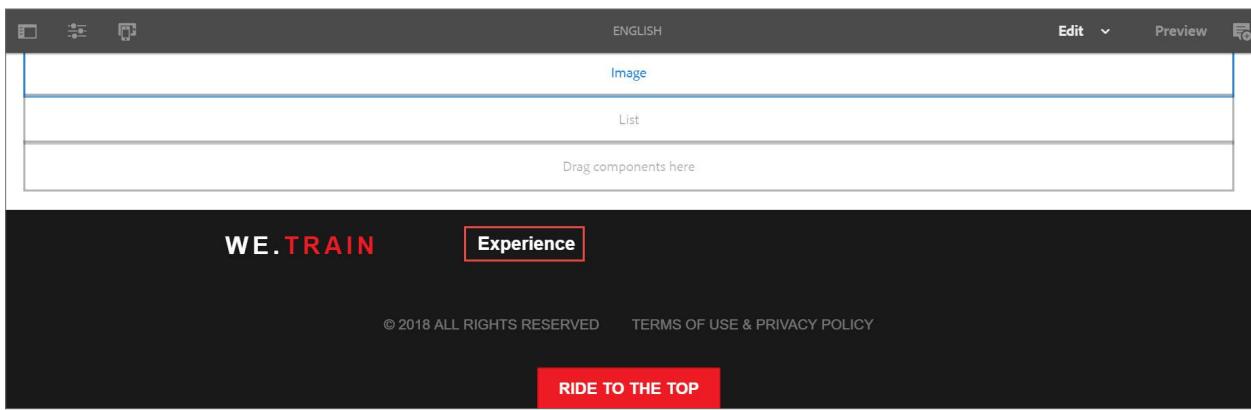
Add

?

X

✓

61. Navigate to the tab where the English page of the We.Train site is open and refresh the page. Notice the footer is updated with the link to Experience page that you added in the previous step.



62. Click **Preview** from the page toolbar, click **Experience** in the footer to navigate to the Experience page within the We.Retail site from the English page.
63. Open different pages of the We.Train site and observe the changes in content policy.



Creating Custom Components

Introduction

Adobe Experience Manager (AEM) components are used to hold, format, and render the content made available on your webpages. Depending on the component you want to implement in your project, you can extend or customize an existing component, rather than defining and developing the entire structure from scratch.

Objectives

After completing this course, you should be able to:

- Explain the important features of components
- Create a custom component
- Add a dialog field validation to the Stockplex component
- Add a base style to the Stockplex component
- Add a style system to the Stockplex component
- Add a Sling model as the business logic
- Add a selector to the component
- Create localization information
- Test the localized content
- Extend the core component
- Extend the Navigation UI

Features of Components

By default, AEM provides a wide range of component implementations on a standard instance. You can develop custom components according to your project requirements by extending and customizing the existing components.

The features of the components in AEM are:

- **Component Placeholder:** Helps validate if the component is on the page and is available only in page Edit mode. The placeholder is not available in page Preview mode and nor is it available on the publish instance. If the content has been added to the component, the placeholder does not appear on the page.
- **Dialog Validation:** Provides a straightforward validation framework that helps create custom form element validators and interfaces them programmatically. Registering custom validators is done by calling a jQuery based `$.validator.register` method. The register method takes a single JavaScript object literal argument. The parameter looks for four properties: `selector`, `validate`, `show`, and `clear`, of which only `selector` is required.
- **Style System:** Enables a template author to define style classes in the content policy of a component so that a content author can select them when editing the component on a page. These styles can be alternative visual variations of a component, making it more flexible. This eliminates the need to develop a custom component for each style or to customize the component dialog to enable such style functionality. It leads to more reusable components that can be quickly and easily adapted to the needs of content authors without any AEM back-end development.
- **Sling Models for Components:** When developing an AEM project, you can define a model object (a Java object) and map that object to Sling resources. A Sling Model is implemented as an OSGi bundle. A Java class located in the OSGi bundle is annotated with `@Model` and the adaptable class (for example, `@Model(adaptables = Resource.class)`). The data members (Fields) use `@Inject` annotations. These data members map to node properties.

- **Sling Model Exporter:** Enables new annotations to be added to Sling Models that define how the Model can be exported as JSON. With Sling Model Exporter, you can obtain the same properties as a JSON response, without creating a Sling Servlet. You need to export the Sling Model by using the Jackson exporter. The Sling Model Exporter can be used as a Web service or as a REST API.
- **Selectors:** Provides a way to choose, which variation of the script should be rendered when a user requests a page. During the resource resolution step, if the first character in the request URL after the resource path is a dot (.), the string after the dot up to (but not including the last dot before the next slash character or the end of the request URL) comprises the selectors. If the resource path spans the complete request URL, no selectors exist. If only one dot follows the resource path before the end of the request URL or the next slash, no selectors exist.
- **i18n Translation:** Provides a **Strings & Translations** console for managing the various translations of texts used in the component UI. The translator tool helps manage English strings and their translations. The dictionaries are created in the repository. From this console, you can search, filter, and edit the English and translated texts. You can also export dictionaries to XLIFF format for translating, and then import the translations back into the dictionaries. It is also possible to add the i18n dictionaries to a translation project from this console. You can either create a new one or add to an existing project.

Exercise 1: Create a custom component

You need to install the **adls-training-project-v5.0.zip** package on your AEM author instance. The package contains the required training project structure, page component with clientlibs, template, and site structure that was already created for you.

In the following exercises, you will create a custom component that will display stock information based on the user specification. The user will have the ability to change the design as well as export the content as JSON. The component that you will build is called the Stockplex component.

To create the Stockplex component:

1. Click **Adobe Experience Manager** from the header bar. You will be in the **Tools** console.
2. Click **CRXDE Lite**. The **CRXDE Lite** page opens.
3. Navigate to the **/apps/training/components** folder.
4. Select the **content** folder, click **Create** from the actions bar, and then click **Create Component** from the drop-down menu. The **Create Component** dialog box opens.
5. Enter the following in the dialog box:
 - a. **Label:** **stockplex**
 - b. **Title:** **StockPlex**
 - c. **Description:** We.Train Complex Stock Component
 - d. **Group:** **We.Train**
6. Click **Next**, and then click **OK**. The component is created.
7. Click **Save All** from the actions bar.

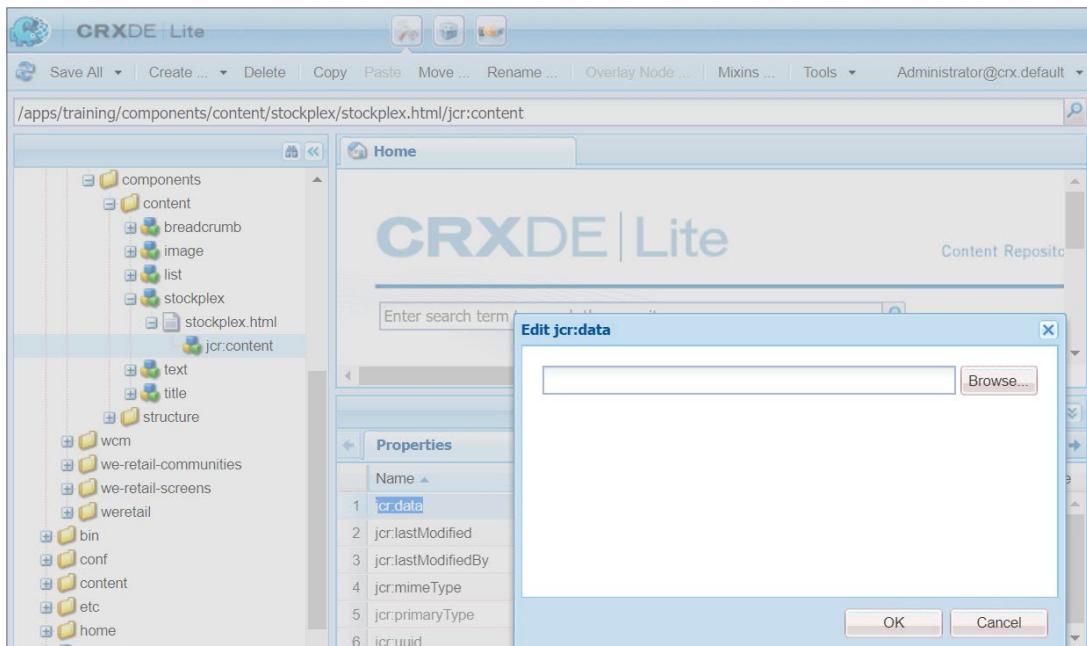


Note: You can also press Ctrl+S (Windows users) and Command+S (Mac users) to save your work in CRXDE Lite.

8. Navigate to the stockplex component, select the **stockplex.jsp**, click **Rename** from the actions bar, and change the file name to **stockplex.html**.
9. Click **Save All** from the actions bar.
10. Double-click the **stockplex.html** to open the script for editing.

To add code to the stockplex.html page:

11. Select the **jcr:content** node that is below **stockplex.html**. The **Properties** tab opens.
12. Double-click the **jcr:data** property. The **Edit jcr:data** dialog box opens.
13. Click **Browse** as shown. The **Open** dialog box opens.



14. Navigate to the **Exercise_Files** folder on your file system.

15. Select `placeholder_stockplex.html`, and then click **Open**. The **Edit jcr:data** dialog box opens.
16. Click **OK**. The default sample code is replaced with the code in `stockplex.html`.
17. Open the **stockplex.html** to view the code that you added. Observe the following code snippet of `stockplex.html`.

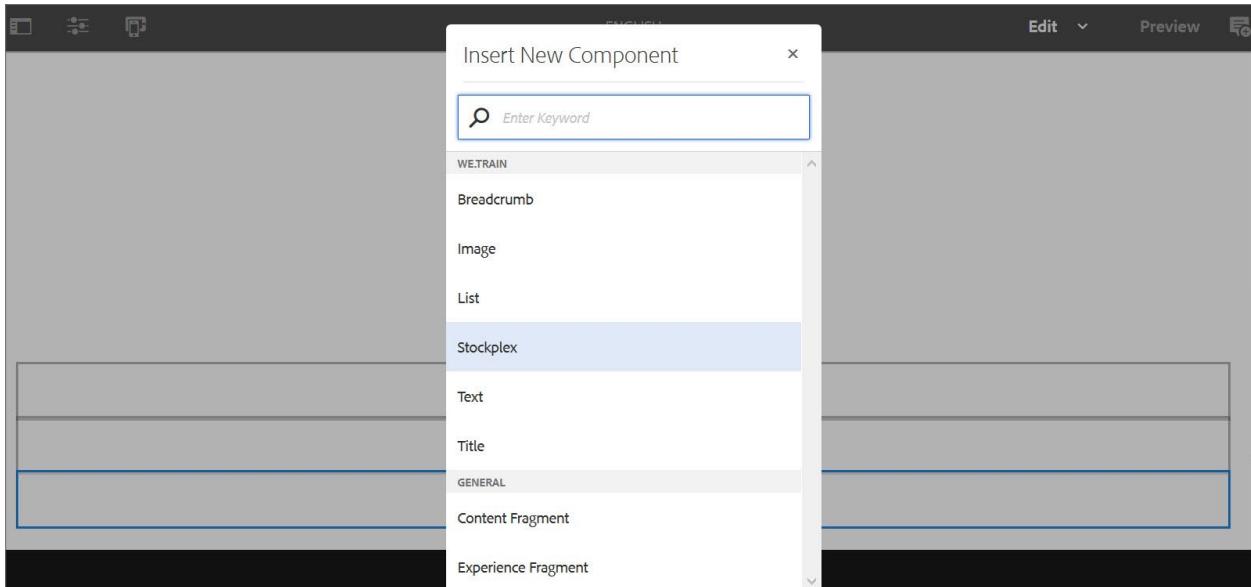
```
01. | <!-- If there is no stock symbol added to the dialog, create a component placeholder -->
02. | <div class="cq-placeholder" data-emptytext="Stockplex Component"></div>
```

18. Click **Save All**.

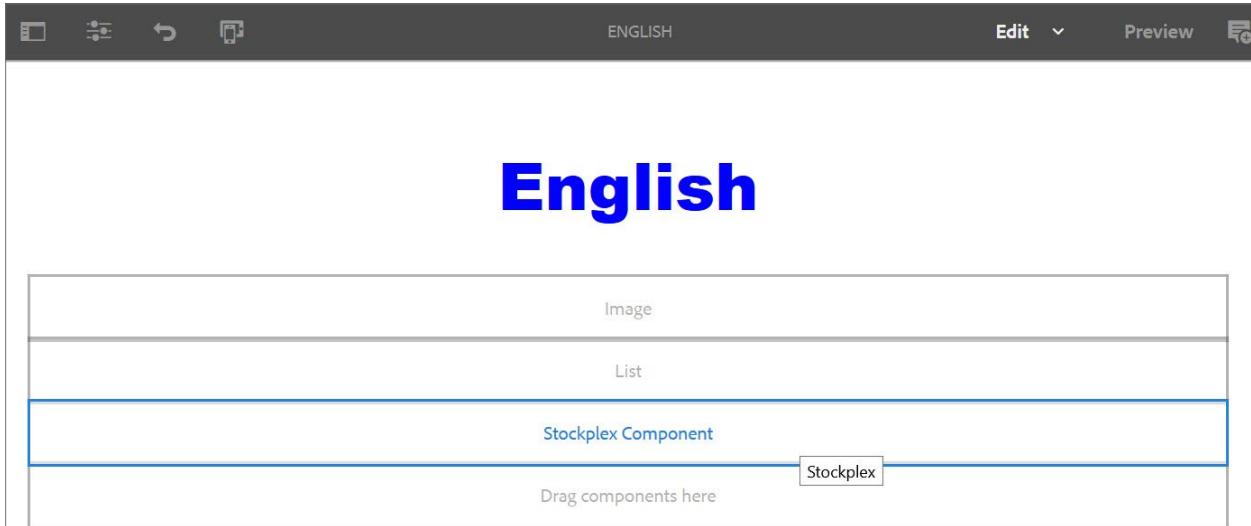
You will create a cq:dialog node to enable the component delete functionality and to provide a means for the author to enter content.

19. Select the **stockplex** component, click **Create** from the actions bar, and then click **Create Node**. The **Create Node** dialog box opens.
20. Enter the following:
 - a. **Name:** `cq:dialog`
 - b. **Type:** `nt:unstructured`
21. Click **OK** and then click **Save All** from the actions bar.
22. Open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The **English** page of We.Train site opens.
23. Click **Edit** in the upper right to open the page in edit mode.
24. Scroll down the page, click the **Drag components here** placeholder, and then click the Insert component (plus) icon from the component toolbar. The **Insert New Component** dialog box opens.

25. Notice that the Stockplex component is automatically available on the page, since the component is added to the We.Train group. If the Stockplex component does not appear on the page, you need to enable the component in the Layout Container of the Content Page template.



26. Select the **Stockplex** component from the list to add it to the page as shown:



Exercise 2: Add a dialog field validation to the Stockplex component

To add dialog field validation to the stockplex component:

1. Ensure you are in CRXDE Lite or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components/content** folder.
3. Select the stockplex node and double-click the **stockplex.html** to open the script for editing.
4. Update the **stockplex.html** script by replacing the existing code with new code (**dialog_stockplex.html**) from the **Exercise_Files** folder.
5. Open the **stockplex.html** to view the code that you added. Notice that in the following code snippet, you are checking the properties of the object to see if an author can add content into the dialog.

```

01. <div class="cmp-stockplex" data-sly-test.symbol="${properties.symbol}">
02.   <div class="cmp-stockplex__symbol">${properties.symbol}</div>
03.   <div class="cmp-stockplex__currentPrice">Current Value: 300</div>
04.
05.   <div data-sly-test.summary="${properties.summary}">
06.     <h3>Summary: ${properties.summary}</h3>
07.   </div>
08.
09.   <div class="cmp-stockplex__button">
10.     <a href="#">
11.       <button>Placeholder</button>
12.     </a>
13.   </div>
14.
15.   <div class="cmp-stockplex__details" data-sly-test.summary="${properties.showStockDetails}">
16.     <ul class="column1">
17.       <li>Request Date: November 13, 2018</li>
18.       <li>Open Price: 300</li>
19.       <li>Range High: 303</li>
20.     </ul>
21.     <ul class="column2">
22.       <li>Range Low: 299</li>
23.       <li>Close: 303</li>
24.       <li>Volume: 30000</li>
25.     </ul>
26.   </div>
27.
28. </div>
29.
30. <!-- If there is no stock symbol added to the dialog, create a component placeholder -->
31. <div data-sly-test="${!symbol}" class="cq-placeholder" data-emptytext="Stockplex Component"></div>
```

6. Click **Save All** from the actions bar.

To create a dialog:

7. Select the **cq:dialog** node that is below the **stockplex** component.
8. Right-click the **cq:dialog** node and click **Delete** from the actions bar.
9. Click **Save All** from the actions bar.
10. Navigate to the **/apps/core/wcm/components/title/v2/title/cq:dialog** node, select the **cq:dialog** node, and then click **Copy** from the actions bar.
11. Navigate to the **/apps/training/components/content** folder, select the **stockplex** node, and then click **Paste** from the actions bar.
12. On the **cq:dialog** node:
 - a. Update the **jcr:title** to **Stockplex**.
 - b. Right-click in the **helpPath** property, and then click **Delete** from the list.
13. Click **Save All** from the actions bar.
14. Navigate to the **/apps/training/components/content/stockplex/cq:dialog/content/items/tabs/items/properties/items/columns/items/column/items** node.
15. Select the **types** node and click **Delete** from the actions bar.
16. Click **Save All** from the actions bar.
17. Select the **defaulttypes** node and click **Delete** from the actions bar.
18. Navigate to the bottom of the **cq:dialog** structure, and go to **/apps/training/components/content/stockplex/cq:dialog/content/items/tabs/items/properties/items/columns/items/column/items/title** node.

19. Select the **title** node, click **Rename** from the actions bar, and change it to **symbol**.

20. Update the following properties of the **symbol** node as shown:

Name	Type	Value
name	String	./symbol
fieldLabel	String	Stock Symbol
fieldDescription	String	Enter a 4-character stock symbol

The screenshot shows the CRXDE Lite interface with the following details:

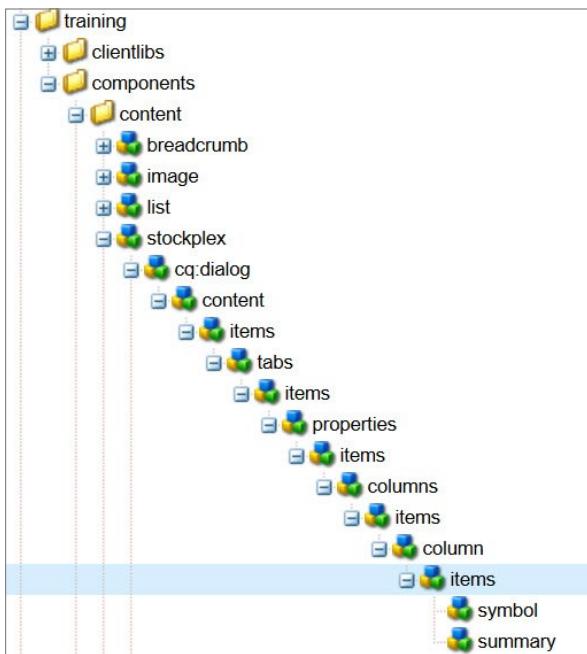
- Node Structure:** The left sidebar shows a tree view of nodes under the path /apps/training/components/content/stockplex/cq:dialog/content/items/tabs/items/columns/items/column/items/symbol. The 'symbol' node is selected.
- Properties View:** The right panel displays the properties of the selected 'symbol' node. The 'Properties' tab is active, showing the following table:

Name	Type	Value	Protected	Mandatory	Multiple	Auto Created
fieldDescription	String	ADBE	false	false	false	false
fieldLabel	String	Stock S...	<input checked="" type="checkbox"/>	false	false	false
jcr:primaryType	Name	nt:unstr...	<input checked="" type="checkbox"/>	true	false	true
name	String	./summ...	false	false	false	false
sling:resourceType	String	granite/...	<input checked="" type="checkbox"/>	false	false	false

21. Select the **symbol** node, and then click **Copy** from the actions bar.

22. Select the **items** node that is above the **symbol** node, and click **Paste** from the actions bar.

23. Select the **Copy of symbol** node and rename it to **summary**. The summary and symbol nodes are siblings as shown:



24. Update the following properties of the **summary** node:

Name	Type	Value
name	String	./summary
fieldLabel	String	Summary of Stock
fieldDescription	String	Enter a summary description of the stock

25. Select the **symbol** node, and then click **Copy** from the actions bar.

26. Select the **items** node that is above the **symbol** node, and click **Paste** from the actions bar.

27. Select the **Copy of symbol** node and rename it to **stockdetails**. The summary and stockdetails nodes are siblings.

28. Update the following properties of the **stockdetails** node:

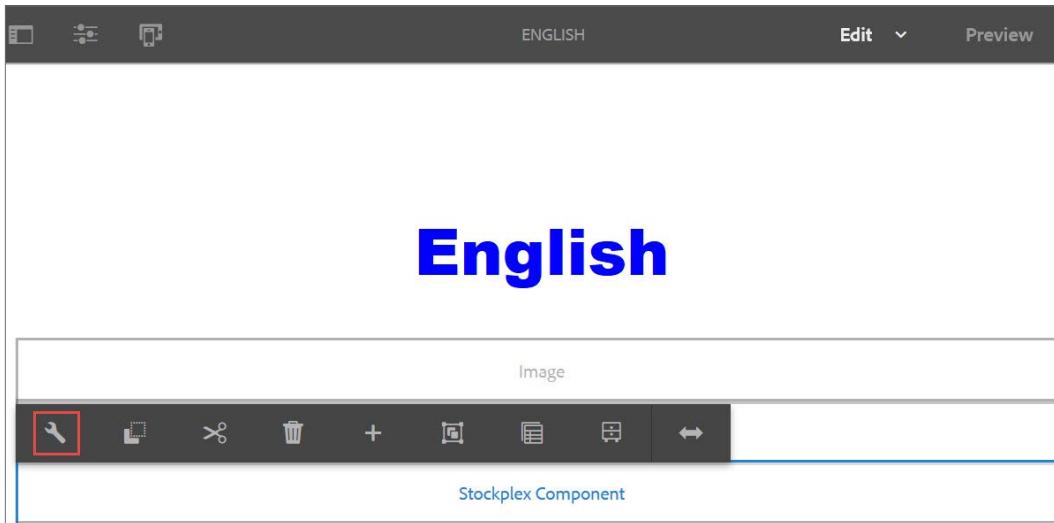
Name	Type	Value
name	String	./showStockDetails
fieldDescription	String	Include requestDate, upDown, openPrice, range high/low, volume, and others
sling:resourceType	String	granite/ui/components/coral/foundation/form/checkbox

29. Add the following new properties to the stockdetails node:

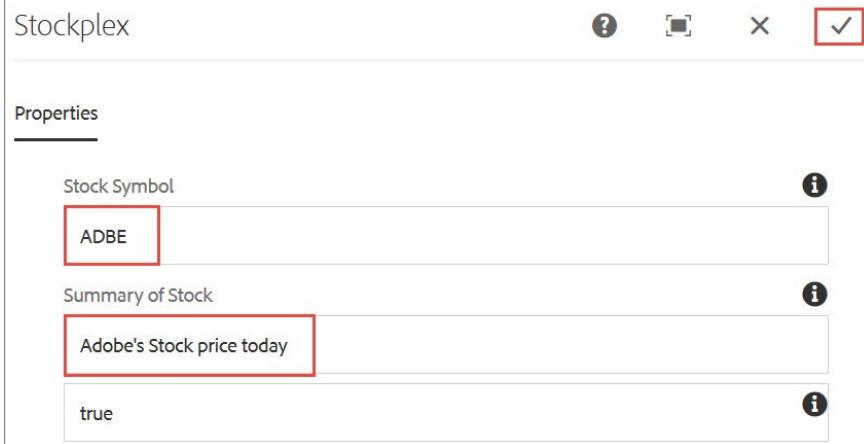
Name	Type	Value
text	String	Include Stock Details
value	Boolean	true

30. Select the **fieldLabel** property, right-click it, and then click **Delete** from the list.

31. Click **Save All** from the actions bar.
32. Navigate to the tab where the English page is open or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The **English** page of We.Train site opens.
33. Select the **Stockplex** component, and click the **Configure** (wrench) icon as shown. The **Stockplex** dialog box opens.



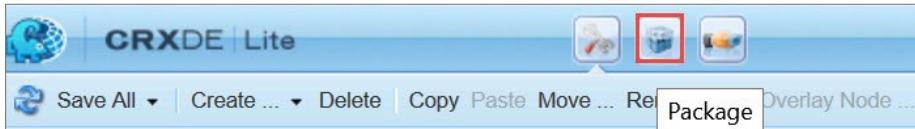
34. Enter the following values:
 - a. **Symbol:** ADBE
 - b. **Summary:** Adobe's Stock price today
35. Click the Done (checkmark) icon as shown. You will be taken back to the **English** page.



Now that you can view the content rendered on the stockplex component, add validation for the Stock symbol.

36. Ensure you are in CRXDE Lite or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.

37. Click the Package icon from the header bar as shown. The **CRX Package Manager** page opens.

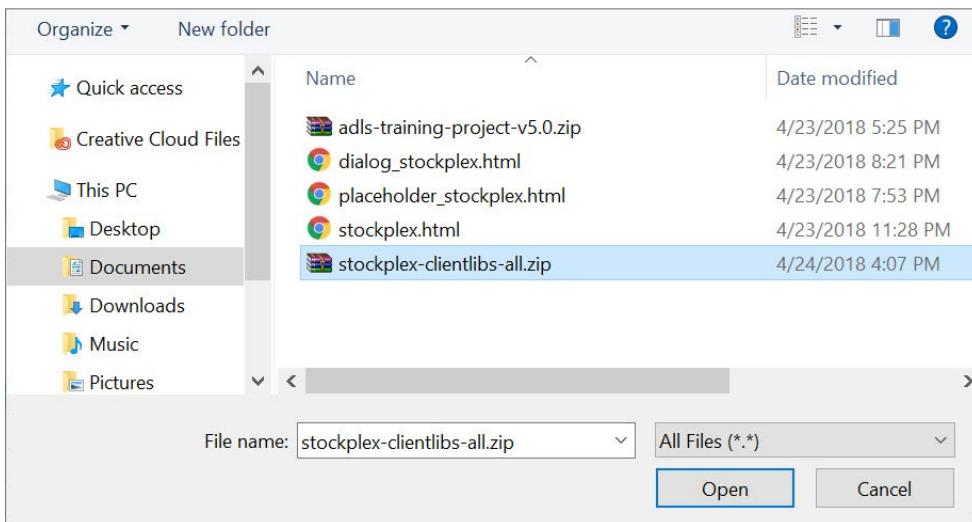


38. Click **Upload Package** from the actions bar as shown. The **Upload Package** dialog box opens.



39. Click **Browse** in the dialog box. The **Open** dialog box opens.

40. Navigate to the **Exercise_Files** folder on your system, select the **stockplex-clientlibs-all.zip** package, and then click **Open** as shown. You will be taken back to the **Upload Package** dialog box.

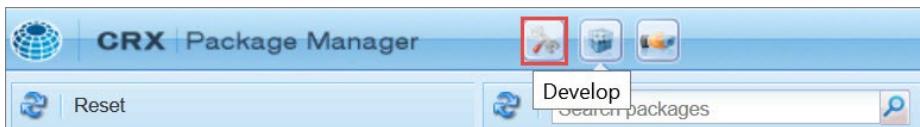


41. Click **OK** in the **Upload Package** dialog box. The package is uploaded.

42. Click **Install** from the actions bar below the **stockplex-clientlibs-all.zip** package. The **Install Package** dialog box opens.

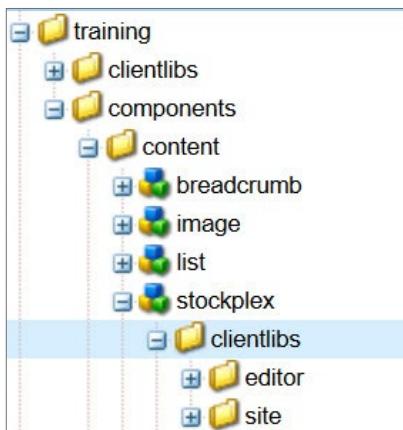
43. Click **Install** again. The package is installed.

44. Click the Develop icon from the header bar as shown. The CRXDE Lite page opens.



45. Navigate to the `/apps/training/components/content/stockplex/clientlibs` folder. Notice that the content package contains two client libraries for the Stockplex component:

- a. editor: A client library containing a validation script for a dialog.
- b. site: A css/js file for the component design.

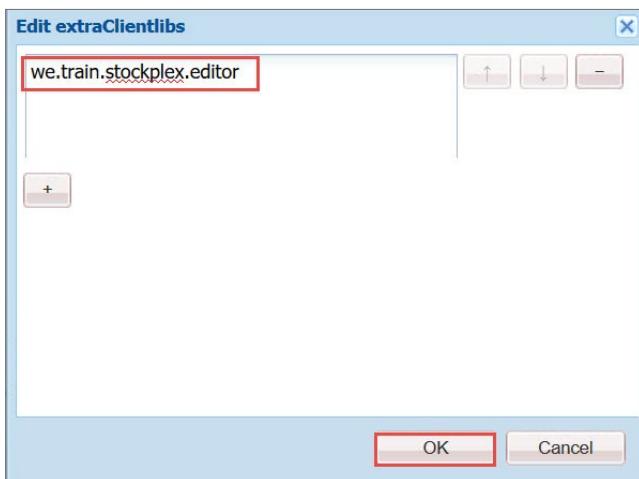


To use the stockplex editor client library, you need to add it to the dialog box.

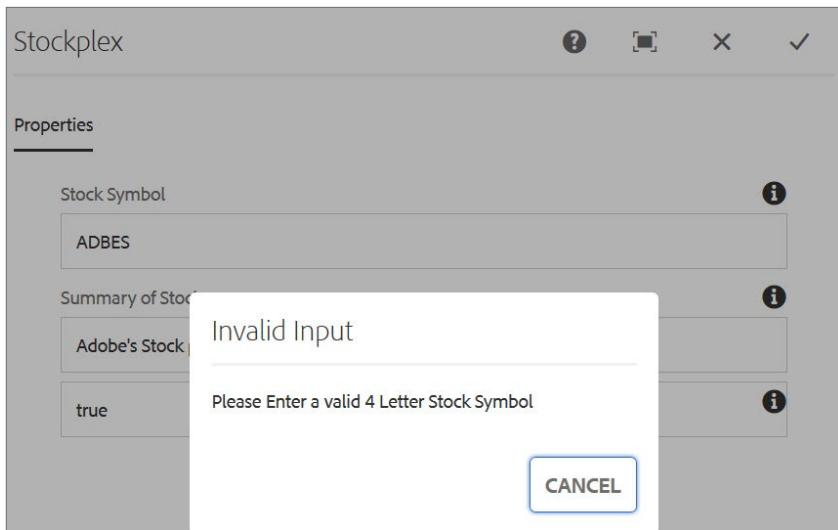
46. Navigate to the `/apps/training/components/content/stockplex/cq:dialog` node.

47. In **Properties** section, double-click in the **extraClientlibs**. The **Edit extraClientlibs** dialog box opens.

48. Update the value to `we.train.stockplex.editor` and click **OK** as shown. The existing value is replaced with the new value.



49. Click **Save All** from the actions bar.
50. Navigate to the tab where the **English** page is open.
51. Select the **Stockplex** component and click the Configure (wrench) icon. The **Stockplex** dialog box opens.
52. In the **Stock Symbol** field, enter more than four characters, and then click the Done (checkmark) icon. The **Invalid Input** dialog box with the message opens as shown:



53. Click **Cancel** to close the dialog box, and then click the Cancel (X) icon in the Stockplex dialog box.

Exercise 3: Add a base style to the Stockplex component

In the previous exercise, you installed the client libraries for the stockplex component. In this exercise, you will add a design to the Stockplex component.

1. Ensure you are in CRXDE Lite or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components/content/stockplex/clientlibs/site** node. Notice that the client libs for the design of the stockplex component are available in this folder.

To use the design in stockplex component, you need to add the design to the HTL.

3. Navigate to the **/apps/training/components/content** folder.
4. Select the **stockplex** node and double-click the **stockplex.html** to open the script for editing.

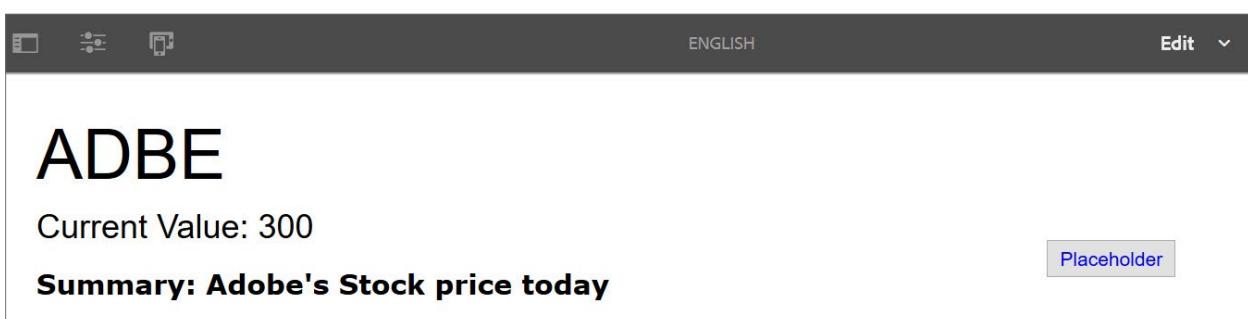
5. Update the **stockplex.html** script by replacing the existing code with new code (**clientlibs_stockplex.html**) from the **Exercise_Files** folder.
6. Click **Save All**.
7. Open the **stockplex.html** to view the code that you added. Observe the following code snippet.

```

01. <sly data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html"
02.     data-sly-call="${clientlib.css @ categories='we.train.stockplex'}" />
03.
04. <div class="cmp-stockplex" data-sly-test.symbol="${properties.symbol}">
05.     <div class="cmp-stockplex__column1">
06.         <div class="cmp-stockplex__symbol">${properties.symbol}</div>
07.         <div class="cmp-stockplex__currentPrice">Current Value: 300</div>
08.
09.         <div class="cmp-stockplex__summary" data-sly-test.summary="${properties.summary}">
10.             <h3>Summary: ${properties.summary}</h3>
11.         </div>
12.
13.         <div class="cmp-stockplex__button">
14.             <a href="#">
15.                 <button>Placeholder</button>
16.             </a>
17.         </div>
18.     </div>
19. </div>
20. <div class="cmp-stockplex__column2">
21.     <div class="cmp-stockplex__details" data-sly-test.summary="${properties.showStockDetails}">
22.         <ul>
23.             <li class="cmp-stockplex__details-item">
24.                 <span class="cmp-stockplex__details-title">Request Date:</span>
25.                 <br />
26.                 <span class="cmp-stockplex__details-data">November 13, 2018</span>
27.             </li>
28.             <li class="cmp-stockplex__details-item">
29.                 <span class="cmp-stockplex__details-title">Open Price:</span>
30.                 <br />
31.                 <span class="cmp-stockplex__details-data">300</span>
32.             </li>
33.             <li class="cmp-stockplex__details-item">
34.                 <span class="cmp-stockplex__details-title">Range High:</span>
35.                 <br />
36.                 <span class="cmp-stockplex__details-data">303</span>
37.             </li>
38.             <li class="cmp-stockplex__details-item">
39.                 <span class="cmp-stockplex__details-title">Range Low:</span>
40.                 <br />
41.                 <span class="cmp-stockplex__details-data">299</span>
42.             </li>
43.             <li class="cmp-stockplex__details-item">
44.                 <span class="cmp-stockplex__details-title">Close:</span>
45.                 <br />
46.                 <span class="cmp-stockplex__details-data">303</span>
47.             </li>
48.             <li class="cmp-stockplex__details-item">
49.                 <span class="cmp-stockplex__details-title">Volume:</span>
50.                 <br />
51.                 <span class="cmp-stockplex__details-data">30000</span>

```

8. Navigate to the tab where the English page is open or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The **English** page of We.Train site opens.
9. Refresh the English page. Notice that the stockplex component has a design as shown:



Exercise 4: Add a style system to the Stockplex component

In this exercise, you will provide content authors the ability to change the design of a component in the authoring environment without having to write the code. The selectable designs must be precreated by the design team to help authors pick and choose different design they want to use on the page.

1. Ensure you are in CRXDE Lite or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.

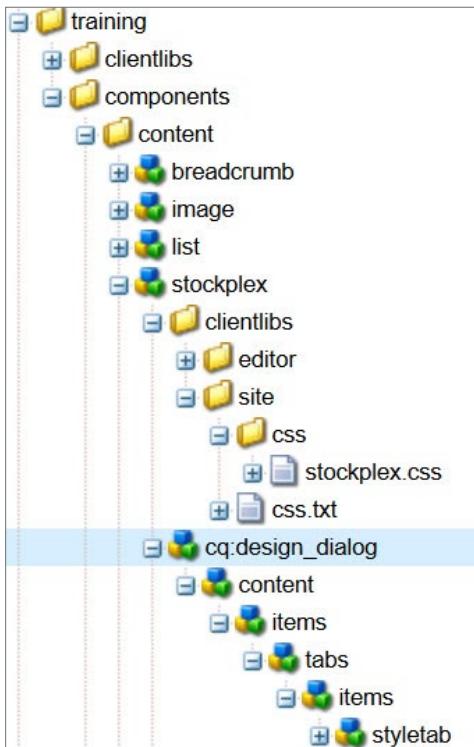
To view the different designs that you want to make available to authors:

2. Navigate to the **/apps/training/components/content/stockplex/clientlibs/site/css/stockplex.css** node. Double-click **stockplex.css** and notice **cmp-stockplex--italic**, **cmp-stockplex--bold**, and **cmp-stockplex--red**. These classes are not used in the base stockplex code and need to be added to the style system.

To help a template author define styles, you need to add the style system to the stockplex content policy.

3. Navigate to the **/apps/core/wcm/components/text/v2/text** node, select the **cq:design_dialog** node, and click **Copy** from the actions bar.
4. Navigate to the **/apps/training/components/content/stockplex** node and click **Paste** from the actions bar. The node is added to the stockplex component.
5. Click **Save All** from the actions bar.
6. Select the **cq:design_dialog** node, and on the **Properties** tab, update the **jcr:title** property to **Stockplex**.

7. Click **Save All** from the actions bar.
8. Navigate to the `/apps/training/components/content/stockplex/cq:design_dialog/content/items/tabs/items` node, select the **plugins** node, and click **Delete** from the actions bar. The node is deleted as shown:

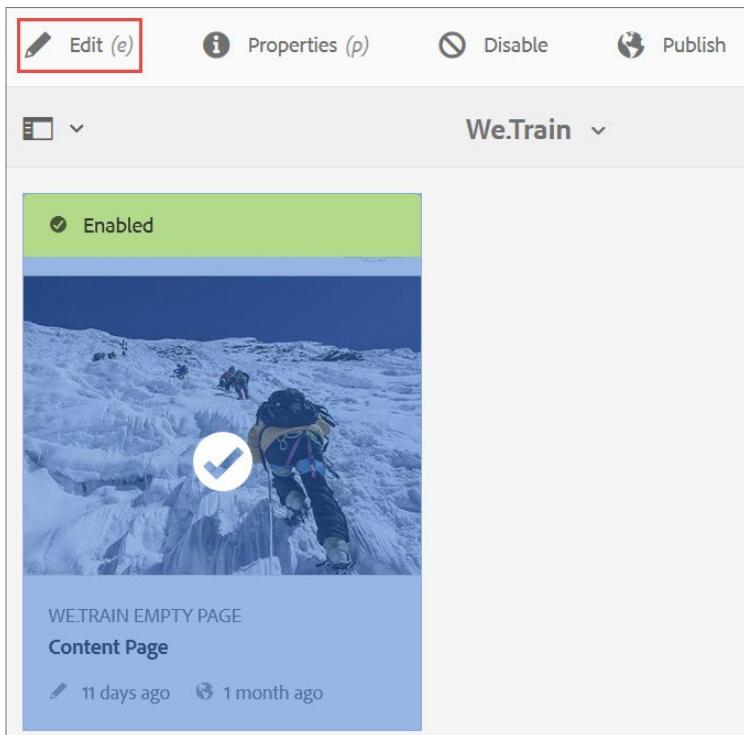


9. Click **Save All** from the actions bar.

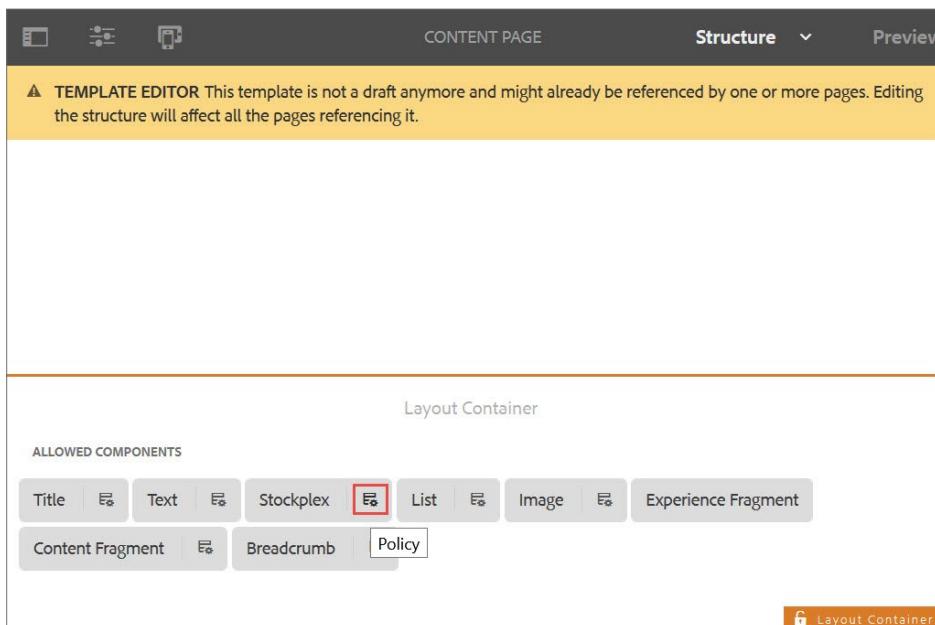
Now, as a template author, you will add the style system to the Stockplex content policy.

10. Click **CRXDE Lite** from the header bar. You will be taken back to the **Navigation** page.
11. Click the Tools (hammer) icon. The **Tools** console opens.

12. Ensure you are in the **General** section and click the **Templates** tile. The **Templates** console opens.
13. Navigate to the **We.Train** folder, select the **Content Page** template, and then click **Edit** from the actions bar as shown. The **Content Page** template editor opens in a new tab of the browser.



14. In the **Layout Container** component, click the Policy icon next to the **Stockplex** component as shown. The **Stockplex** policy wizard opens.



15. In the **Policy** section, enter **Stockplex Style System Policy** in the **Policy Title** field as shown:

Policy

First choose a policy to apply. Policies can be shared across templates

Select policy

Stockplex Style System Policy

Policy Title *

Stockplex Style System Policy

Policy Description

Add a description

Other templates also using the selected policy

There is no item.

16. In the **Properties** section, under the **Styles** tab, click the **Add** button below **Allowed Styles** field.

17. Enter the **Group Name** as **Font Types** and select the **Styles can be combined** checkbox.

18. Click the **Add** button that is below the **Styles** field as shown. A new style field is created.

Properties

Then adjust the settings of the selected policy to configure the component

Styles

Default CSS Classes

Allowed Styles

Font Types Styles can be combined i trash

Styles

<input type="text" value="Style Name"/>	<input type="text" value="CSS Classes"/>	trash
---	--	--------------------

Add

Add

19. Enter the **Style Name** as **Italic** and **cmp-stockplex--italic** as the **Css Classes** as shown:

Allowed Styles

Font Types Styles can be combined i trash

Styles

<input type="text" value="Italic"/>	<input type="text" value="cmp-stockplex--italic"/>	trash
-------------------------------------	--	--------------------

Add

Add

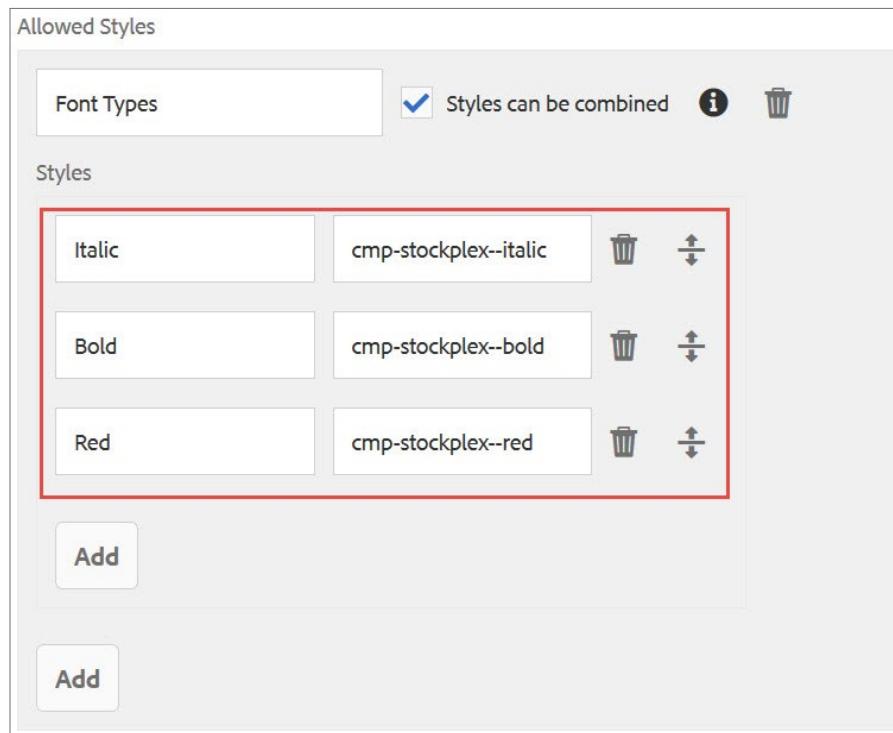
20. Click the **Add** button that is below the **Italic** field. A new style field is created.

21. Enter the **Style Name** as **Bold** and **cmp-stockplex--bold** as the **Css Classes**.

22. Click the **Add** button that is below the **Bold** field. A new style field is created.

23. Enter the **Style Name** as **Red** and **cmp-stockplex--red** as the **Css Classes**.

Your styles should look similar to the one shown in the following screenshot:

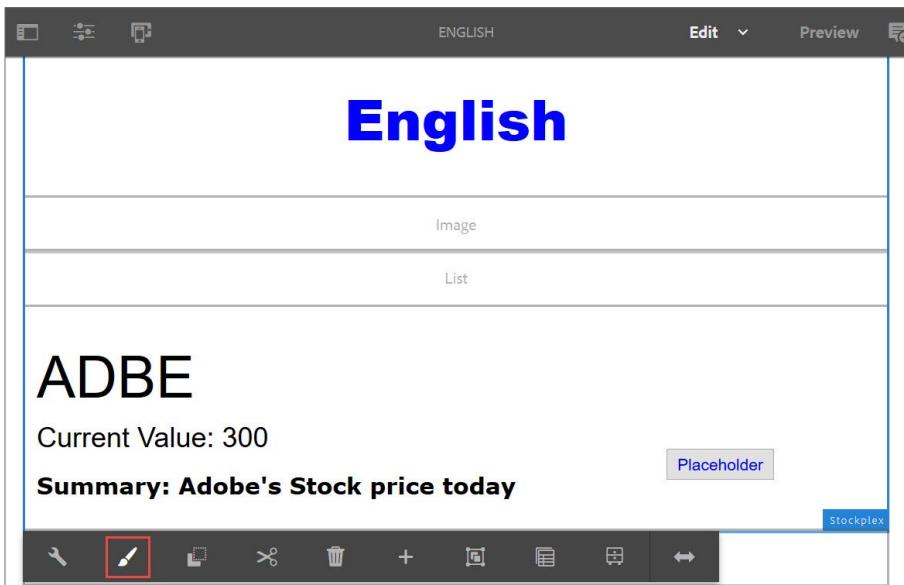


24. Click the Done (checkmark) icon in the Stockplex wizard. You will be taken back to the **Content Page** template editor.

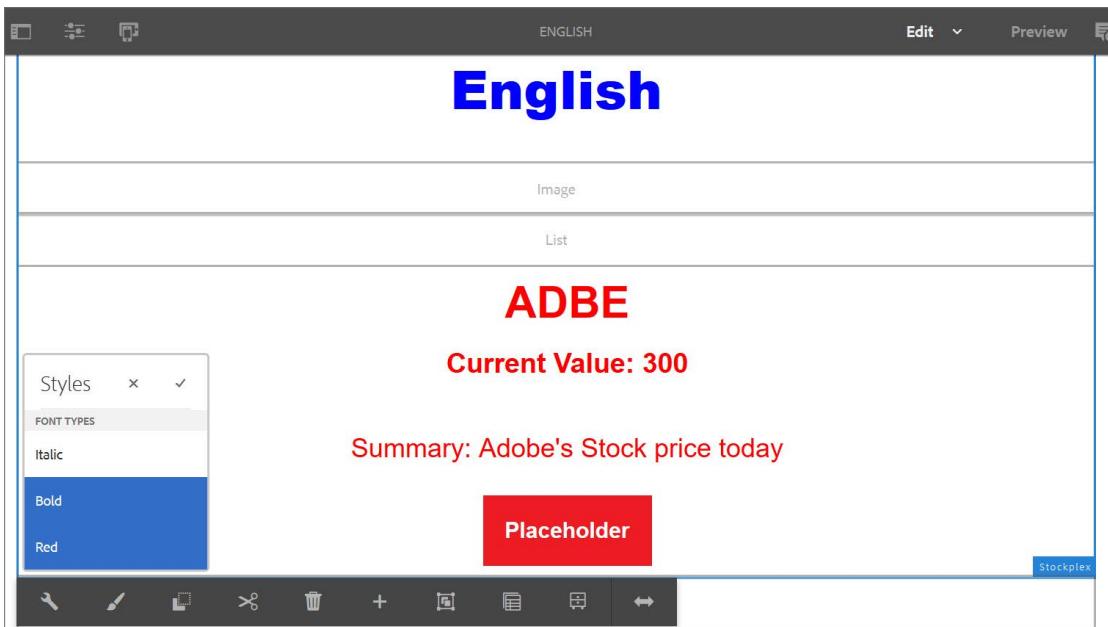
Now that you have the css developed by the designer, and the style system is added by the template editor, authors can use the style system.

25. Navigate to the tab where the English page is open or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The **English** page of We.Train site opens.
26. Refresh the English page.

27. Select the **Stockplex** component. You should now see a new Styles (paintbrush) icon in the component toolbar as shown.



28. Click the Styles (paintbrush) icon from the actions bar, add some styles to the stockplex component, and observe the changes, as shown:



29. If you are unable to view any changes to the styles in the stockplex component, refresh the English page.

Exercise 5: Add a Sling Model as the business logic

In this exercise, you will use a Sling Model that was added to your AEM instance when you uploaded the **we.train.core-0.0.1-SNAPSHOT.jar** in the Create a basic header component exercise.

To see if this Stockplex model is already installed:

1. Click **Adobe Experience Manager** from the header bar.
2. Click the Tools (hammer) icon, click **Operations**, and then click **Web Console**. The **Adobe Experience Manager Web Console Configuration** opens.
3. Click **Status** from the actions bar, and then select **Sling Models** from the drop-down menu. You should see the Stockplex Model under **Sling Models Bound to resource Types For Requests** and **Sling Models Exporter Servlets**.

```
Sling Models Bound to Resource Types *For Requests*:
com.adobe.cq.wcm.core.components.internal.models.v1.TitleImpl - core/wcm/components/title/v2/title
com.adobe.cq.wcm.core.components.internal.models.v1.BreadcrumbImpl - core/wcm/components/breadcrumb/v1/breadcrumb
com.adobe.cq.wcm.core.components.internal.models.v1.form.ContainerImpl - core/wcm/components/form/container/v2/container
com.adobe.cq.wcm.core.components.internal.models.v1.form.TextImpl - core/wcm/components/form/text/v2/text
com.adobe.cq.wcm.core.components.internal.models.v1.TextImpl - core/wcm/components/text/v2/text
com.adobe.cq.wcm.core.components.internal.models.v1.form.ContainerImpl - core/wcm/components/form/container/v1/container
com.adobe.cq.wcm.core.components.internal.models.v1.form.OptionsImpl - core/wcm/components/form/options/v2/options
com.adobe.cq.wcm.core.components.internal.models.v1.LanguageNavigationImpl - core/wcm/components/languagenavigation/v1/languagenavigation
com.adobe.cq.wcm.core.components.internal.models.v1.SocialMediaHelperImpl - core/wcm/components/sharing/v1/sharing
com.adobe.cq.wcm.core.components.internal.models.v2.PageImpl - core/wcm/components/page/v2/page
com.adobe.cq.wcm.core.components.internal.models.v1.BreadcrumbImpl - core/wcm/components/breadcrumb/v2/breadcrumb
com.adobe.cq.wcm.core.components.internal.models.v1.ImageImpl - core/wcm/components/image/v1/image
com.adobe.cq.wcm.core.components.extension.contentfragment.internal.models.v1.ContentFragmentImpl - core/wcm/extension/components/contentfragment/v1/co
com.adobe.cq.wcm.core.components.internal.models.v1.form.HiddenImpl - core/wcm/components/form/hidden/v1/hidden
com.adobe.cq.wcm.core.components.internal.models.v1.form.OptionsImpl - core/wcm/components/form/options/v1/options
com.adobe.cq.wcm.core.components.internal.models.v1.NavigationImpl - core/wcm/components/navigation/v1/navigation
com.day.cq.wcm.foundation.model.impl.PageImpl - wcm/foundation/components/page
com.adobe.cq.wcm.core.components.internal.models.v1.form.HiddenImpl - core/wcm/components/form/hidden/v2/hidden
com.adobe.cq.wcm.core.components.internal.models.v1.form.TextImpl - core/wcm/components/form/text/v1/text
com.adobe.cq.wcm.core.components.internal.models.v2.ImageImpl - core/wcm/components/image/v2/image
com.adobe.cq.wcm.core.components.internal.models.v2.ListImpl - core/wcm/components/list/v2/list
com.adobe.cq.wcm.core.components.internal.models.v1.PageImpl - core/wcm/components/page/v1/page
com.adobe.cq.wcm.core.components.internal.models.v1.SearchImpl - core/wcm/components/search/v1/search
com.adobe.cq.wcm.core.components.internal.models.v1.TextImpl - core/wcm/components/text/v1/text
we.retail.core.model.ProductGrid - weretail/components/content/productgrid
com.adobe.granite.contexthub.impl.models.ContextHubConfigFieldImpl - granite/components/configuration/form/inheritedconfig
com.day.cq.wcm.foundation.model.ResponsiveGrid - wcm/foundation/components/responsivegrid
com.adobe.cq.wcm.core.components.internal.models.v1.ListImpl - core/wcm/components/list/v1/list
com.adobe.cq.wcm.core.components.internal.models.v1.form.ButtonImpl - core/wcm/components/form/button/v1/button
com.adobe.cq.wcm.core.components.internal.models.v1.form.ButtonImpl - core/wcm/components/form/button/v2/button
com.adobe.cq.wcm.core.components.internal.models.v1.TitleImpl - core/wcm/components/title/v1/title

Sling Models Exporter Servlets:
com.adobe.cq.wcm.core.components.extension.contentfragment.internal.models.v1.ContentFragmentImpl exports 'core/wcm/extension/components/contentfragment'
com.adobe.cq.wcm.core.components.internal.models.v1.BreadcrumbImpl exports 'core/wcm/components/breadcrumb/v1/breadcrumb' with selector 'model' and ext
com.adobe.cq.wcm.core.components.internal.models.v1.form.ButtonImpl exports 'core/wcm/components/form/button/v2/button' with selector 'model' and exten
com.adobe.cq.wcm.core.components.internal.models.v1.LanguageNavigationImpl exports 'core/wcm/components/languagenavigation/v1/languagenavigation' with
com.adobe.cq.wcm.core.components.internal.models.v2.ListImpl exports 'core/wcm/components/list/v2/list' with selector 'model' and extension '[Ljava.lang
com.adobe.cq.wcm.core.components.internal.models.v2.TextImpl exports 'core/wcm/components/text/v2/text' with selector 'model' and extension '[Ljava.lang
com.day.cq.wcm.foundation.model.impl.PageImpl exports 'wcm/foundation/components/page' with selector 'model' and extension '[Ljava.lang.String;@2106345
com.adobe.cq.wcm.core.components.internal.models.v1.NavigationImpl exports 'core/wcm/components/navigation/v1/navigation' with selector 'model' and ext
com.adobe.cq.wcm.core.components.internal.models.v1.BreadcrumbImpl exports 'core/wcm/components/breadcrumb/v2/breadcrumb' with selector 'model' and ext
com.adobe.cq.wcm.core.components.internal.models.v1.form.ButtonImpl exports 'core/wcm/components/form/button/v1/button' with selector 'model' and exten
com.adobe.cq.wcm.core.components.internal.models.v1.SearchImpl exports 'core/wcm/components/search/v1/search' with selector 'model' and extension '[Lia
```

4. Navigate to the **Exercise_Files** folder on your system, open the **Stockplex.java** file, and notice the Java class is tied directly to the stockplex component by the **@Model** annotation's property, `resource=training/components/content/stockplex`. The **@Export** annotation allows you to use the Jackson exporter to export the content as JSON.

```

01.  @Model(adaptables=SlingHttpServletRequest.class,
02.          adapters= {ComponentExporter.class},
03.          defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL,
04.          resourceType = {"training/components/content/stockplex"})
05.  @Exporter(name="jackson", extensions = "json")
06.  public class Stockplex implements ComponentExporter{
07.
08.      @ScriptVariable
09.      private Resource resource;
10.
11.      @ValueMapValue
12.      private String symbol;
13.
14.      @ValueMapValue
15.      private String summary;
16.
17.      @ValueMapValue
18.      private String showStockDetails;
19.
20.      //content root of for stock data
21.      @ResourcePath(path = "/content/stocks")
22.      private Resource root;
23.
24.      @ScriptVariable
25.      private Style currentStyle;
26.
27.      //Get the stock model for the stock
28.      private StockModel getStock() {
29.          if(root == null) return null;
30.          //check to see if the stock data was imported and exists
31.          Resource stockResource = root.getChild(symbol);
32.          if(stockResource == null) return null;
33.
34.          return stockResource.adaptTo(StockModel.class);
35.      }

```

5. If you do not see the Stockplex model, you need to install the **we.train.core-0.0.1-SNAPSHOT.jar** into your AEM instance.

To use the Stockplex model, you need to update the stockplex.html script:

6. Ensure you are in CRXDE Lite or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
7. Navigate to the **/apps/training/components/content** node.
8. Select the **stockplex** node and double-click **stockplex.html** to open the script for editing.
9. Update the **stockplex.html** script by replacing the existing code with new code (**slingmodel_stockplex.html**) from the **Exercise_Files** folder.
10. Click **Save All** from the actions bar.
11. Open the **stockplex.html** to view the code that you added. Notice that the `data-sly-use="com.adobe.ats.core.models.Stockplex"` and notice how all values for the content are coming from the model rather than the dummy data. In addition, notice how the button is now been updated with an `href="${resource.path @ selectors='model', extension='json'}"`. When you request a resource with `model.json`, if there is a sling model for that resource, it will display the JSON output.

```

01. <sly data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html"
02. 2   data-sly-call="${clientlib.css @ categories='we.train.stockplex'}" />
03.
04.
05. 4 <div class="cmp-stockplex" data-sly-test.symbol="${properties.symbol}">
06. 5   <sly data-sly-use.stockplex="com.adobe.ats.core.models.Stockplex" />
07.
08. 7     <div class="cmp-stockplex__column1">
09. 8       <div class="cmp-stockplex__symbol">${stockplex.symbol}</div>
10. 9       <div class="cmp-stockplex__currentPrice">Current Value: ${stockplex.currentPrice}</div>
11.
12.
13. 12   <div class="cmp-stockplex__summary" data-sly-test.summary="${stockplex.summary}">
14. 13     <h3>Summary: ${stockplex.summary}</h3>
15. 14   </div>
16.
17. 16   <div class="cmp-stockplex__button">
18. 17     <a href="${resource.path @ selectors='model', extension='json'}">
19. 18       <button>See this stock via JSON</button>
20. 19     </a>
21. 20   </div>
22. 21 </div>
23. 22 <div class="cmp-stockplex__column2">
24. 23   <div class="cmp-stockplex__details" data-sly-test="${stockplex.showStockDetails}">
25. 24     <ul data-sly-list.dataKey="${stockplex.data}">
26. 25       <li class="cmp-stockplex__details-item">
27. 26         <span class="cmp-stockplex__details-title">${dataKey}:</span>
28. 27         <br />
29. 28         <span class="cmp-stockplex__details-data">${stockplex.data[dataKey]}</span>
30. 29       </li>
31. 30     </ul>
32. 31   </div>
33. 32 </div>
34. 33 </div>
35.
36. 35 <!-- If there is no stock symbol added to the dialog, create a component placeholder -->
37. 36 <div data-sly-test="${!symbol}" class="cq-placeholder" data-emptytext="Stockplex Component"></div>
```

To observe the JSON output:

12. Navigate to the tab where the English page is open or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The **English** page of We.Train site opens.
13. Refresh the page. Notice how you no longer see the "dummy" data, and the button's text is updated to **See this stock via JSON** as shown:

The screenshot shows the 'English' page with the Stockplex component. The component has two sections: 'Image' and 'List'. Below the component, the text 'ADBE' is displayed in large red letters, followed by 'Current Value: No Data' in red text. A red button labeled 'See this stock via JSON' is present. The page toolbar at the top includes icons for back, forward, and search, along with 'ENGLISH', 'Edit', and 'Preview' buttons.

To view the output:

14. Click **Preview** from the page toolbar and click the **See this stock via JSON** button on the page. The page with the following output opens in the same tab as shown:

The screenshot shows the JSON output in a code editor-like interface. The code is as follows:

```
{"symbol": "ADBE", "summary": "Adobe's Stock price today", "data": {}, "currentPrice": "No Data", "type": "training/components/content/stockplex"}
```

Optional Exercise

Add live data to the Stockplex Component

To view the actual stock information in the Stockplex component, you need to add a datasource to the component. This data source is built in Java using a Polling importer. To learn how to build the back-end code for the component, you can register for "Extend and Customize Adobe Experience Manager" course, which teaches you how to code polling importers, Sling models, and service users in detail.

Exercise 6: Add a selector to the component

Let us look back at Sling URL decomposition, resource resolution, and how Sling finds the rendering script. In this exercise, you will add print.html script to the page node, and observe how Sling will first attempt to find a script to matches the selector.

1. Ensure you are in CRXDE Lite or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components/structure** folder, select the **page** node, click **Create** from the actions bar, and then click **Create File** from the list. The **Create File** dialog box opens.
3. Enter **print.html** in the **Name** field and click **OK**. The file is created.
4. Click **Save All** from the actions bar.
5. Select the **jcr:content** node that is below **print.html**. The **Properties** tab opens.
6. Double-click the **jcr:data** property. The **Edit jcr:data** dialog box opens.
7. Click **Browse**. The **Open** dialog box opens.
8. Navigate to the **Exercise_Files** folder on your file system.
9. Select **print.html**, and then click **Open**. The **Edit jcr:data** dialog box opens.
10. Click **OK**. The default sample code is replaced with the code in **print.html**.
11. Click **Save All** from the actions bar.
12. Open the **print.html** to view the code that you added. Observe the following code snippet:

```
01. WE.<strong>TRAIN</strong>: ${currentPage.title} Page Print
02. <hr />
03.
04. <!--/* Include all content components for printing */-->
05. <div data-sly-resource="${'root/responsivegrid' @resourceType='wcm/foundation/components/responsivegrid'}"></div>
```

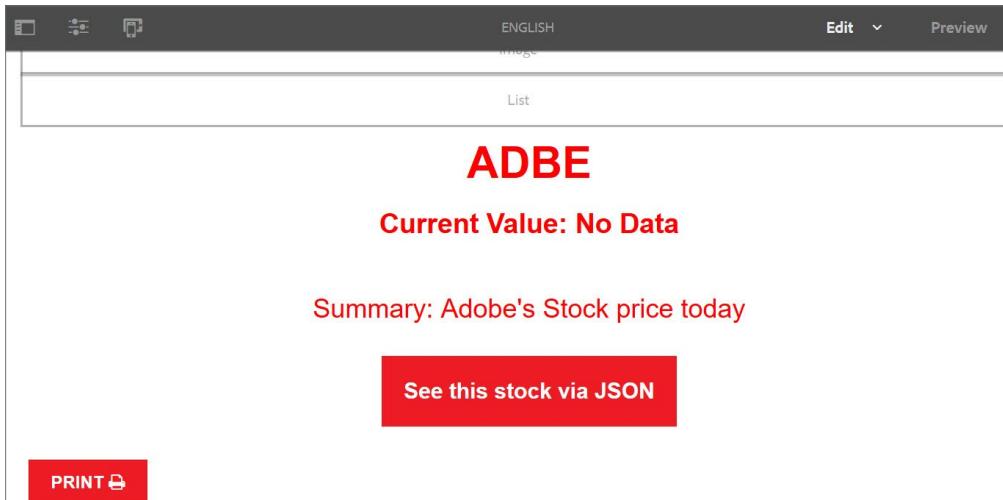
You need to update the footer component to call the print script with a selector.

13. Navigate to the `/apps/training/components/structure/footer/footer.html` file.
14. Select the `jcr:content` node that is below `footer.html`. The **Properties** tab opens.
15. Double-click the `jcr:data` property. The **Edit jcr:data** dialog box opens.
16. Click **Browse**. The **Open** dialog box opens.
17. Navigate to the **Exercise_Files** folder on your file system.
18. Select `print_footer.html`, and then click **Open**.
19. Click **OK** in the **Edit jcr:data** dialog box. The default sample code is replaced with the code in `print_footer.html`.
20. Click **Save All** from the actions bar.
21. Open the `footer.html` to view the code that you added. Observe the following code snippet:

```
01. <!--/* Inserts a print selector into the request so that the page/print.html script can be run. */-->
02. <a href="${currentPage.Path @ selectors='print', extension='html'}" class="btn btn-primary btn-print">
03.   ${ 'Print' @ i18n } <i class="fa fa-print" aria-hidden="true"></i>
04. </a>
05.
06. <sly data-sly-use.footer="footer.js">
07.   <footer class="we-Footer">
08.     <div class="container">
09.
10.       <div class="row">
11.
12.         <div class="col-md-4">
13.           <div class="we-Logo we-Logo--big">
14.             we.<strong>Train</strong>
15.           </div>
16.         </div>
17.
18.         <div class="row col-md-8 col-xs-12 we-footer-links">
19.           <sly data-sly-list.item="${footer.items}">
20.             <div class="col-lg-2 col-md-2 col-xs-3">
21.               <div class="we-Footer-nav">
22.                 <h2 class="h4">
23.                   <a href="${ item.path @ extension = 'html'}">${item.title || item.name}</a>
24.                 </h2>
25.               </div>
26.             </div>
27.           </sly>
28.         </div>
```

To observe the changes:

22. Navigate to the tab where the English page is open or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The English page of We.Train site opens.
23. Notice how the **Print** button is added to the bottom of the page before the footer component as shown:



24. Click **Preview** from the page toolbar and click the **Print** button. The page opens in the same tab with the following information as shown:



Exercise 7: Create the localization information

In this exercise, you will create localized content. In addition, you will look at HTL code that specifies the internationalization process.

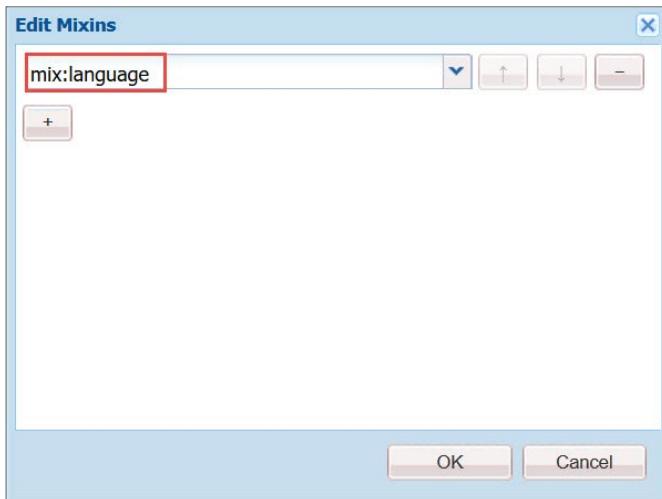
1. Ensure you are in CRXDE Lite or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components/structure/footer** node.
3. Double-click the **footer.html** to open the script for editing. Notice the following code snippet and observe the i18n content:

```
01. <div class="row we-Footer-section we-Footer-section--sub">
02.     <div class="we-Footer-section-item">
03.         <span class="text-uppercase text-muted">${ '© {0} All rights reserved' @ format=
04.             [footer.currentYear], i18n }</span>
05.     </div>
06.     <div class="we-Footer-section-item">
07.         <a href="#" class="text-uppercase text-muted">${ 'Terms of use & privacy policy' @ i18n }</a>
08.     </div>
09.
10.    <div class="row">
11.        <div class="col-md-12">
12.            <div class="text-center">
13.                <a href="#top" class="btn btn-primary">
14.                    ${ 'Ride to the top' @ i18n }
15.                    <i class="fa fa-arrow-up" aria-hidden="true"></i>
16.
17.                </a>
18.            </div>
19.        </div>
    </div>
```

Let us internationalize the copyright statement on the footer of a website.

4. Navigate to the **/apps/training/components/structure** folder.
5. Select the **footer** node, click **Create** from the actions bar, and click **Create Folder** from the list. The **Create Folder** dialog box opens.
6. Enter **i18n** in the **Name** field and click **OK**. The folder is created.
7. Click **Save All** from the actions bar.

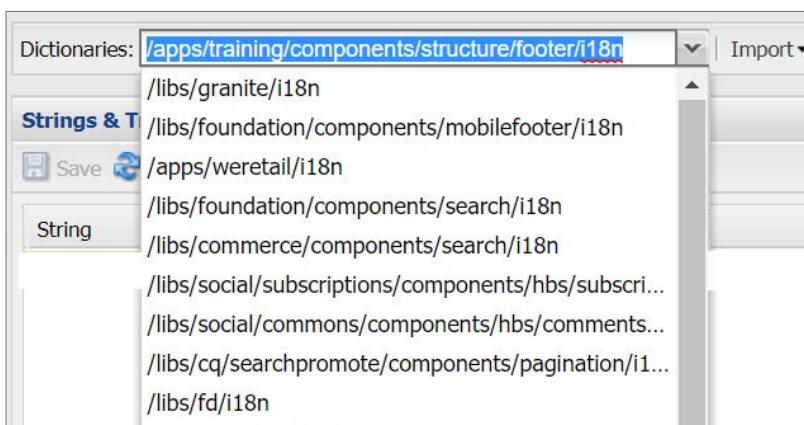
8. Select the **i18n** folder and create a child folder named **fr**.
9. Click **Save All** from the actions bar.
10. Select the **fr** folder and click **Mixins** from the actions bar. The **Edit Mixins** dialog box opens.
11. Click **+** icon in the **Edit Mixins** dialog box.
12. Select **mix:language** from the drop-down list as shown:



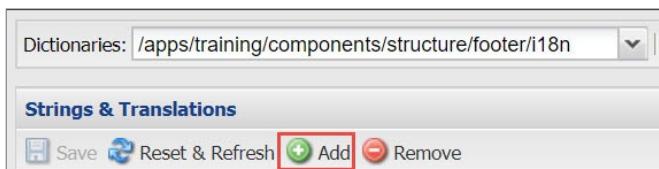
13. Click **OK**, and then click **Save All** from the actions bar.
14. Select the **fr** folder and add the following property:

Name	Type	Value
jcr:language	String	fr

15. Add <http://localhost:4502/libs/cq/i18n/translator.html> URL in the address bar of the browser and press Enter. The **Strings & Translations** page opens.
16. Select **/apps/training/components/structure/footer/i18n** from the **Dictionaries** drop-down menu as shown:



17. Click **Add** from the actions bar as shown. The **Add String** dialog box opens.

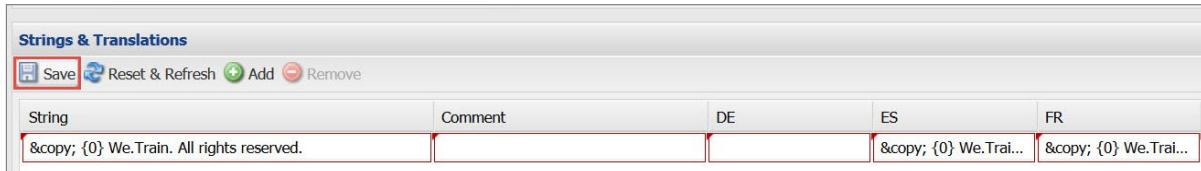


18. Enter the following information in the appropriate fields:

Field Name	Value
String	© {0} We.Train. All rights reserved.
ES	© {0} We.Train. Todos los derechos reservados.
FR	© {0} We.Train. Tous droits réservés.

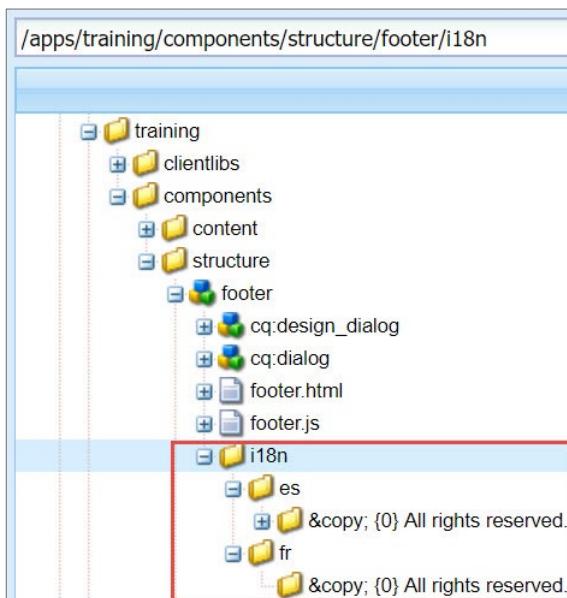
19. Click **OK**.

20. Click **Save** from the actions bar as shown:



21. Navigate to the tab where the **CRXDE Lite** page is open or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.

22. Navigate to the **/apps/training/components/structure/footer/i18n** folder. Notice, a new folder **es** along with child node was created and the **fr** folder has a child node as shown:

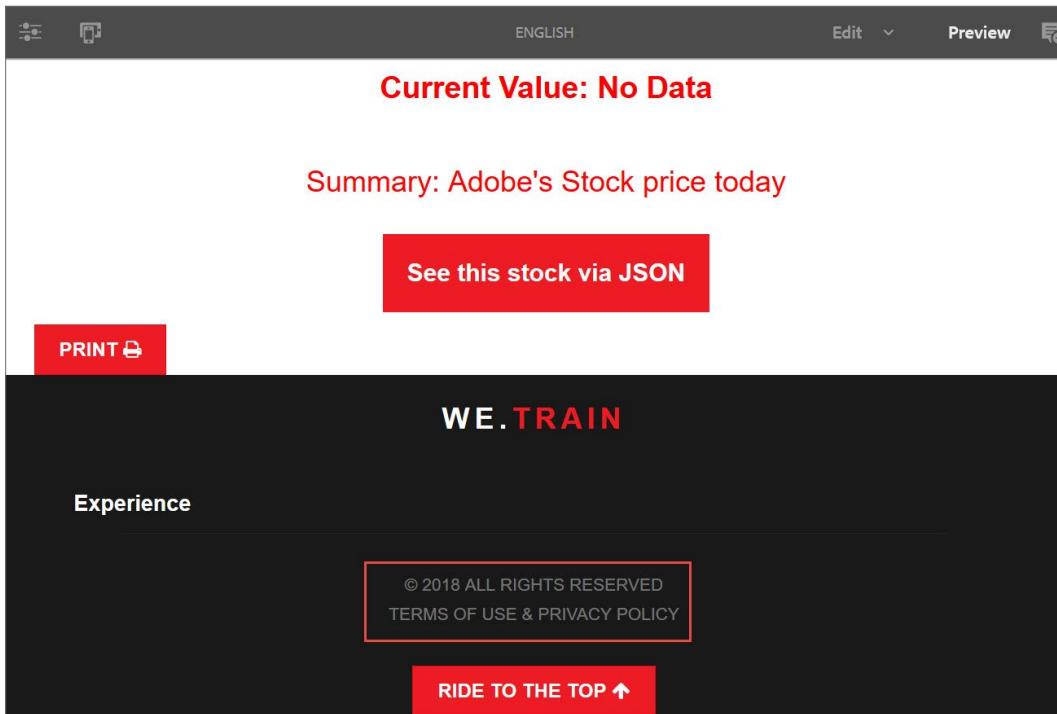


23. Examine the properties of the **es** and **fr** child nodes.

Exercise 8: Test the localized content

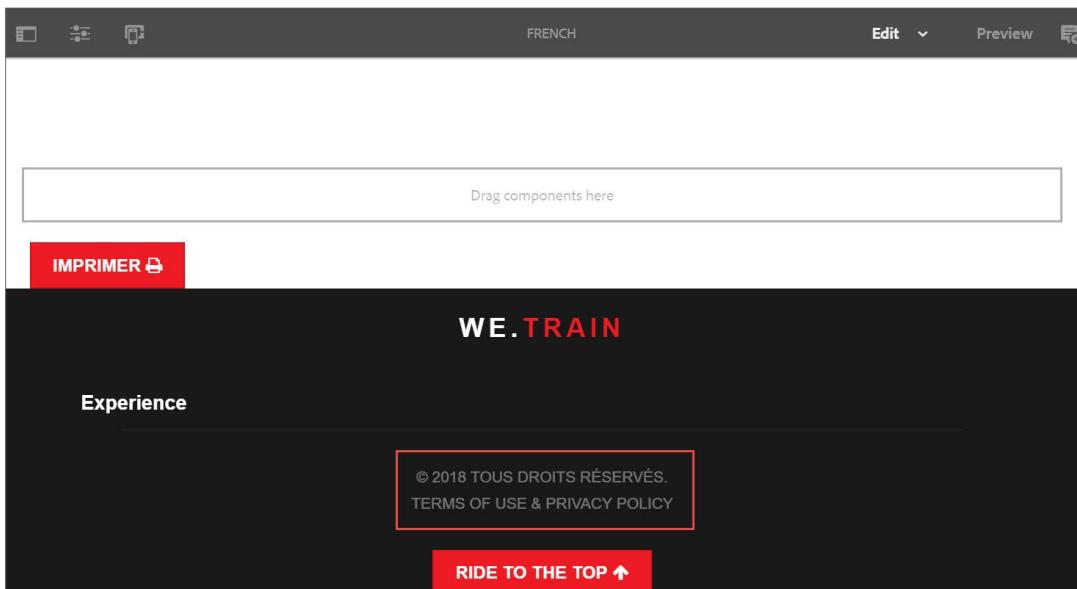
To test the localized content:

1. Navigate to the tab where the **English** page is open or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The **English** page of the We.Train site opens. Notice, the footer of the page as shown:



2. Navigate to the tab where the **AEM Sites** tab is open or open a new tab in your browser, add <http://localhost:4502/sites.html/content> in the address bar, and then press Enter. The **Sites** console opens.
3. Navigate to the **We.Train** site, click the thumbnail icon beside the **French** page, and then click **Edit** from the actions bar. The page opens in a new tab.

4. Scroll down and notice the footer. The copyright message is translated to French as shown:



Optional Exercise

Create a new supported language through JSON

Create a JSON file to translate the footer content of the English page to Italian language. Also, create JSON files for es and fr (use the json files from the Exercise_Files folder). Under the i18n folder, create a new file called it.json (es.json and fr.json) and add the contents from the Exercise_Files folder. Once done, go and create an Italian (it) language branch and see if the content is translated.

Optional Exercise

Manage supported languages

AEM supports nine supported languages in the translator tool. You can customize the translator tool to show more/fewer languages depending on the requirement(s). To customize this console, you can create a new node, /etc/languages-, and add a multi-value property called **languages**. You can use this to set which languages are supported in the translator tool.

You should try and customize the translator tool to only support Spanish (es) and French (fr).

If you need help with the customization, refer this guide: <https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/i18n-translator.html#ManagingSupportedLanguages>

Sling Resource Merger

The AEM UI is based on a set of nodes in JCR, and its underlying structure. These nodes reside in the /libs folder. When you create a project, any customization required is done by overlaying the corresponding content node in the JCR under the /apps folder. This is done by overlaying the content node in the JCR.

The two methods used for customizations are:

- Overlays
- Sling Resource Merger

Overlays

Sling's Resource Resolver will search for resources in a list of specific paths, as defined by the Apache Sling JCR Resource Resolver. The default search path is first /apps and then /libs. As a result, you can change the out-of-the-box functionality as provided in /libs by adding a resource or file at the same path in /apps. This ability to override default functionality is called an overlay.

You must follow these steps to use overlays in AEM:

1. Recreate the node structures from /libs under /apps.
2. Copy the content node that you want to change in /apps, and then work on this copy.
3. When a resource is requested, it is resolved according to the search path logic.
4. It retrieves the resource from /apps, failing which, it looks at /libs.

An overlay involves taking the predefined functionality and imposing your own definitions over that to override the standard functionality.

Sling Resource Merger

Sling Resource Merger is applicable only to the touch UI, and enables you to customize consoles and page authoring. With Sling Resource Merger, you can overlay nodes without replicating all of their ancestors. It uses difference mechanisms along with resource resolver search paths to merge overlays of resources.

The Sling Resource Merger provides services to access and merge resources. It merges the overlays of resources by using resource resolver search paths and difference mechanisms. A customized Sling vocabulary is used to manage overrides of nodes and their properties. With this, the overlay resources/properties (defined in /apps) are merged with the original resources/properties (from /libs).

The Sling Resource Merger is used to ensure that all changes are made in /apps, and avoids the need to make any changes in /libs. After the structure is created, you can add your own properties to the nodes under /apps. The content of /apps has a higher priority than that of /libs. The properties defined in /apps indicate how the content merged from /libs are to be used.

The following table describes difference between using Overlays and Sling Resource Merger:

Overlays	Sling Resource Merger
Works based on search paths (/libs + /apps)	Works based on Resource Type Hierarchy
Need to copy the whole subtree	Extends within an almost empty subtree
All the properties are duplicated	Only required properties are overlaid
When upgrades are done to the /libs folder, these changes have to be manually recreated under /apps.	As properties are not copied, only the structure, upgrades are automatically reflected in /apps.

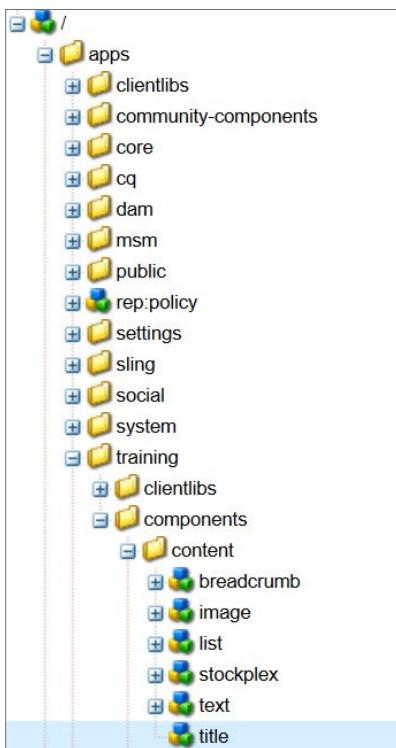
After you copy a system file from /libs to /apps to overlay it with custom code, your custom code will not pick up any modifications to the system component/file/script/dialog box, which result from the application of a hotfix, feature pack, or upgrade. A careful examination of the release notes of any upgrade, feature pack, or hotfix should be a step in your upgrade plan. This way, you will be able to evaluate any changes and for incorporating them into your application.

Exercise 9: Extend a core component

In this exercise, you will extend a core component without losing the functionality that it provides and ensuring it works for future core component upgrades.

To extend the Title component with the Sling Resource Merger:

1. Navigate to the tab where the **CRXDE Lite** page is open or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components/content** folder. Notice a **title** proxy component is available as shown:



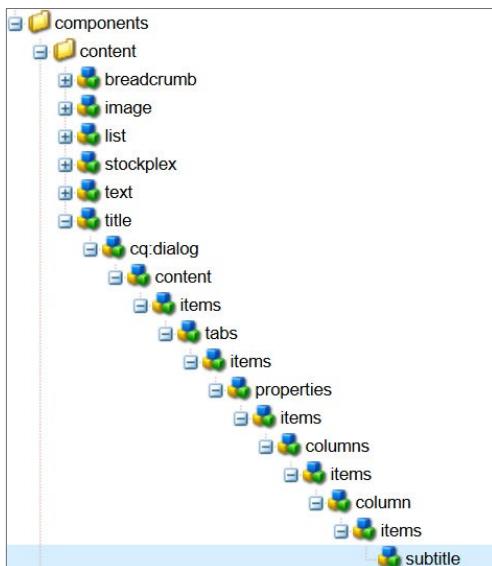
3. Navigate to the **/apps/core/wcm/components/title/v2/title** node, select the **cq:dialog** node, and click **Copy** from the actions bar.
4. Navigate to the **/apps/training/components/content/title** node, and click **Paste** from the actions bar. The **cq:dialog** node is added to the title proxy component.
5. Click **Save All** from the actions bar.

6. Navigate to the `/apps/training/components/content/title/cq:dialog/content/items/tabs/items/properties/items/columns/items/column/items` node, select the **types** node, and click **Delete** from the actions bar.
7. Click **Save All** from the actions bar.
8. Similarly, select the **defaulttypes** node that is below the **items** node, and click **Delete** from the actions bar.
9. Click **Save All** from the actions bar.



Note: You need to delete the formfields that you want to inherit from the v2 title component. As a result, the custom formfields you create next will be merged with the v2 title dialog.

10. Select the **title** node, click **Rename** from the actions bar, and rename it to **subtitle** as shown:

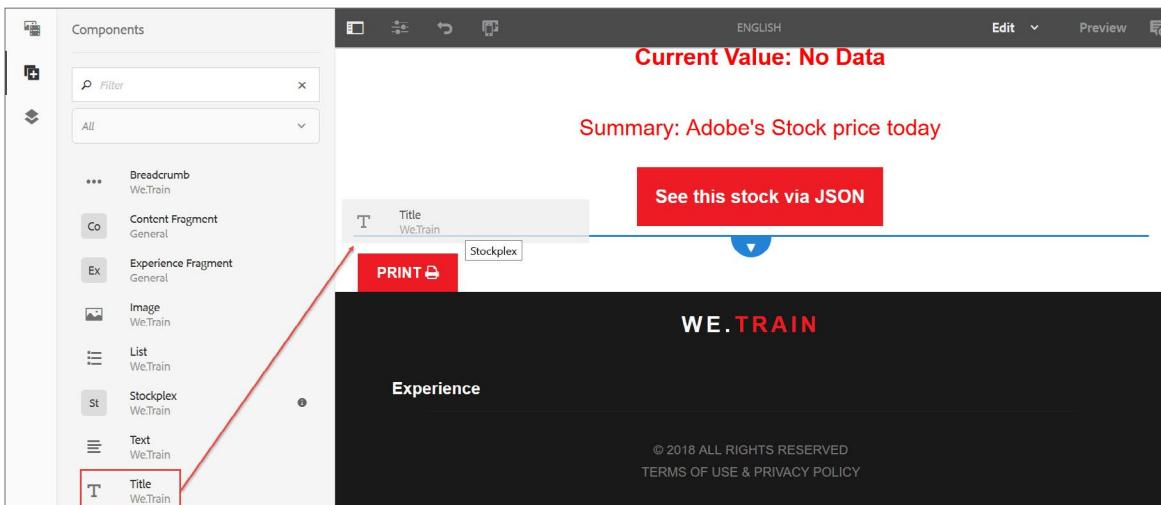


11. Update the following properties of the **subtitle** node:

Name	Type	Value
fieldLabel	String	Subtitle
fieldDescription	String	Enter your custom subtitle
name	String	./subtitle

12. Click **Save All** from the actions bar.
13. Navigate to the tab where the **English** page is open or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The **English** page of We.Train site opens.
14. Ensure the page is open in edit mode or click **Edit** from the page toolbar. The page is now editable.

15. Click the Toggle Side Panel icon from the page toolbar. The panel opens.
16. Click the Components icon. The Component browser opens.
17. Drag the **Title** component from the browser to below the **Stockplex** component as shown. The **Title** component is added to the page.



18. Select the **Title** component and click the Configure (wrench) icon from the component toolbar as shown. The **Title** dialog box opens.



19. Notice the new field **Subtitle** is added to the dialog box.
20. Enter the following:
 - a. **Title: My new title**
 - b. **Type / Size: H1**
 - c. **Subtitle: My custom subtitle**

21. Click the Done (checkmark) icon in the upper right, as shown. You will be taken back to the **English** page.



Notice how at this point you do not see the subtitle on the page. You need to create a local title.html script to reference the new subtitle property.

22. Navigate to the tab where the **CRXDE Lite** page is open or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
23. Navigate to the **/apps/training/components/content** folder, select the **title** node, click **Create** from the actions bar, and then click **Create File** from the list. The **Create File** dialog box opens.
24. Enter **title.html** in the **Name** field and click **OK**. The file is created below the **title** node.
25. Click **Save All** from the actions bar.
26. Select the **jcr:content** node that is below **title.html**. The **Properties** tab opens.
27. Double-click the **jcr:data** property. The **Edit jcr:data** dialog box opens.
28. Click **Browse**. The **Open** dialog box opens.
29. Navigate to the **Exercise_Files** folder on your file system.
30. Select **title.html**, and then click **Open**. The **Edit jcr:data** dialog box opens.
31. Click **OK**. The default sample code is replaced with the code in **title.html**.
32. Click **Save All** from the actions bar.

33. Open the **title.html** to view the code that you added. Notice the script is the exact same script as the core title component, except there is a new line of code that adds the subtitle to the component:

```

01. <h1 data-sly-use.title="com.adobe.cq.wcm.core.components.models.Title"
02.     data-sly-use.template="core/wcm/components/commons/v1/templates.html"
03.     data-sly-element="${title.type}"
04.     data-sly-test.text="${title.text}">${title.text}</h1>
05.
06.     <!--/* Subtitle added to HTL */-->
07.     <h2 data-sly-test="${properties.subtitle}">${properties.subtitle}</h2>
08.
09. <sly data-sly-call="${template.placeholder @ isEmpty=!text}"></sly>
```

34. Navigate to the tab where the **English** page is open and refresh the page.

35. If the page is not open, open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The **English** page of We.Train site opens.

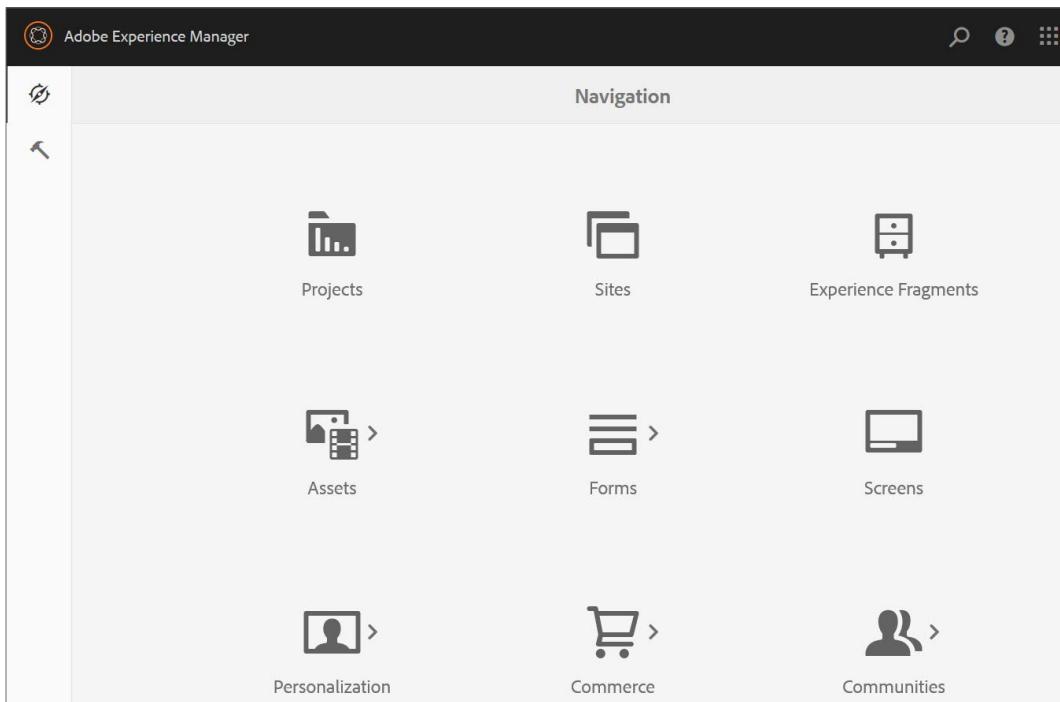
36. Observe the new subtitle on the pages as shown:

The screenshot shows the Adobe Experience Manager interface with the 'ENGLISH' page selected. The page content includes a red banner with the text 'Current Value: No Data', a summary statement 'Summary: Adobe's Stock price today', a red button labeled 'See this stock via JSON', and the main title 'My new title' followed by the subtitle 'My custom subtitle'. A placeholder area for dragging components is visible below the main content, and a 'PRINT' button is at the bottom left.

Exercise 10: Extend the Navigation UI

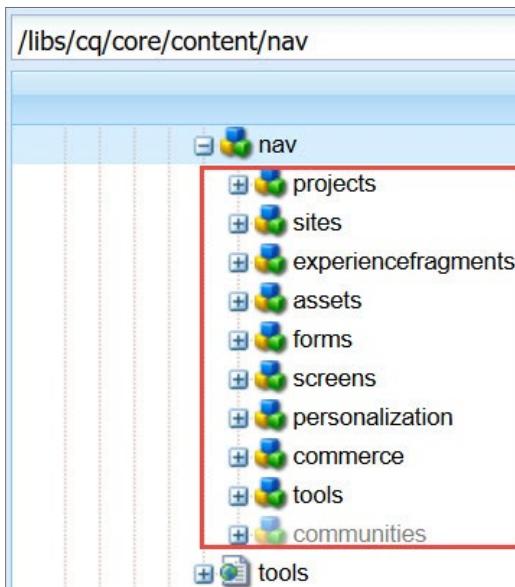
In this exercise, you will modify the navigation UI buttons by using the Sling Resource Merger.

1. Open a new tab of the browser, add <http://localhost:4502/aem/start.html> in the address bar, and press Enter. The **Navigation UI** opens as shown:



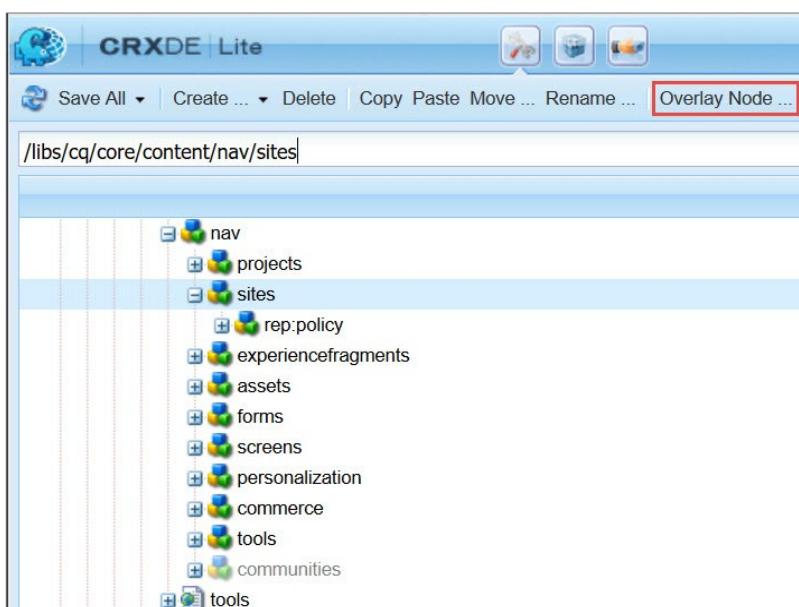
To modify the navigation button:

2. Navigate to the tab where the **CRXDE Lite** page is open or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
3. Navigate to the **/libs/cq/core/content/nav** node. Notice how the child nodes of nav are the definition of the global navigation icons (consoles) for AEM.

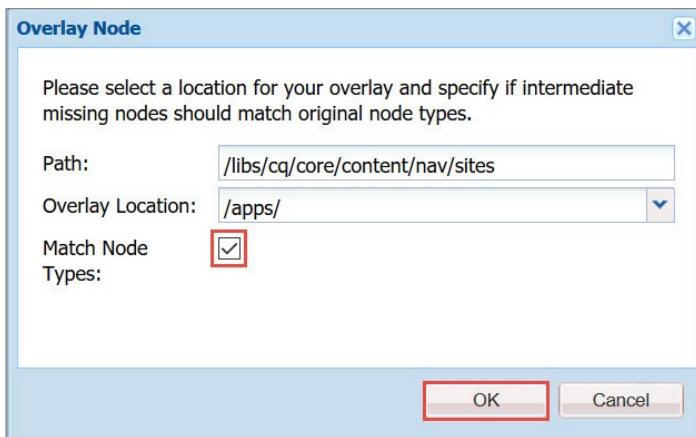


To modify the navigation icons, hide icons, or add a new icon, you need to change the list under **/libs/cq/core/content/nav**. It is a best practice to recreate the structure in **/apps** by using the Sling Resource Merger to avoid any changes to the **/libs** structure.

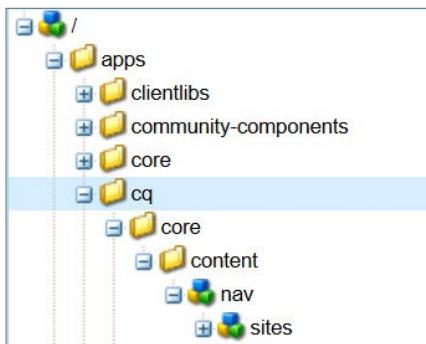
4. Navigate to the **/libs/cq/core/content/nav** node, select the **sites** node, and click **Overlay Node** from the actions bar, as shown. The **Overlay Node** dialog box opens.



5. Ensure the **Overlay Location** field has **/apps/** as its value, select the **Match Node Types** checkbox, and then click **OK** as shown:



6. Click **Save All** from the actions bar.
7. Navigate to the **/apps** folder, and notice, the **/cq/core/content/nav/sites** structure is created, as shown:



8. If you cannot view the **/apps/cq** folder structure, refresh the **CRXDE Lite** page.

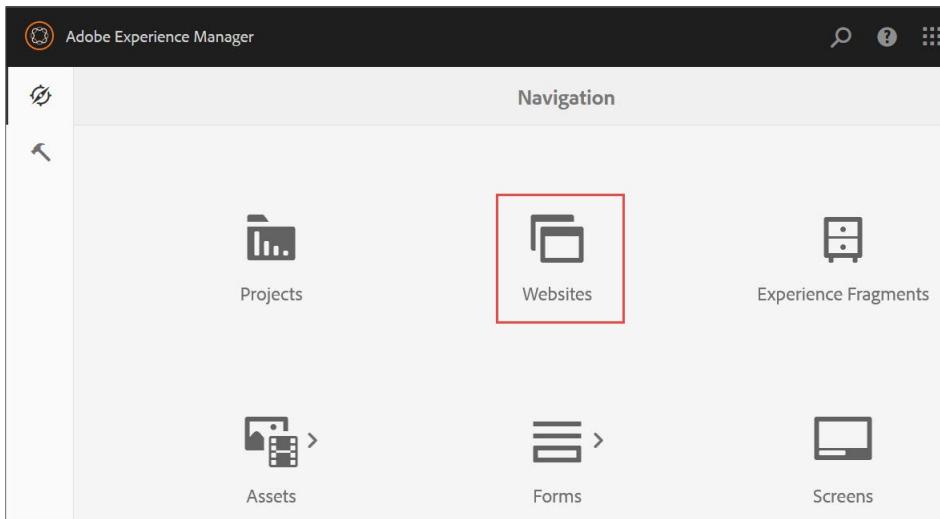


Note: The overlay node structure is reproduced, but the properties are not created. Remember, the Sling Resource Merger will merge properties, and nodes.

9. Add the following property to sites under **/apps/cq/core/content/nav/** node:

Name	Type	Value
jcr:title	String	Websites

10. Click **Save All** from the actions bar.
11. Open the tab where the **Navigation** page is open. Refresh the page and notice how the **Sites** button is now renamed to **Websites** as shown:



Optional Exercise

Hide a navigation button

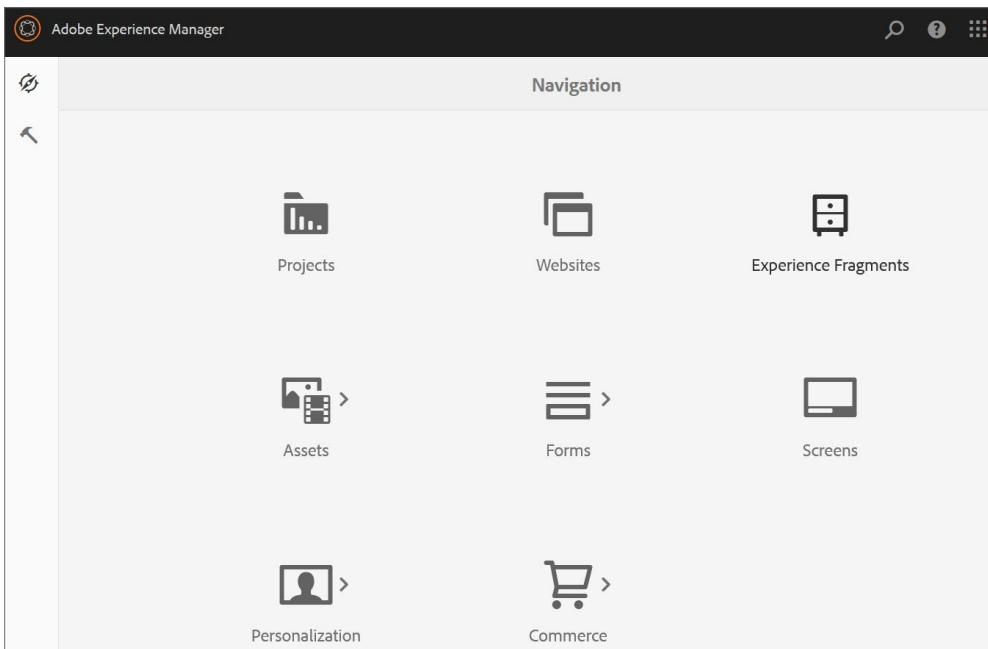
To hide a navigation button:

1. Navigate to the tab where the **CRXDE Lite** page is open or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/cq/core/content/nav** node, click **Create** from the actions bar, and click **Create Node** from the list. The **Create Node** dialog box opens.
3. Enter **communities** in the Name field, ensure the Type is **nt:unstructured**, and then click **OK**. The node is created.
4. Click **Save All** from the actions bar.
5. Add the following property to the **communities** node:

Name	Type	Value
sling:hideResource	Boolean	true

6. Click **Save All** from the actions bar.

7. Open the tab where the **Navigation** page is open. Refresh the page and notice, the **Communities** is hidden from **Navigation** as shown:



Optional Exercise

Add a new navigation button

To add a new navigation button:

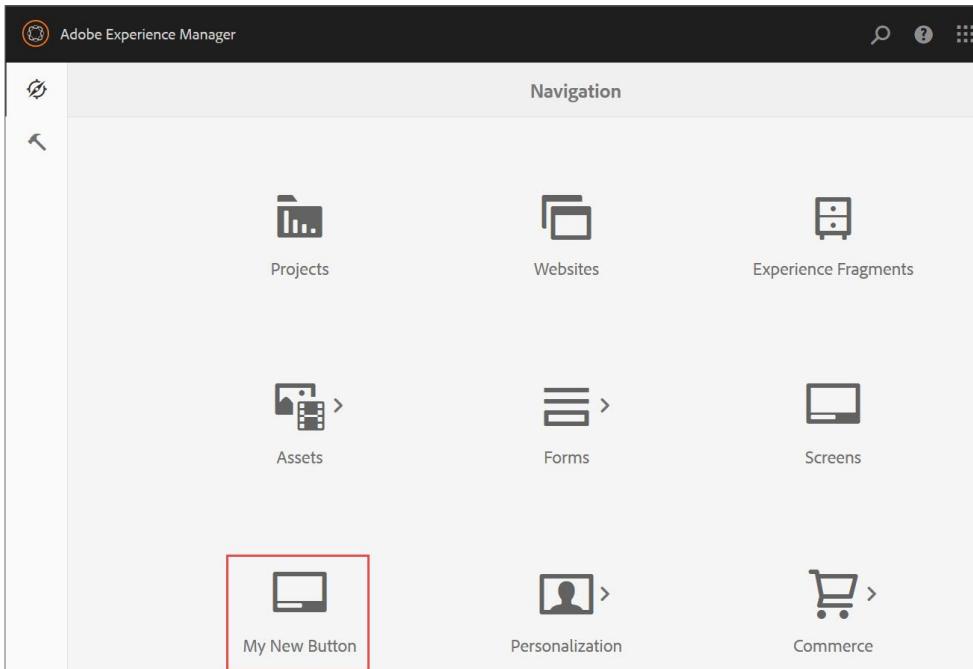
1. Navigate to the tab where the **CRXDE Lite** page is open or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Navigate to the `/libs/cq/core/content/nav` node, select the screens node, and click **Copy** from the actions bar.
3. Navigate to the `/apps/cq/core/content/nav` node and click **Paste** from the actions bar. The screens node is added.
4. Click **Save All** from the actions bar.
5. Select the **screens** node, double-click the node, and rename it to **mybutton**.
6. Click **Save All** from the actions bar.

7. Update the following properties of the **mybutton** node:

Name	Type	Value
jcr:title	String	My New Button
id	String	cq-mybutton
href	String	/sites.html

8. Click **Save All** from the actions bar.

9. Open the tab where the **Navigation** page is open. Refresh the page and notice, a new console **My New Button** is created, as shown:



10. Click **My New Button**. The **Sites** console opens.



Preparing for Production

Introduction

Before implementing Adobe Experience Manager (AEM) as the Content Management System (CMS), you need to prepare AEM for production and optimize the performance of AEM during deployment.

Objectives

After completing this course, you should be able to:

- Explain how to prepare AEM for production
- Complete the empty page template type
- Create production templates
- Create a code content package
- Explain the AEM environment

AEM in Production

A developer needs to move their code into production or Quality Assurance (QA). This involves finishing the template-type, creating any initial editable templates for production, finalizing/creating the skeleton website structure, packaging up the code, and moving the code through environments to production. For example, as a developer, this means after you have completed developing the components and added them to the template. You need to move the changes from the template to the template type, if you do not want the template authors to recreate all the configurations that you have set up (allowed components, clientlibs, and thumbnails).

Developers create the first few templates of the project. This helps create the initial site structure, proper documentation to be created on company customizations, and sample templates for template authors.

After the initial templates are in production, template authors will maintain them and also create new templates based on the business need. The developers set the standard, and template editors adhere to that standard.

You have created the code in CRXDE Lite, but realistically, it would be developed through an external editor in a Maven project. The Maven project will have a content package that contains the complete code, which goes into Java Content Repository (JCR). The content package helps move the code within different environments. You can move code from development to production by using the content package.

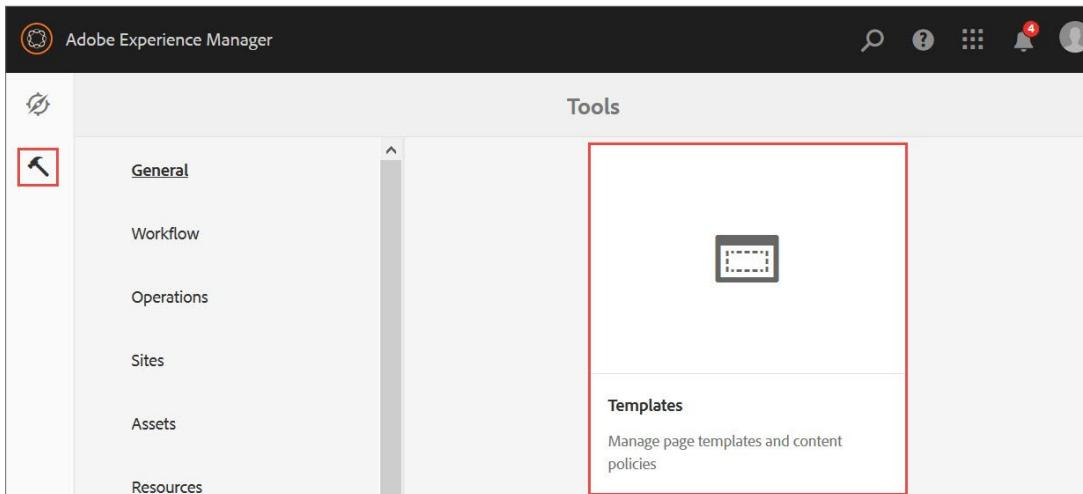
Exercise 1: Complete the empty page template type

Now that you have tested all structure components (assuming they have gone through a full QA process), you can complete the template-type.

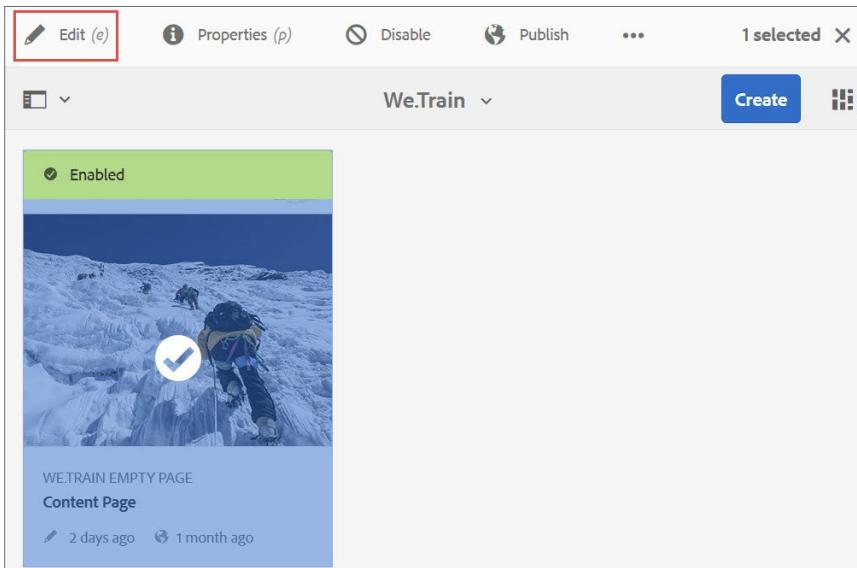
You need to install the **adls-training-project-v6.0.zip** package on your AEM author instance. The package contains the required training project structure, page component with clientlibs, template, and site structure that was created for you.

In this exercise, you will enable template authors to use the content components in the template along with the structure components.

1. Click **Adobe Experience Manager** from the header bar, and click the Tools icon.
2. Ensure you are in the **General** section, and click **Templates** as shown. The **Templates** console opens.



3. Navigate to the **We.Train** folder, select the **Content Page** template, and click **Edit** from the actions bar as shown. The **Content Page** template editor opens on a new tab.



4. Select the **Layout Container [Root]** and click the Policy icon from the component toolbar. The Layout Container wizard opens.
5. In the **Policy** section, notice the structure components you added previously to the policy.
6. In the **Properties** section, under **Allowed Components**, select the **We.Train** component group, and then click the Done (checkmark) icon.

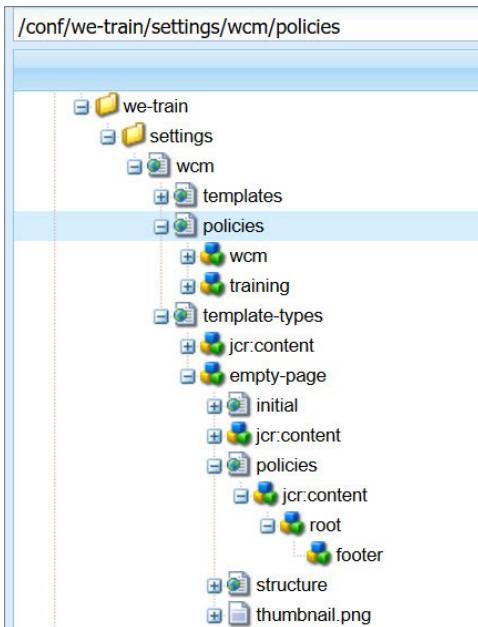
You need to copy the updates that you have made to the template back to the template-type.

7. Ensure you are in CRXDE Lite or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
8. Navigate to the **/conf/we-train/settings/wcm/templates/content-page/policies/jcr:content** node, select the **root** node, and click **Copy** from the actions bar. The root node you are copying has the policy for the root layout container. This ensures the approved structure components are available for a template author.
9. Navigate to the **/conf/we-train/settings/wcm/template-types/empty-page/policies/jcr:content** node, and click **Paste** from the actions bar. The node is copied.
10. Click **Save All** from the actions bar.



Note: You can also press Ctrl+S (Windows users) and Command+S (Mac users) to save your work in CRXDE Lite.

11. Navigate to the `/conf/we-train/settings/wcm/template-types/empty-page/policies/jcr:content/root` node, select the **responsivegrid** node, and click **Delete** from the actions bar. This will help template authors determine the components to be used on a page. Your folder structure should look similar to the below screenshot:



If you need to auto-add any other configurations or components to a new template, you can copy those nodes/properties over to the template-type.

Exercise 2: Create production templates

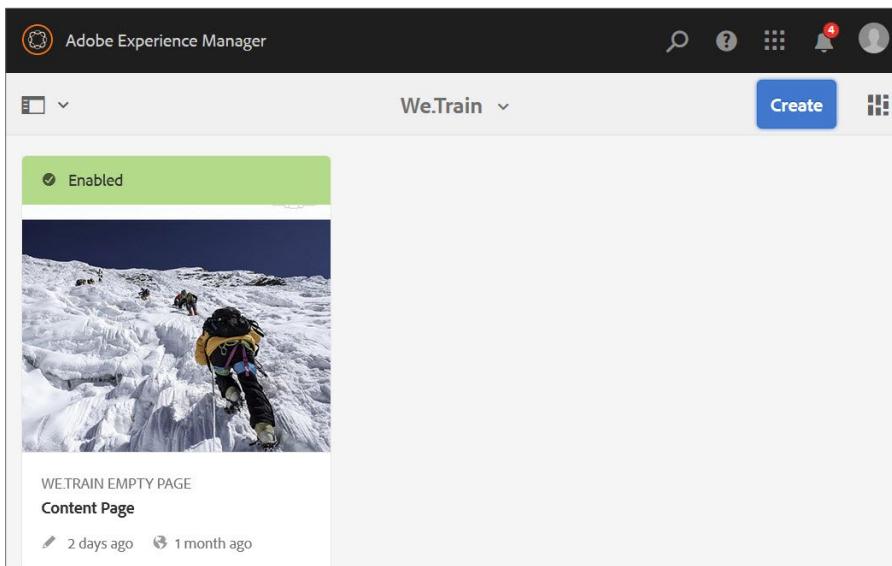
You have completed the development of a template-type and one template that can be used by template authors. In this exercise, you will create two more responsive templates that will be immediately available for content authors. In production, template authors can continue creating new templates or update the three templates created by the development team.

You will create the following two new templates:

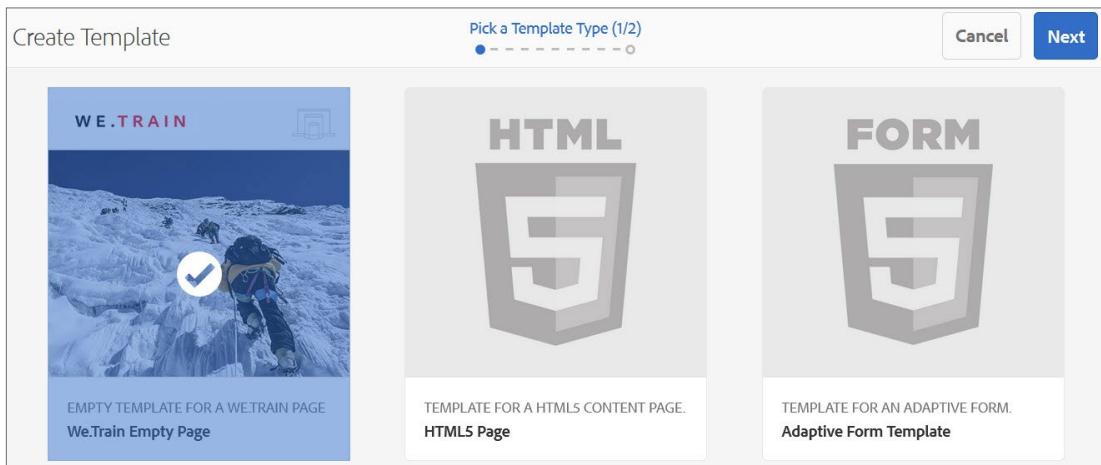
- Sidebar Page Template
- Stockplex Template

To create a Sidebar template:

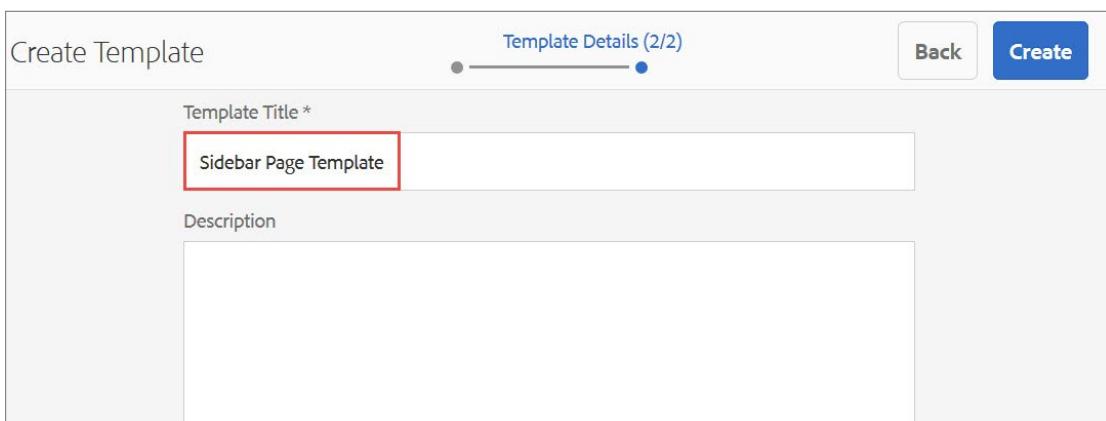
1. Click **Adobe Experience Manager** from the header bar, and click the Tools icon.
2. Ensure you are in **General** section, and click **Templates**. The **Templates** console opens.
3. Navigate to the **We.Train** folder and click **Create** from the actions bar as shown. The **Create Template** wizard opens.



4. Select the **We.Train Empty Page** template and click **Next** as shown. The **Template Details(2/2)** wizard opens.

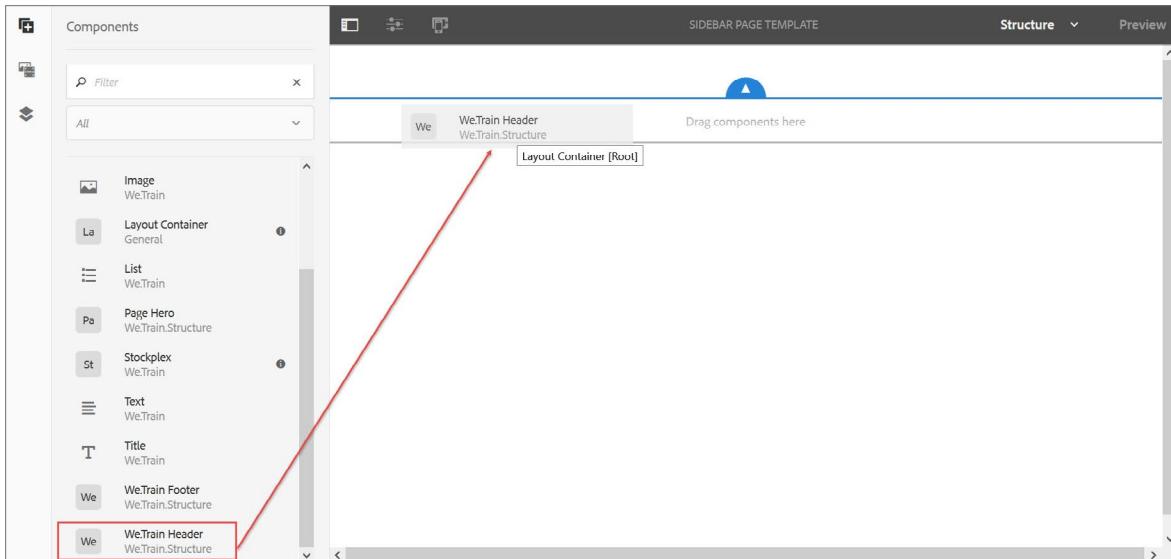


5. Enter **Sidebar Page Template** in the **Template Title*** field, and then click **Create** as shown. The **Success** dialog box opens.

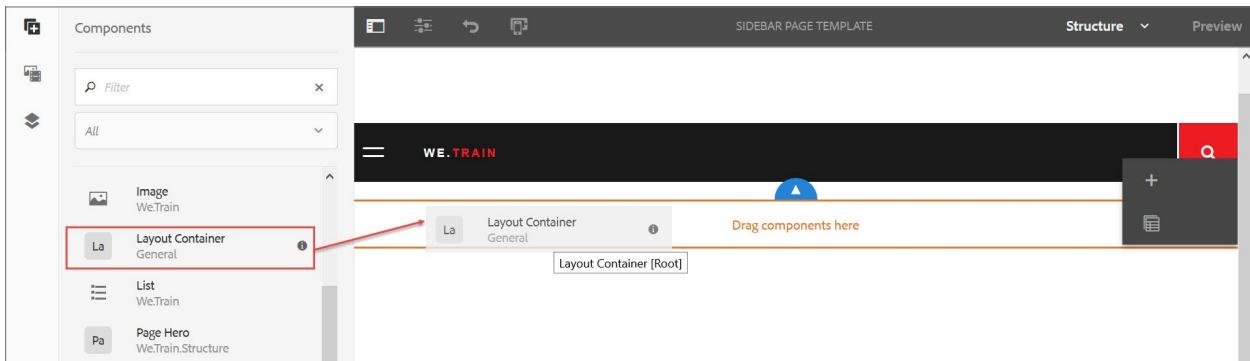


6. Click **Open**. The **Sidebar Page Template** opens on a new tab.
7. Ensure the template is open in the **Structure** mode.

8. Click the Toggle Side Panel icon from the toolbar. The panel opens.
9. Click the Components icon. The Components browser opens.
10. Ensure the All option is selected in the drop-down menu in the Components browser.
11. Drag the **We.Train Header** component from the components browser to the **Drag components here** area placeholder as shown. The component is added.

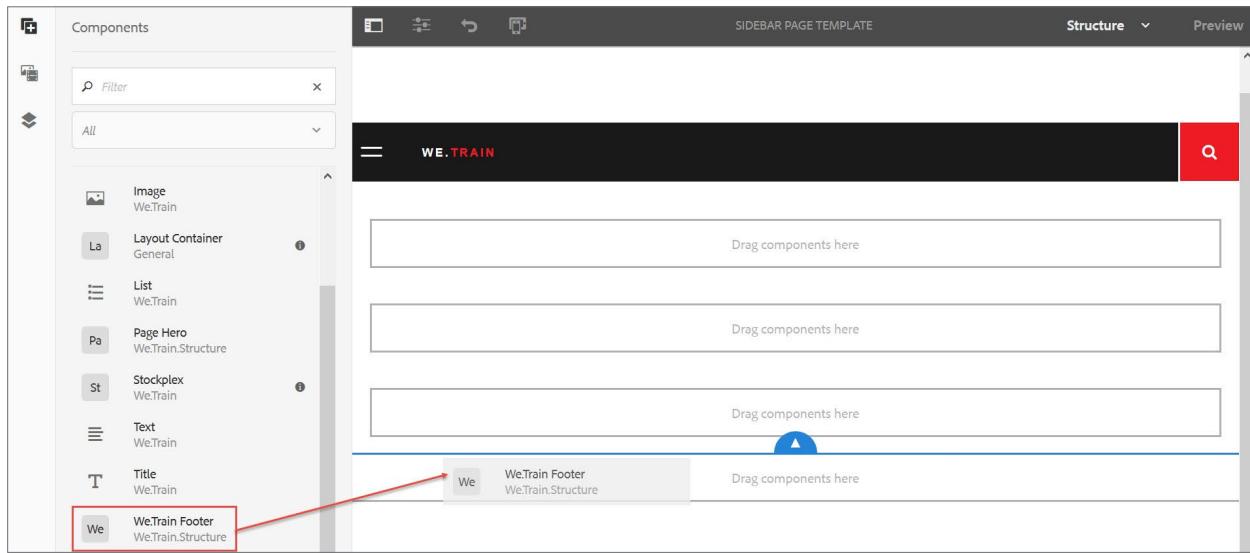


12. Drag the **Layout Container** component from the components browser, and then drop it below the header component as shown. The component is added.

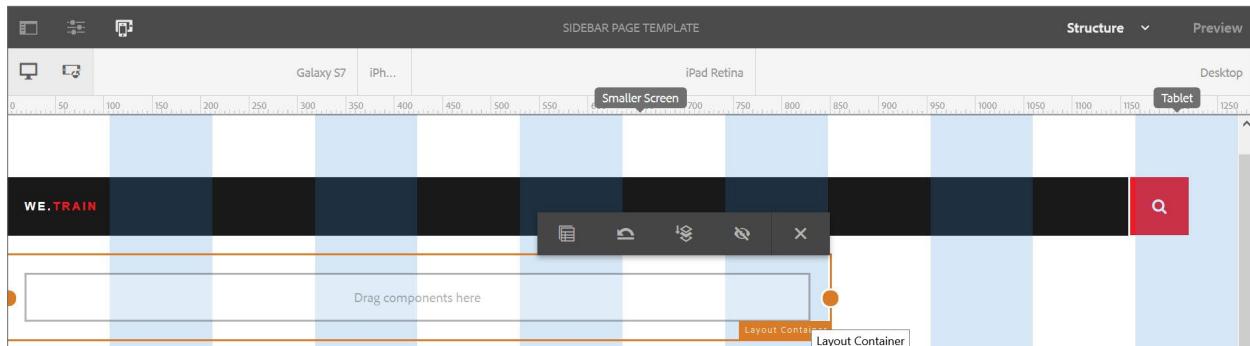


13. Similarly, add two more **Layout Container** components below the first **Layout Container** component.

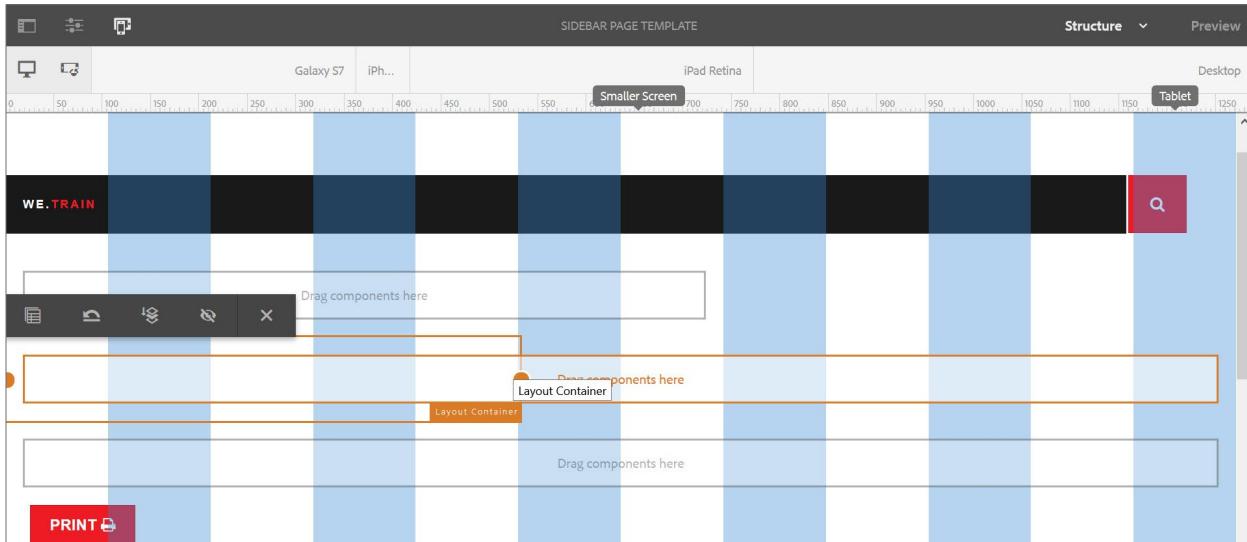
14. Drag the **We.Train Footer** component from the Components browser to the **Drag components here** area placeholder as shown. The component is added.



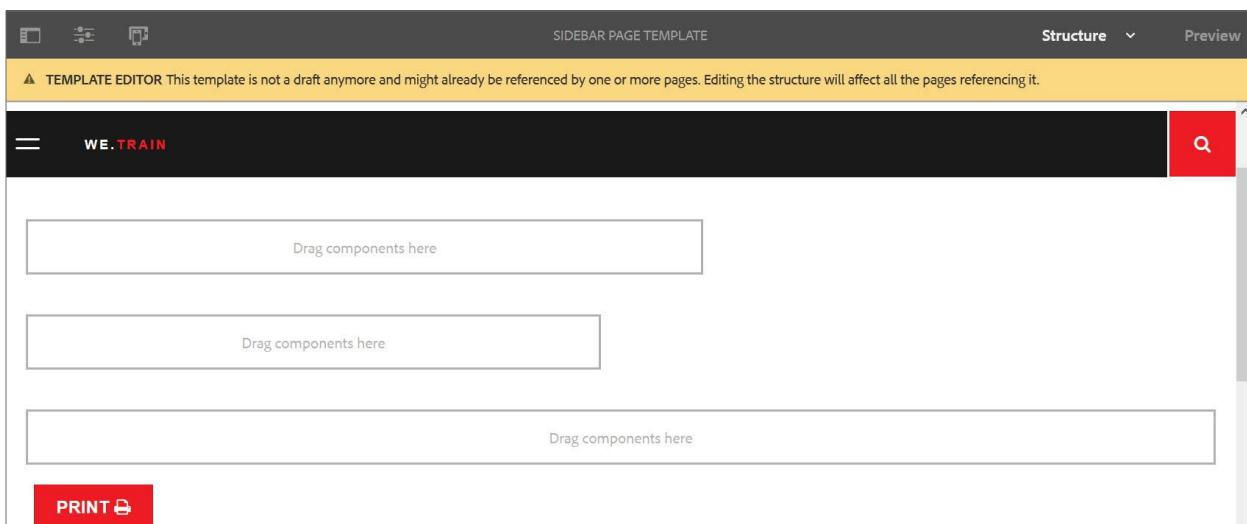
15. Select the first **Layout Container**, click the Layout icon from the component toolbar, and drag the right orange handler up to 2/3rd of the grid system as shown. The **Layout Container** resizes accordingly.



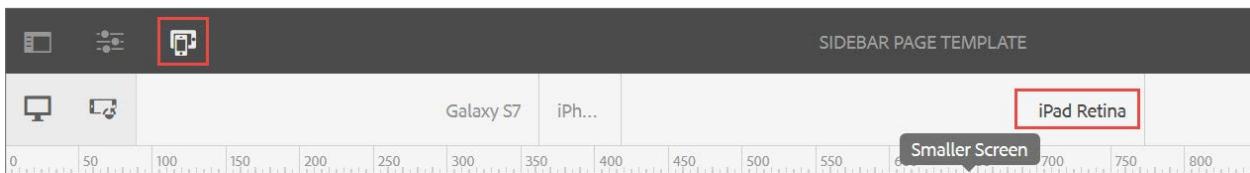
16. Select the middle **Layout Container**, click the Layout icon, and drag the right orange handler up to 1/3rd of the grid system as shown. The **Layout Container** resizes accordingly.



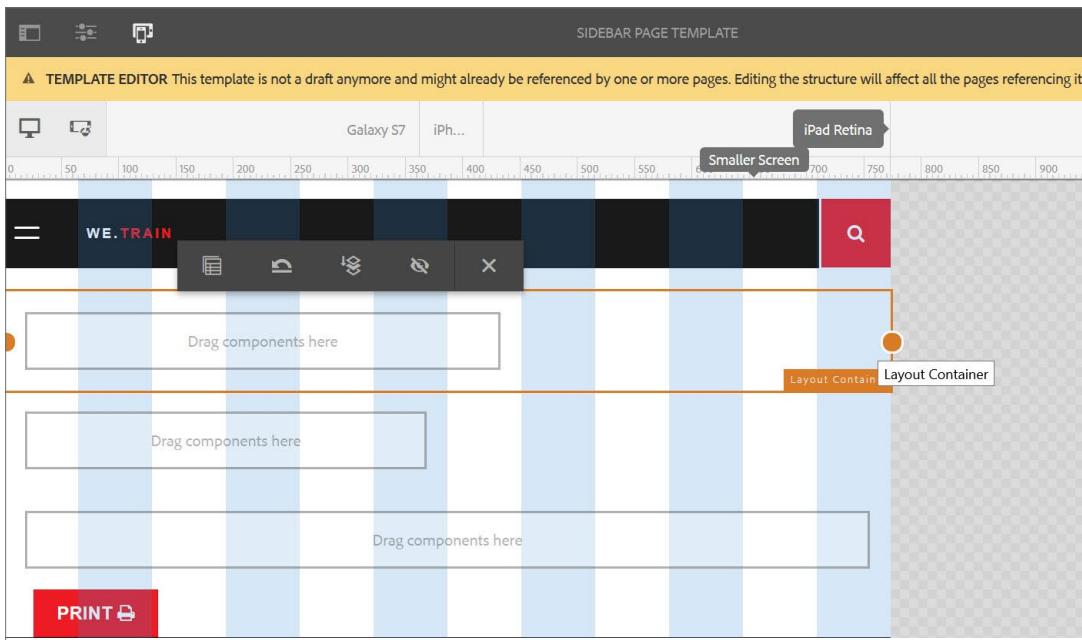
17. The last layout container will remain as it is (12/12). You will have a 2/3, 1/3, and 12/12 layout for Desktop as shown:



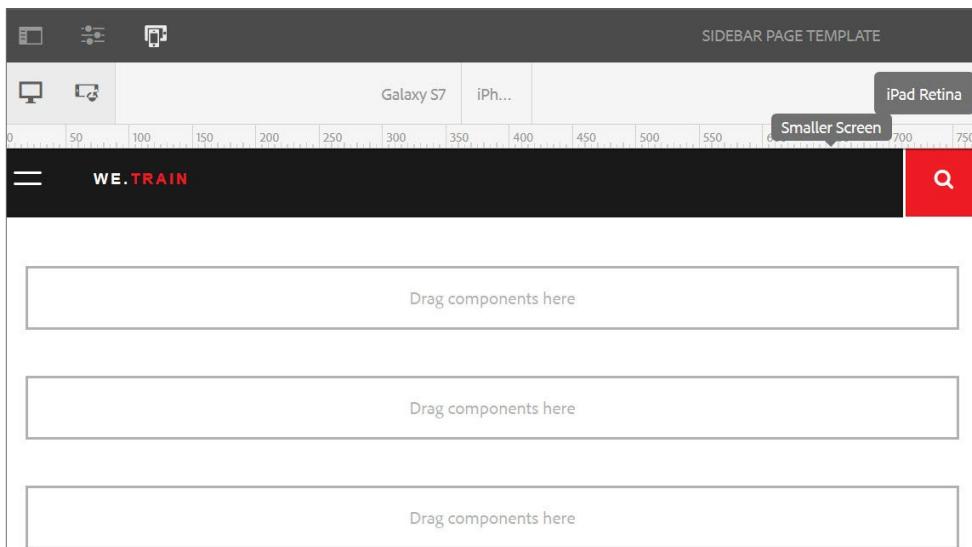
18. Click the Emulator icon from the toolbar, and click the **iPad Retina** device from the group as shown. The template rearranges itself to the device size.



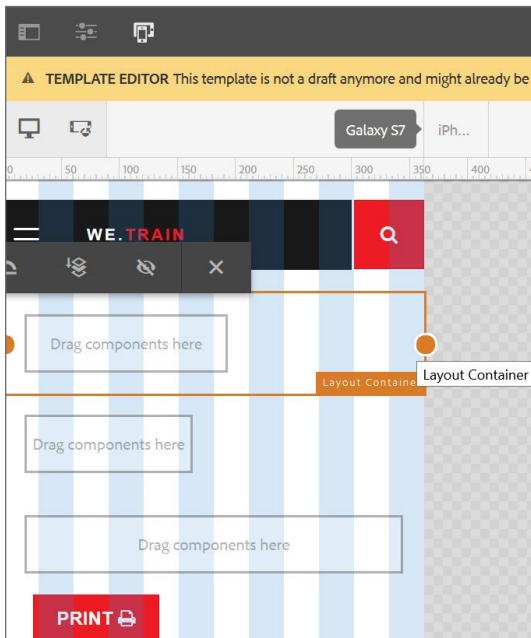
19. Select the first **Layout Container**, click the Layout icon, and adjust the container to take up the entire grid system as shown:



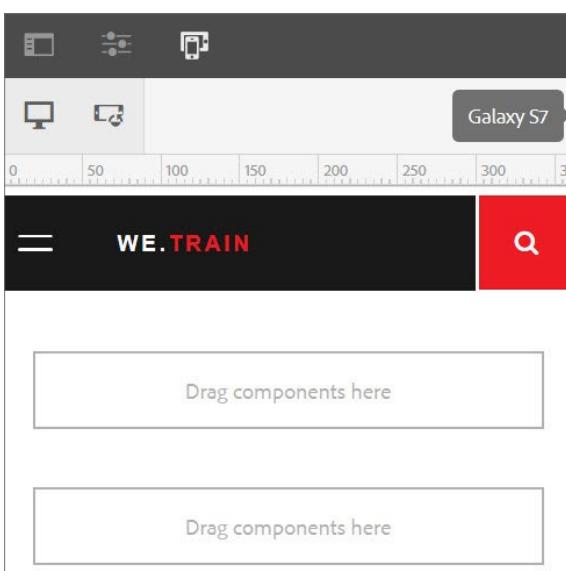
20. Similarly, adjust the other two **Layout Containers** to take up the entire grid system. You will have a 12/12, 12/12, 12/12 layout for Tablets as shown:



21. Click the Emulator icon from the toolbar, and click the **Galaxy S7** device from the group as shown. The template rearranges itself to the device size.
22. Select the top **Layout Container**, click the Layout icon and adjust the container to take up the entire grid system.
23. Select the middle **Layout Container**, click the Layout icon, and then click the Hide component (eye) icon as shown. The **Layout Container** is hidden from the grid.



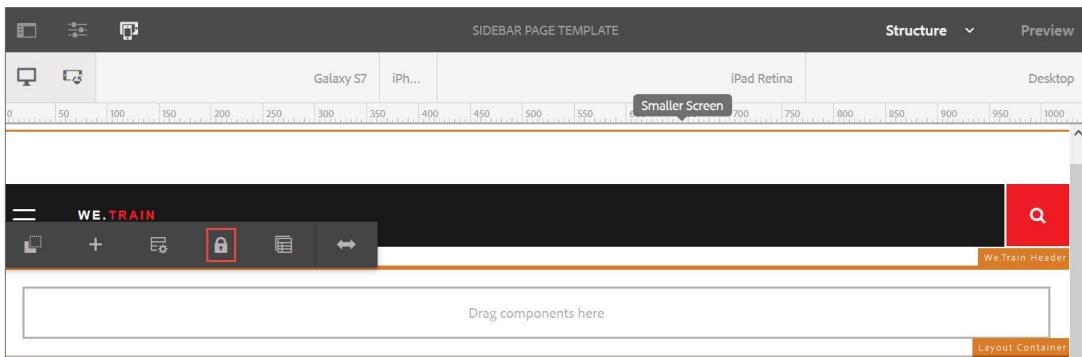
24. Select the last **Layout Container**, click the Layout icon and adjust the container to take up the entire grid system. You will have a 12/12 and 12/12 layout smaller screens and the right sidebar is hidden as shown:



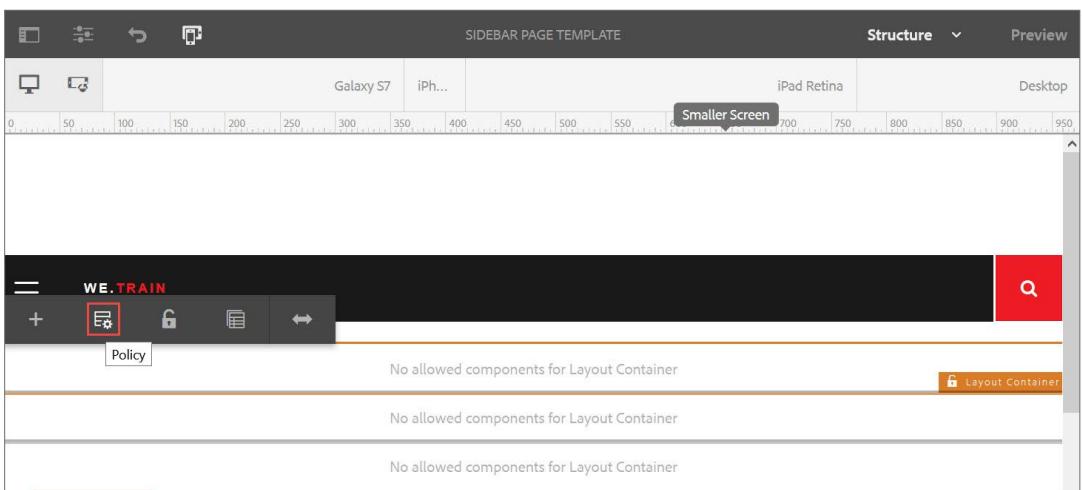
25. Click the Emulator icon from the toolbar, and click the **Desktop** device from the group. The template rearranges itself to the device size.

26. For each **Layout Container** you added, perform the following actions:

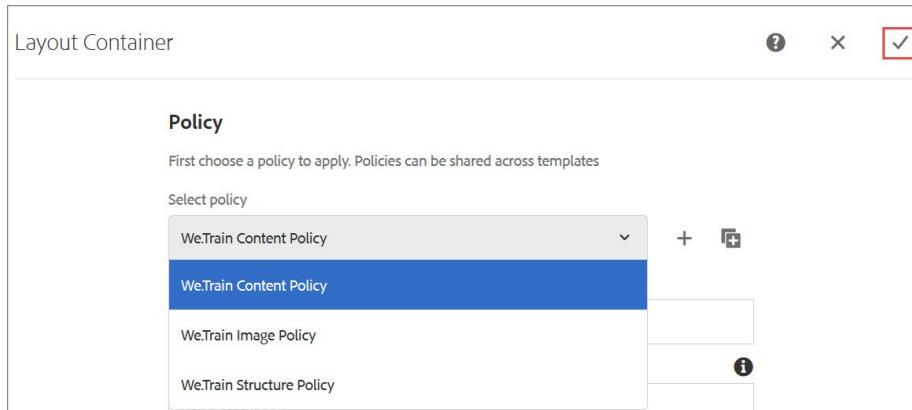
- Select the **Layout Container** component and click the Unlock structure component (lock) icon from the component toolbar as shown. The component is unlocked.



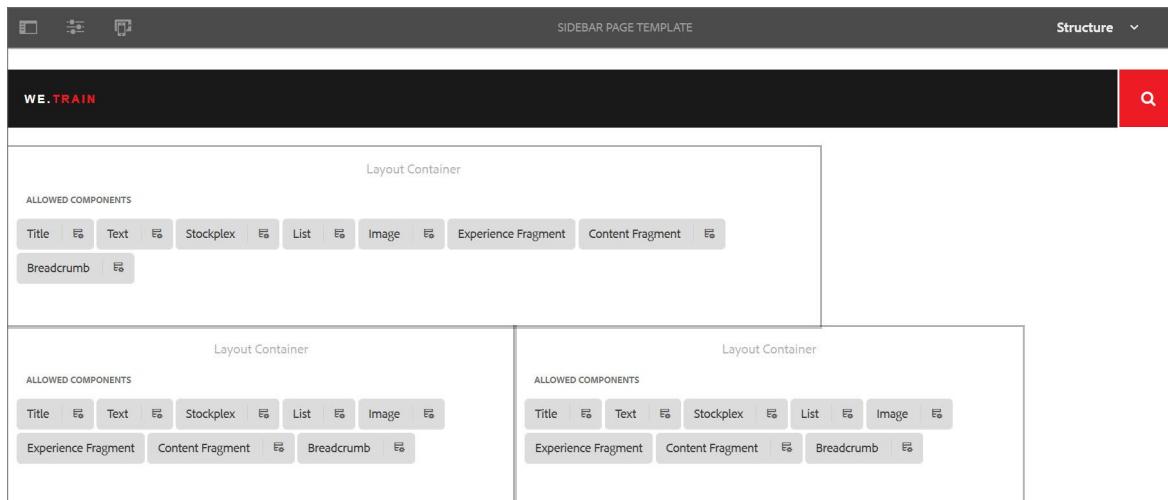
- Select the Layout Container component and click the Policy icon from the component toolbar as shown. The Layout Container wizard opens.



- i. In the **Policy** section, select the **We.Train Content Policy** from the **Select policy** drop-down menu.
- ii. Click the Done (checkmark) icon. You will be taken back to the template as shown:



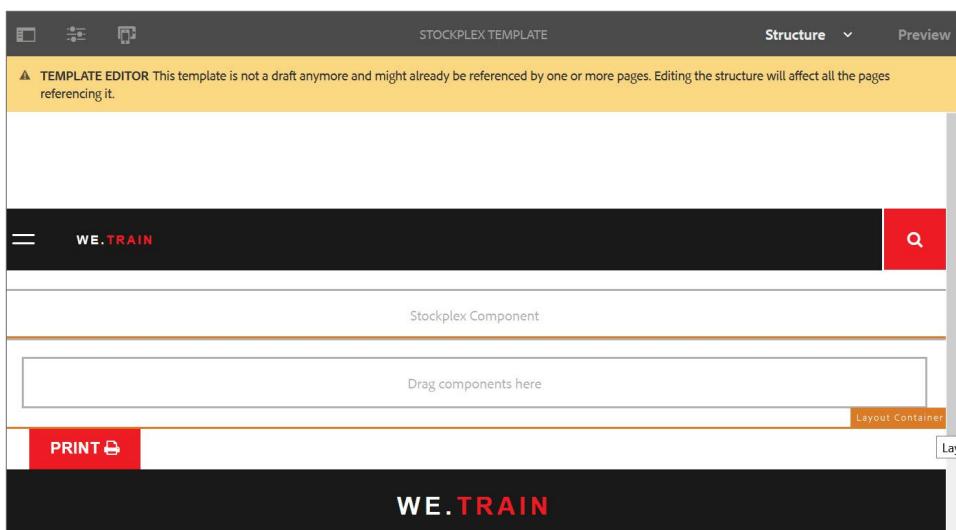
27. Your template should look similar to the one given in the below screenshot:



28. Navigate to the tab where the **Templates** console is open.
29. Select the **Sidebar Page Template**, and click **Enable** from the actions bar. The **Enable** dialog box opens.
30. Click **Enable**. The template is enabled.
31. With the **Sidebar Page Template** already selected, click **Publish** from the actions bar. The **Publish** dialog box opens.
32. Verify all checkboxes are selected, and click **Publish**. The **INFO The page has been published** success message appears.

To create a Stock Template:

33. Navigate to the tab where the **Templates** console is open.
34. Navigate to the **We.Train** folder and click **Create** from the actions bar. The **Create Template** wizard opens.
35. Select the **We.Train Empty Page** template and click **Next**. The **Template Details(2/2)** wizard opens.
36. Enter **Stockplex Template** in the **Template Title*** field, and then click **Create**. The **Success** dialog box opens.
37. Click **Open**. The **Stockplex Template** opens on a new tab.
38. Ensure the template is open in the **Structure** mode.
39. Click the Toggle Side Panel icon from the toolbar. The panel opens.
40. Click the Components icon. The Components browser opens.
41. Ensure **All** option is selected in the drop-down menu in the Components browser.
42. Drag the **We.Train Header** component from the Components browser to the **Drag components here** area placeholder. The **Header** component is added.
43. Drag the **Stockplex** component from the Components browser and drop it below the **Header** component. The **Stockplex** component is added.
44. Drag the **Layout Container** component from the Components browser and drop it below the **Stockplex** component. The **Layout Container** component is added.
45. Drag the **We.Train Footer** component from the Components browser and drop it below the **Layout Container** component. The footer component is added. Your page should look similar to the one given in the below screenshot:



46. Select the **Stockplex** component, and click the Unlock structure component (lock) icon from the component toolbar. The component is unlocked.

47. Select the **Layout Container** component, and click the Unlock structure component (lock) icon from the component toolbar. The component is unlocked.



48. Select the **Layout Container** component and click the Policy icon from the component toolbar.

The **Layout Container** wizard opens.

49. In the **Policy** section, select the **We.Train Content Policy** from the **Select policy** drop-down menu.

50. Click the Done (checkmark) icon. You will be taken back to the template.

51. Navigate to the tab where the **Templates** console is open.

52. Select the **Stockplex Template**, and click **Enable** from the actions bar. The **Enable** dialog box opens.

53. Click **Enable**. The template is enabled.

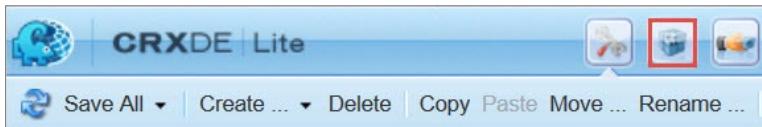
54. With the **Stockplex Template** already selected, click **Publish** from the actions bar. The **Publish** dialog box opens.

55. Ensure all checkboxes are selected, and click **Publish**. The **INFO The page has been published** success message appears.

Exercise 3: Create a code content package

In this exercise, you will include all your work and configurations in your environment into an exportable package.

1. Ensure you are in CRXDE Lite or add <http://localhost:4502/crx/de> URL in the address bar of the browser and press Enter. The **CRXDE Lite** page opens.
2. Click the Package icon from the header bar as shown. The **CRX Package Manager** page opens.



3. Click **Create Package** from the actions bar as shown. The **New Package** dialog box opens.

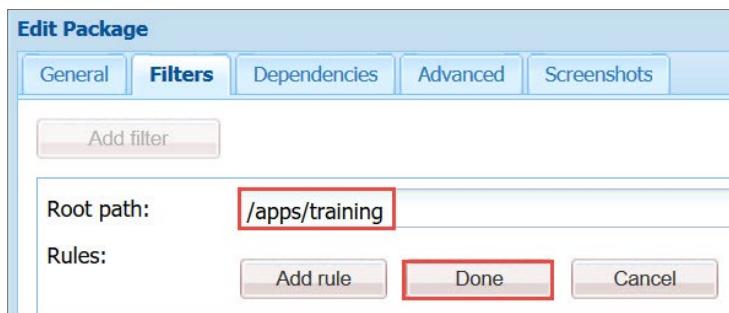


4. Enter the following details as shown:

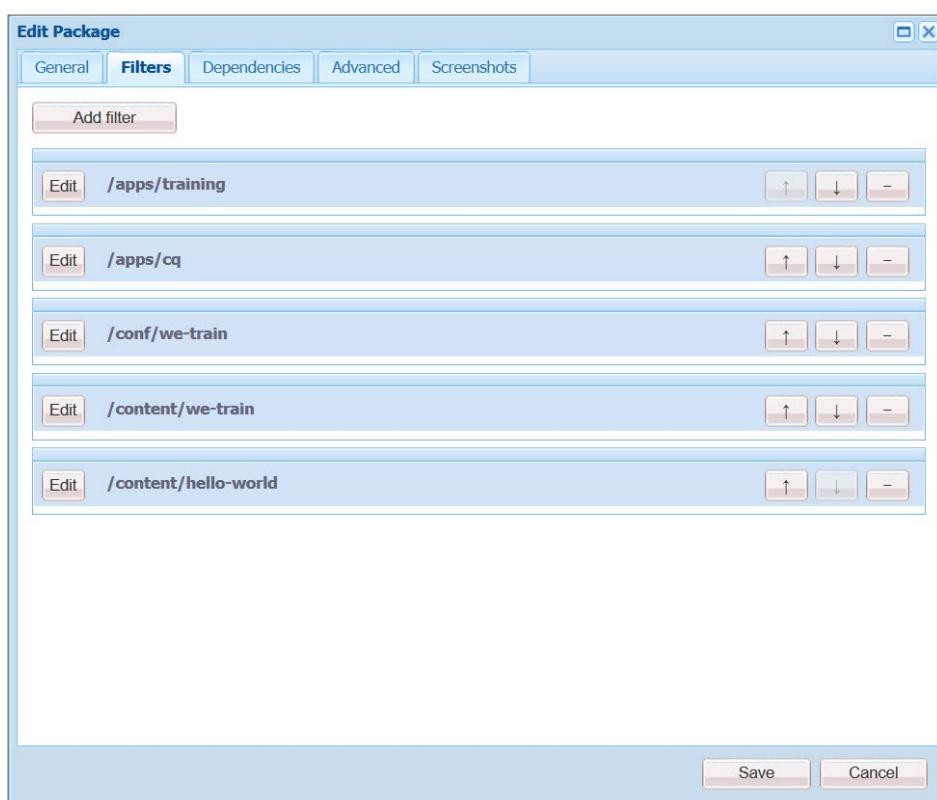
Field	Value
Package Name	<YourName>DevWebsites
Version	v1.0
Group	training

A screenshot of the 'New Package' dialog box. It contains three input fields: 'Package Name' with value 'chuckgrantDevWebsites', 'Version' with value 'v1.0', and 'Group' with value 'training'. Both the 'Group' field and the 'OK' button at the bottom are highlighted with red boxes. At the bottom right of the dialog box are 'OK' and 'Cancel' buttons.

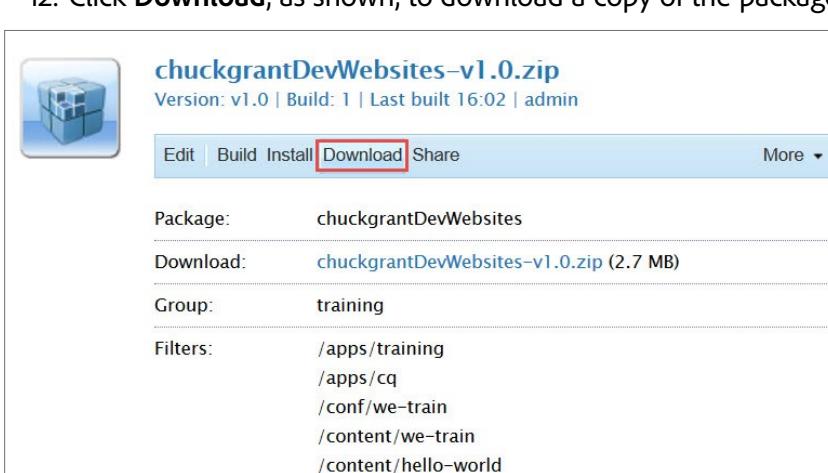
5. Click **OK**. You will be taken back to the **CRX Package Manager** page.
6. Click **Edit** in the newly-created package. The **Edit Package** dialog box opens.
7. To add filters to the package, click the **Filters** tab, and then click **Add filter**.
8. For the **Root path**, browse and select the **/apps/training** folder, click **OK**, and then click **Done** as shown:



9. Add the following additional filters (and click **Done** after adding each new filter):
 - a. /apps/cq
 - b. /conf/we-train
 - c. /content/we-train
 - d. /content/hello-world
10. Click **Save** as shown:



11. Click **Build** to build the package, and then click **Build** again in the confirmation dialog box. The package is now ready for download.
12. Click **Download**, as shown, to download a copy of the package to your computer.



NOTE: If you are using a ReadyTech environment, you can upload your package for use after class to a file-sharing tool such as the Adobe Document Cloud (<https://cloud.acrobat.com>). You must use your Adobe ID to access the Document Cloud. If you do not have an Adobe ID, you may register for one for free.

Alternatively, you can place the package file in c:\ReadyTech\Outbox and your instructor can retrieve them and email them to you at the end of class.

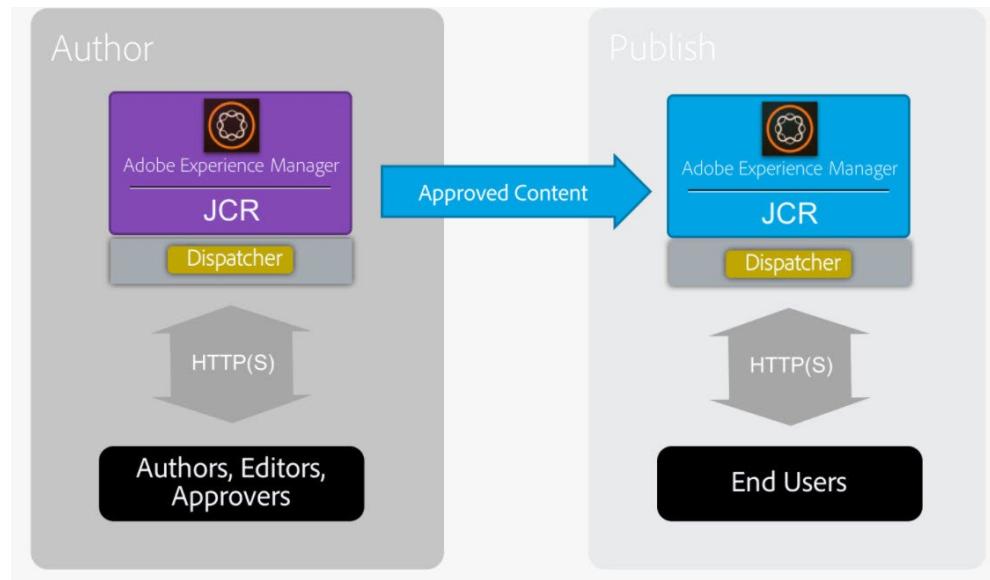
AEM Environment

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

Instances

In AEM terminology, an instance is a copy of AEM running on a server. AEM installations usually involve at least two instances running on separate computers and a dispatcher.

The author Instance is used by content authors, editors, and approvers to create and review content. When the content is approved, it is published to the publish Instance from where it is accessed by the end users.



Dispatcher

The Dispatcher is the caching and/or load balancing tool used by AEM. It handles caching and URL filtering and is installed on the webserver. You can increase the security of your AEM instance by using the Dispatcher in conjunction with an enterprise-class webserver.

Why use Dispatcher to implement Caching?

There are two basic approaches to web publishing:

- Using static Web Servers, such as Apache or IIS. This approach is simple and fast.
- Using Content Management Servers, which provide dynamic, real-time, intelligent content but require much more computation time and other resources.

The Dispatcher creates realize an environment that is both fast and dynamic. It works as part of a static HTML server, such as Apache and:

- Store as much of the site content as possible in the form of a static website
- Access the layout engine as little as possible.

The static content is handled with exactly the same speed and ease as on a static web server. In addition, you can use the administration and security tools available for your static web server(s).

The dynamic content is generated as needed, without slowing down the system.

The Dispatcher contains mechanisms to generate and update static HTML based on the content of the dynamic site. You can specify in detail which documents are stored as static files and which are always generated dynamically.

Replication Agents

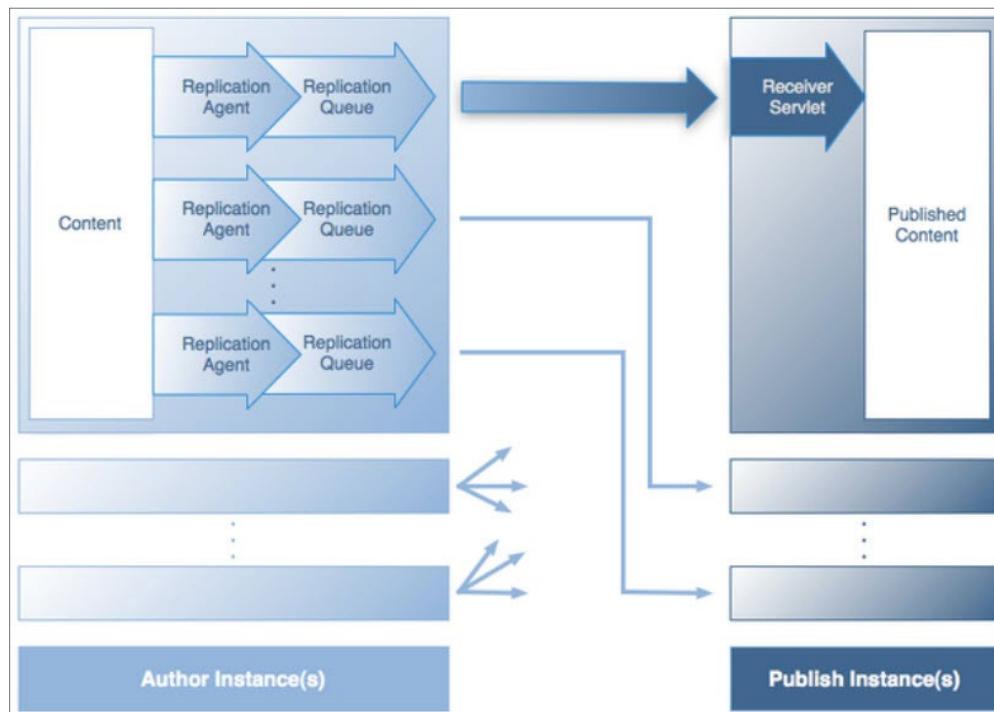
Replication agents help:

- Publish (activate) content from an author environment to a publish environment
- Flush content from the Dispatcher cache explicitly
- Return user input, such as form input, from the publish environment to the author environment

Replicating from Author to Publish

Replication, to a publish instance or a Dispatcher, occurs in several steps:

1. The author requests that certain content be published (activated). This can be initiated by a manual request or by preconfigured automatic triggers.
2. The request is passed to the appropriate default replication agent. An environment can have several default agents, which will be selected for such actions.
3. The replication agent "packages" the content and places it in the replication queue.
4. From the Websites tab, the colored status indicator is set for the individual pages.
5. The content is transported from the queue to the publish environment by using the configured protocol, usually HTTP.
6. A servlet in the publish environment receives the request and publishes the received content. The default servlet is `http://localhost:4503/bin/receive`.



Replicating from Publish to Author

In some cases, the data from the publish instance is returned to the author instance by using a reverse replication and then redistributed to other publish environments. Due to security considerations, any traffic from the publish instance to the author instance must be strictly controlled.

The reverse replication uses an agent in the publish instance to reference the author instance. This agent places the data into an outbox. This outbox is matched with replication listeners in the author instance. The listeners poll the outboxes to collect any data entered and then distribute it as necessary. This ensures that the author instance controls all traffic.

In other cases, such as forums, blogs, comments, and reviews, the amount of user-generated content (UGC) being entered in the publish instance is difficult to efficiently synchronize across AEM instances by using replication

Performance Guidelines

You must follow these guidelines when working with AEM:

- TarMK with File Datastore is the recommended architecture for most customers:
 - › The minimum topology is one author instance, two publish instances, and two Dispatchers.
 - › The binary-less replication is turned on if the File Datastore is shared.
- MongoMK with File Datastore is the recommended architecture for horizontal scalability of the author tier:
 - › The minimum topology is three author instances, three MongoDB instances, two publish instances, and two Dispatchers.
 - › The binary-less replication is turned on if the File Datastore is shared.
- Nodestore should be stored on the local disk and not on a Network Attached Storage (NAS).
- Amazon S3 is the recommended datastore for a total volume of assets above 5TB:
 - › The Amazon S3 datastore is shared between the author tier and the publish tier.
 - › The binary-less replication must be turned on.
 - › Datastore Garbage Collection requires a first run on all author nodes and publish nodes and then a second run on author nodes.
- The custom index should be created in addition to the out-of-the box index based on most common searches:
 - › Lucene indexes should be used for custom indexes.
- Workflow customization can substantially improve the performance. For example, you can remove the video step in the Update Asset workflow and disable the listeners that are not used.

Security Checklist

You must follow this checklist when working with AEM security:

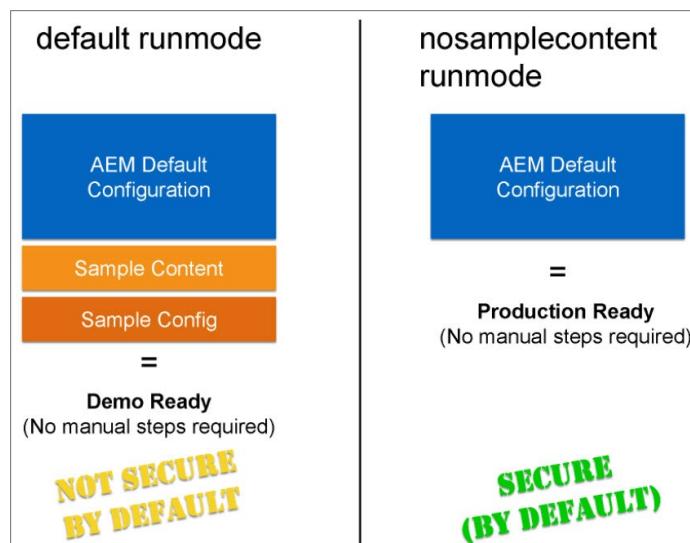
- Run AEM in the production ready mode
- Enable HTTPS for the transport layer security
- Install security hotfixes
- Change the default passwords for the AEM and OSGi console admin accounts
- Implement the custom error handler
- Complete the Dispatcher security checklist

AEM in Production Ready Mode

AEM includes a **nosamplecontent** runmode to automate the process to prepare an AEM instance for deployment in a production environment.

The new run mode will automatically configure the instance to adhere to the security best practices described in the security checklist and remove all sample We.Retail applications and configurations in the process.

The following screenshot depicts the AEM runmode:



To run AEM in production ready mode, you must add the **nosamplecontent** through the **-r** runmode switch to your existing startup arguments, as shown:

```
java -jar aem-quickstart.jar -r nosamplecontent
```

You must make the following configuration changes when running AEM in production ready mode:

- The CRXDE Support bundle (`com.adobe.granite.crxde-support`) is disabled by default in production ready mode. It can be installed at any time from the Adobe public Maven repository.
- The Apache Sling Simple WebDAV Access to repositories (`org.apache.sling.jcr.webdav`) bundle will only be available on author instances.
- Newly created users need to change their password on the first login. This does not apply to the admin user.
- Generate debug info is disabled for the Apache Java Script Handler.
- Mapped content and Generate debug info are disabled for the Apache Sling JSP Script Handler.
- The Day CQ WCM Filter is set to edit on author and disabled on publish instances.
- The Adobe Granite HTML Library Manager is configured with the following settings:
 - a. Minify: enabled
 - b. Debug: disabled
 - c. Gzip: enabled
 - d. Timing: disabled
- The Apache Sling GET Servlet is set to support secure configurations by default. The following table lists the configurations:

Configuration	Author	Publish
TXT rendition	disabled	disabled
HTML rendition	disabled	disabled
JSON rendition	enabled	enabled
XML rendition	disabled	disabled
<code>json.maximumresults</code>	1000	100
Auto Index	disabled	disabled

References

You can use the following links for more information on:

- Resources for AEM developers:

- › WKND Tutorial:

<https://helpx.adobe.com/experience-manager/kt/sites/using/getting-started-wknd-tutorial-develop.html>

- › ACS Commons:

<https://adobe-consulting-services.github.io/acs-aem-commons/>

- › AEM Developer Chrome Extension:

<https://chrome.google.com/webstore/detail/aem-developer/hgjhcngrldfpakbnffnbnmcmohfmfc?hl=en-US>

- › AEM GEMs sessions:

<https://docs.adobe.com/content/ddc/en/gems.html>

- › AEM Community:

<https://forums.adobe.com/community/experience-cloud/marketing-cloud/experience-manager>

- › AEM Developer Docs:

<https://helpx.adobe.com/experience-manager/6-4/sites/developing/user-guide.html>

- › AEM Featured Videos:

<https://helpx.adobe.com/experience-manager/kt/index/aem-6-4-videos.html>

- › AEM Tutorials:

<https://helpx.adobe.com/experience-manager/tutorials.html>

- Dispatcher:

<https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher.html>

- Replication Agents:

<https://helpx.adobe.com/experience-manager/6-4/sites/deploying/using/replication.html>

- Performance Guidelines:

<https://helpx.adobe.com/experience-manager/6-4/sites/deploying/using/performance-guidelines.html>

- Security Checklist:

<https://helpx.adobe.com/experience-manager/6-4/sites/administering/using/security-checklist.html>

- Production Ready Mode:

<https://helpx.adobe.com/experience-manager/6-4/sites/administering/using/production-ready.html>



Testing and Debugging Site Content

Introduction

Before you plan to launch your websites created in Adobe Experience Manager (AEM) and make them available for visitors across the globe, you need to thoroughly test the website content. This will ensure the website works on different device touch-points seamlessly and provides consistent experience to all visitors.

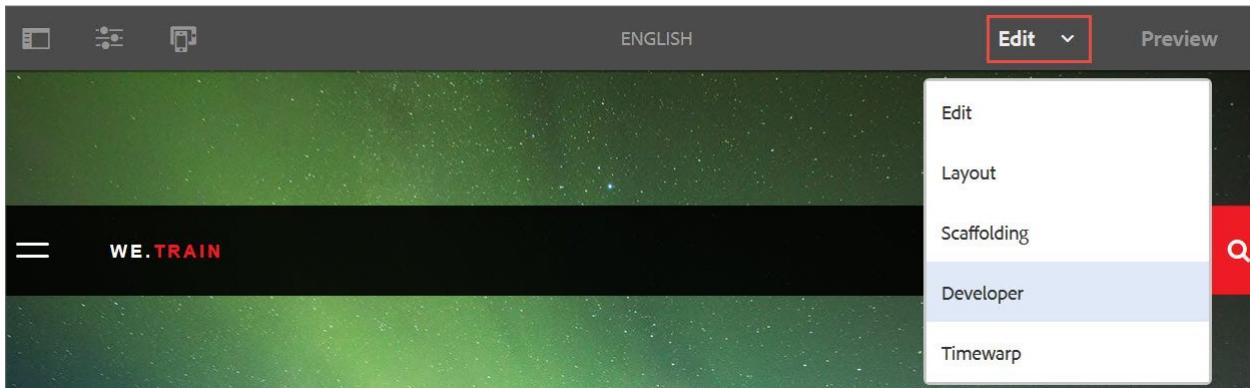
Objectives

After completing this course, you will be able to:

- Explain how to debug errors using the Developer mode
- Explain how to debug errors using parameters
- Explain the Hobbes Testing framework
- Create a Hobbes test suite
- Run the Hobbes test suite

Developer Mode

The Developer mode appears as a side panel of the page editor. To open this mode, select Developer from the mode selector in the toolbar of the page editor as shown:



There are two tabs in the Developer mode, Components and Errors, as shown:.

A screenshot of the "Components" tab in the developer mode sidebar. The sidebar has a red box around the "Errors" tab icon. The "Components" tab is active, showing a tree view of page components with their execution times: "page (4)" (0.78s), "header" (8ms), "servicecomponents" (5ms), "footer" (5ms), and "responsivegrid (5)" (0.60s). There are edit icons next to each component entry.

Components

The Components tab displays a component tree that:

- Outlines the chain of components and templates rendered on the page (SLY and JSP). You can expand the tree to show context within the hierarchy.
- Shows the server-side computational time needed to render the component.
- Enables you to expand the tree and select specific components within the tree. When you select a component, details such as the repository path and the links to scripts (accessed in CRXDE Lite) are displayed.
- Highlights the selected components (in the content flow, indicated by a blue border).

The Components tab provides you with the associated script for each component. You can click the Edit icon to navigate directly to the script in CRXDE Lite.

Errors

The Errors tab displays a warning if:

- A component writes an entry to the error log with error details and direct links to the appropriate code within CRXDE Lite
- A component opens an admin session

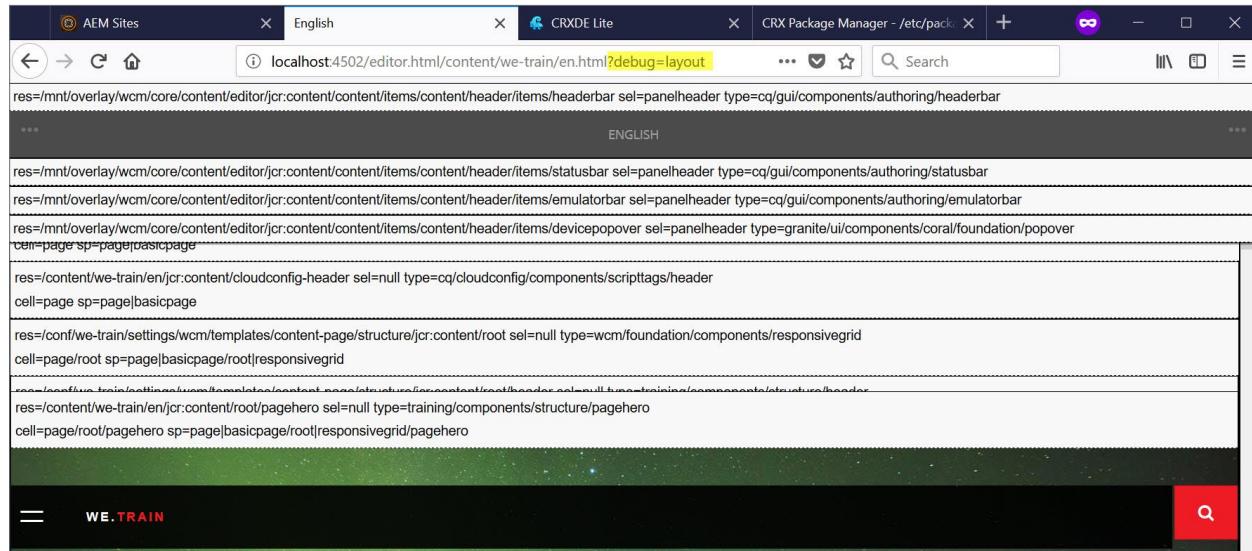
Parameter Debugging

You can debug errors on a page by appending the following parameters in the URL:

- ?debug=layout
- ?debugClientLibs=true

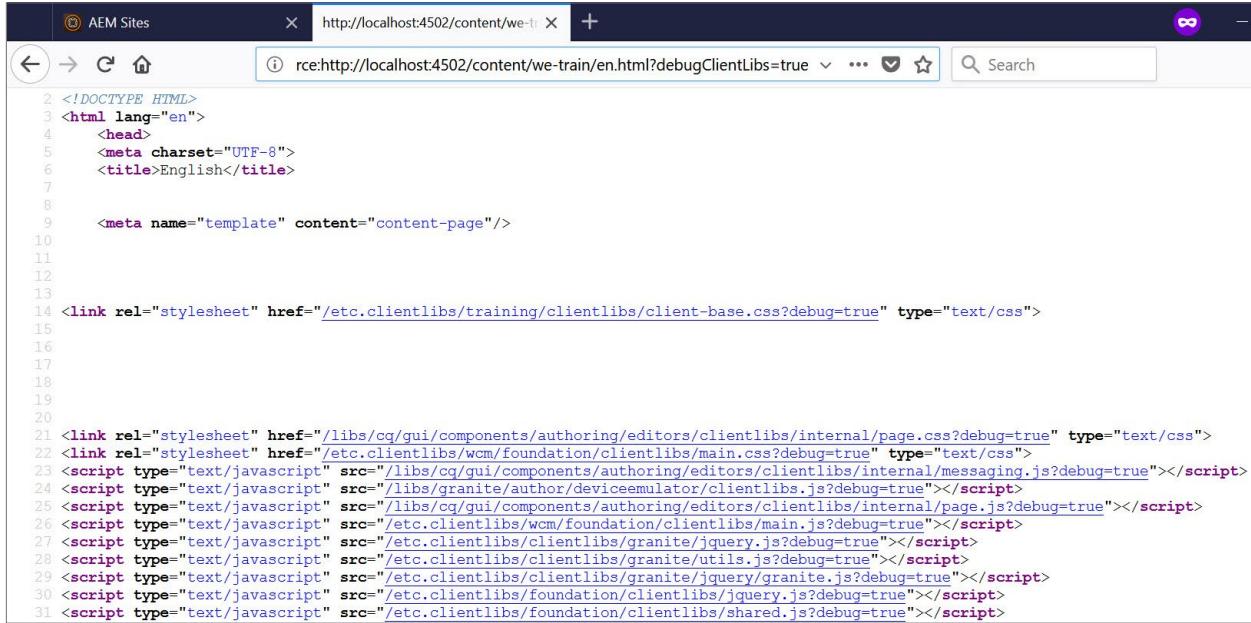
?debug=layout

This parameter lists the information about renderers, selectors, and resources as shown:



?debugClientLibs=true

This parameter lists the client libraries loaded onto the page in the page source HTML as shown:



The screenshot shows a browser window with the URL `rce:http://localhost:4502/content/we-train/en.html?debugClientLibs=true`. The page source code is displayed, showing numerous `<link>` and `<script>` tags for client libraries. The code is heavily annotated with line numbers from 1 to 31, indicating the specific lines of the source code.

```

2  <!DOCTYPE HTML>
3  <html lang="en">
4    <head>
5      <meta charset="UTF-8">
6      <title>English</title>
7
8
9      <meta name="template" content="content-page"/>
10
11
12
13
14      <link rel="stylesheet" href="/etc.clientlibs/training/clientlibs/client-base.css?debug=true" type="text/css">
15
16
17
18
19
20
21      <link rel="stylesheet" href="/libs/cq/gui/components/authoring/editors/clientlibs/internal/page.css?debug=true" type="text/css">
22      <link rel="stylesheet" href="/etc.clientlibs/wcm/foundation/clientlibs/main.css?debug=true" type="text/css">
23      <script type="text/javascript" src="/libs/cq/gui/components/authoring/editors/clientlibs/internal/messaging.js?debug=true"></script>
24      <script type="text/javascript" src="/libs/granite/author/deviceemulator/clientlibs.js?debug=true"></script>
25      <script type="text/javascript" src="/libs/cq/gui/components/authoring/editors/clientlibs/internal/page.js?debug=true"></script>
26      <script type="text/javascript" src="/etc.clientlibs/wcm/foundation/clientlibs/main.js?debug=true"></script>
27      <script type="text/javascript" src="/etc.clientlibs/clientlibs/granite/jquery.js?debug=true"></script>
28      <script type="text/javascript" src="/etc.clientlibs/clientlibs/granite/utils.js?debug=true"></script>
29      <script type="text/javascript" src="/etc.clientlibs/clientlibs/granite/jquery/granite.js?debug=true"></script>
30      <script type="text/javascript" src="/etc.clientlibs/foundation/clientlibs/jquery.js?debug=true"></script>
31      <script type="text/javascript" src="/etc.clientlibs/foundation/clientlibs/shared.js?debug=true"></script>

```

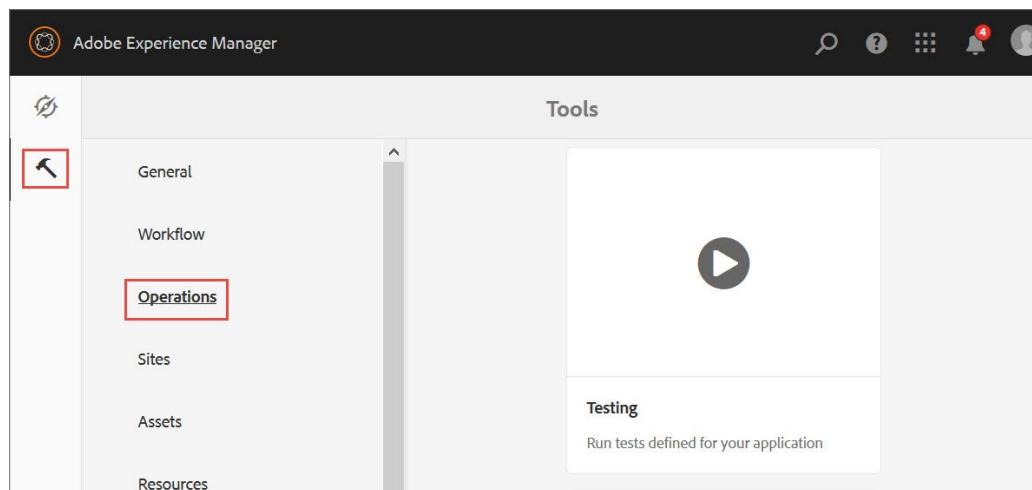
Hobbes Testing Framework

Functional testing is a software testing process used within software development in which software is tested to ensure it conforms with all requirements and is dedicated to the user experience. Before you can publish your site, you need to verify every component in it is working.

AEM provides a framework for automating tests for UI. Using the framework, you can write and run UI tests directly in a web browser. The framework provides a JavaScript API for creating tests.

The AEM test framework uses Hobbes.js, a testing library written in JavaScript. The Hobbes.js framework was developed for testing AEM as part of the development process. The framework is now available for public use for testing your AEM applications. Hobbes test suites are created as modified client libraries inside the project. The client library has specific category and dependency values. Each test suite is a JavaScript file within the client library.

You can access the Hobbes Testing framework from **Tools > Operations > Testing** console as shown:

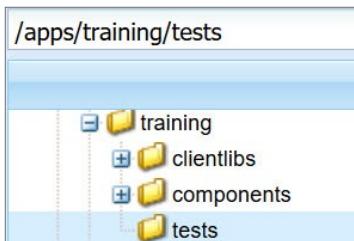


Exercise 1: Create a Hobbes testing suite

You need to install the **adls-training-project-v7.0.zip** package on your AEM author instance. The package contains the required training project structure, page component with clientlibs, template, and site structure that has already been created for you.

To create a Hobbes testing suite:

1. Click **Adobe Experience Manager** from the header bar. You will be in the **Tools** console.
2. Click **CRXDE Lite**. The **CRXDE Lite** page opens.
3. Navigate to the **/apps/training** folder.
4. Right-click the **training** folder, and select **Create... > Create Node...** from the list. The **Create Node** dialog box opens.
5. Enter the following:
 - a. **Name: tests**
 - b. **Type: cq:ClientLibraryFolder**
6. Click **OK**. The node is added to the **training** folder as shown:



7. Click **Save All** from the actions bar.



Note: You can also press **Ctrl+S** (Windows users) and **Command+S** (Mac users) to save your work in CRXDE Lite.

8. Add the following two array properties to the **tests** node as shown:

Name	Type	Value
categories	String[]	granite.testing.hobbes.tests
dependencies	String[]	granite.testing.hobbes.testrunner

Name	categories	Type	String	Value	granite.testing.hobbes.tests	Multi		
------	------------	------	--------	-------	------------------------------	--------------	--	--

Name	dependencies	Type	String	Value	granite.testing.hobbes.testrunner	Multi		
------	--------------	------	--------	-------	-----------------------------------	--------------	--	--

9. Click **Save All** from the actions bar.

10. Right-click the **tests** folder, and select **Create... > Create File...** from the list. The **Create File** dialog box opens.

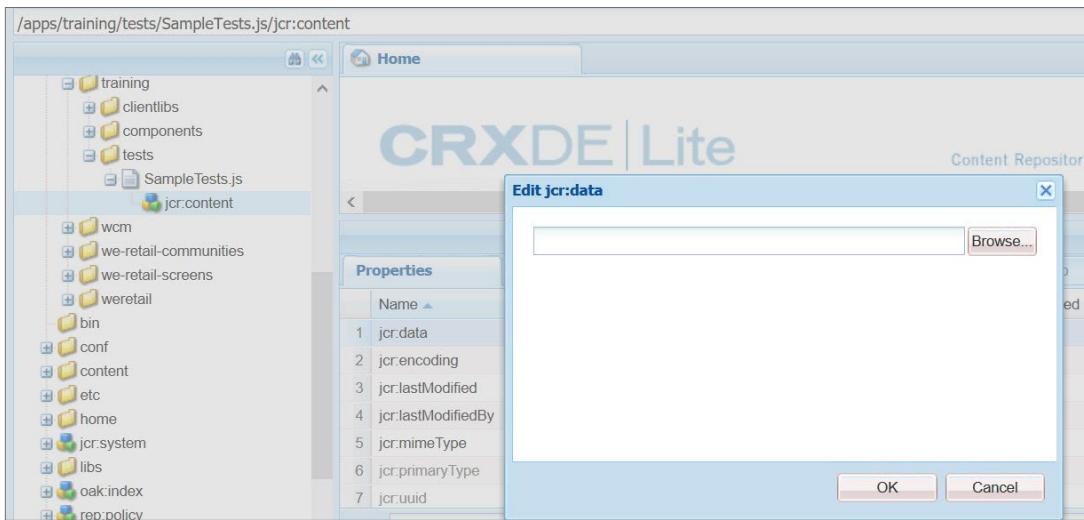
11. Enter **SampleTests.js** in the **Name** field, and then click **OK**. The file is added to the **tests** folder.

12. Click **Save All** from the actions bar.

13. Select the **jcr:content** node that is below **SampleTests.js**. The **Properties** tab opens.

14. Double-click the **jcr:data** property. The **Edit jcr:data** dialog box opens.

15. Click **Browse** as shown. The **Open** dialog box opens.



16. Navigate to the **Exercise_Files** folder on your file system.

17. Select **SampleTests.js** and then click **Open**.

18. Click **OK** in the **Edit jcr:data** dialog box. The new code is added to **SampleTests.js**.

19. Open the **SampleTests.js** to view the code you added. Observe, the following code snippet:

```

01. hobs.param("siteURL", "/content/we-train");
02. hobs.param("page1", "/en");
03. hobs.param("page2", "/en/about-us");
04.
05. new hobs.TestSuite("We.Train Tests", {path:"/apps/training/tests/SampleTests.js", register: true})
06.
07.     .addTestCase(new hobs.TestCase("Hero component on Home Page")
08.         .navigateTo("%siteURL%page1%.html")
09.         .asserts.location("%siteURL%page1%.html", true)
10.         .asserts.visible(".cmp-pagehero", true)
11.     )
12.
13.     .addTestCase(new hobs.TestCase("Hero component on SubPage")
14.         .navigateTo("%siteURL%page2%.html")
15.         .asserts.location("%siteURL%page2%.html", true)
16.         .asserts.visible(".cmp-pagehero", true)
17.     )
18.
19.     .addTestCase(new hobs.TestCase("Print Selector inserted on added")
20.         .navigateTo("%siteURL%page1%.html")
21.         .asserts.location("%siteURL%page1%.html", true)
22.         .click("a.btn-print",true)
23.         .asserts.isTrue(function(){ return hobs.window.location.href.indexOf("/en.print.html") > -1;})
24. );

```

20. Click **Save All** from the actions bar.

21. Right-click the **tests** folder, and select **Create... > Create File...** from the list. The **Create File** dialog box opens.

22. Enter **js.txt** in the **Name** field, and then click **OK**. The file is added to the **tests** folder.

23. Click **Save All** from the actions bar.

24. Update the **js.txt** script by replacing the existing code with new code (**js.txt**) from the **Exercise_Files** folder.

25. Open the **js.txt** and observe the the code you added as shown:

```

01. //Add Hobbes tests to the clientlib
02. SampleTests.js

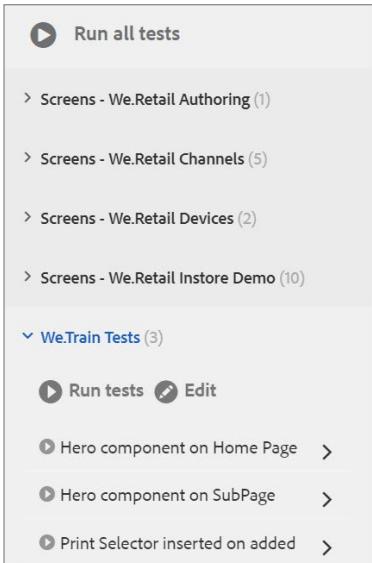
```

26. Click **Save All** from the actions bar.

Exercise 2: Run the Hobbes testing suite

To run a Hobbes testing suite:

1. Click **Adobe Experience Manager** from the header bar. You will be in the **Tools** console.
2. Click **Operations**, and then **Testing**. The **Testing** console opens.
3. Notice that the **We.Train Tests** you created in the previous exercise appear in the Testing console:



- Click the drop-down arrow to the left of **We.Train Tests** (3).
- Click **Run tests** from the panel to run the Hobbes testing suite. The tests may take up to 10-15 seconds to complete. Notice, all tests pass successfully.

The screenshot shows the We.Train Tests interface. On the left, there's a sidebar with a 'Run all tests' button and a list of test categories: Screens - We.Retail Authoring (1), Screens - We.Retail Channels (5), Screens - We.Retail Devices (2), Screens - We.Retail Instore Demo (10), and We.Train Tests (3). The 'We.Train Tests' category is expanded, showing three individual test items: 'Hero component on Home Page' (green checkmark), 'Hero component on SubPage' (green checkmark), and 'Print Selector inserted on added' (green checkmark). Each test item has a right-pointing arrow. Below the sidebar, there's a preview area titled 'WE.TRAIN: English Page Print' showing the text 'English', 'My new title', and 'My custom subtitle'. At the bottom of the sidebar, there are 'Run tests' and 'Edit' buttons.

Run all tests	WE.TRAIN: English Page Print
> Screens - We.Retail Authoring (1)	English
> Screens - We.Retail Channels (5)	My new title
> Screens - We.Retail Devices (2)	My custom subtitle
> Screens - We.Retail Instore Demo (10)	
▼ We.Train Tests (3)	
Run tests Edit	
✓ Hero component on Home Page >	
✓ Hero component on SubPage >	
✓ Print Selector inserted on added >	

You will see many different pieces of content "flash" before your eyes. Once testing is completed, notice how all tests pass successfully, which is indicated by the green circle (with a checkmark inside).



Note: If your tests still fail, you need to run another browsing tab in **Incognito Mode** in Google Chrome (if you are using that browser). This may be due to a caching issue. If you are using IE, use File > New Session.

References

You can use the following links for more information on:

- Testing your UI:

<https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/hobbes.html>

- Hobbes.js documentation:

<https://helpx.adobe.com/experience-manager/6-3/sites/developing/using/reference-materials/test-api/index.html>



Customizing AEM Development Environment

Introduction

Adobe Experience Manager (AEM) has a default code editor CRXDE Lite that is used for developing, testing, and debugging code. You can also create AEM projects in other Integrated Development Environments (IDE) such as Eclipse, Brackets, and Lazybones. You can also install extensions to synchronize the AEM content repository with IDEs.

Objectives

After completing this course, you will be able to:

- Explain how Eclipse, Maven, and Lazybones work
- Explain the features of Brackets
- Install Brackets and the AEM Brackets Extension
- Make changes to the repository by using Brackets

Eclipse, Maven, and Lazybones

You can create an AEM application by using the Java Eclipse Integrated Development Environment (IDE). This helps you access the features within the IDE, such as code completion and the ability to remote debug the application. That is, you can set a breakpoint on a line of Java code used for an OSGi bundle and you can walk through the code to troubleshoot issues.

You can synchronize code [both Java code and Java Server pages (JSP) code] in Eclipse with the code in the AEM Java Content Repository (JCR). For example, you have the application logic in Eclipse that represents a JSP component. You can synchronize the code in Eclipse with code in the AEM JCR by using the vault tool. That is, you can check in code you write in Eclipse into AEM JCR.

You can download the latest version of Eclipse from the following link:

<https://eclipse.org/downloads/index-developer.php>

Maven

Apache Maven is an open source tool for managing software projects by automating builds and providing quality project information. It is the recommended build management tool for AEM projects.

You can use Maven to build an OSGi bundle that uses the JCR API. Maven manages the JAR files that a Java project needs in its class path. Instead of searching the Internet trying to find and download third-party JAR files to include in your project's class path, Maven manages these dependencies for you.

You can download Maven 3 from the following link:

<http://maven.apache.org/download.html>

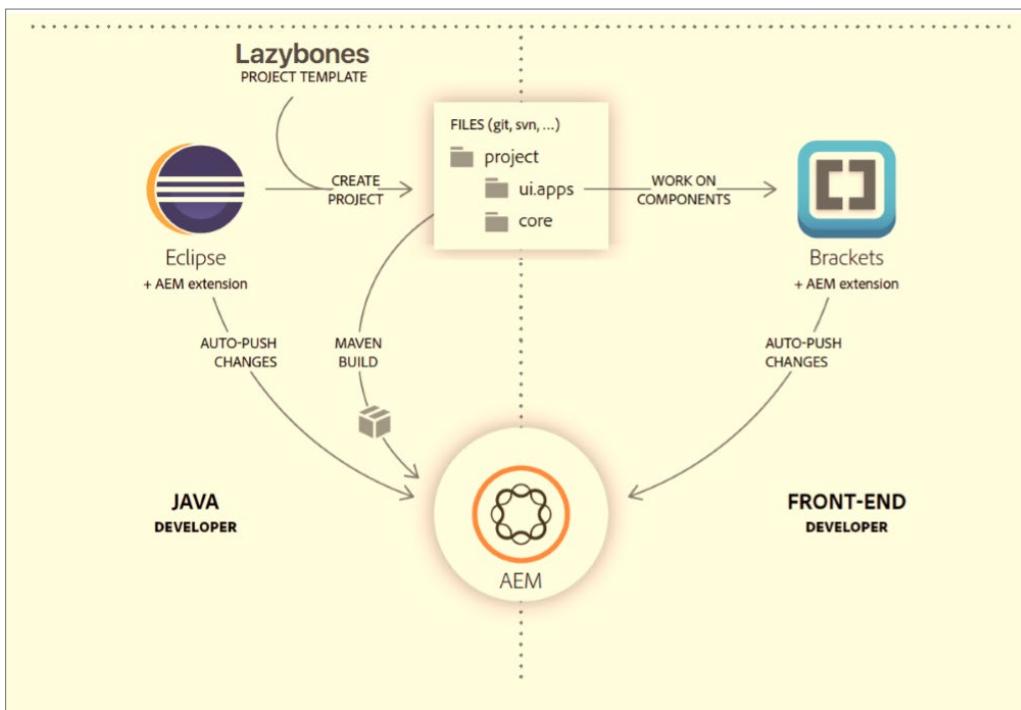
The benefits of building AEM projects in Maven are:

- An IDE-agnostic development environment
- Use of Maven Archetypes and Artifacts provided by Adobe
- Use of Apache Sling and Apache Felix tool sets for Maven-based development setups
- Easy integration with Continuous Integration Systems

Lazybones

Lazybones is a project creation tool that uses packaged project templates. You can use it to set up an AEM project.

The following diagram provides an overview of how an AEM project is created by using Lazybones:



The concept of Lazybones is very similar to Maven archetypes. You can use templates in Lazybones to create a new project structure for a framework or a library. Lazybones also includes a subtemplate feature that helps generate optional controllers and scaffolding inside a project. You can contribute templates by sending pull requests to the GitHub project or publishing the packages to the relevant Bintray repository (<https://github.com/Adobe-Consulting-Services/lazybones-aem-templates>).

Brackets

Brackets is an open source code editor for HTML, HTL, CSS, and JavaScript developed by Adobe.

The features of Brackets are:

- Live preview
- In-line editors
- Pre-processor support
- A collection of add-on extensions

The plugins available for AEM Developers are:

- AEM Brackets Extension
- Extract for Brackets

You can download the latest version of Brackets from the following link:

<http://brackets.io/>

Installing the AEM Brackets Extension

You can install the AEM Brackets extension within by Brackets using the Extension Manager. The AEM Brackets extension helps front-end developers build the components with HTL.

The AEM Brackets extension provides a smooth workflow to edit AEM components and client libraries and leverages the Brackets code editor to access Photoshop files and layers. The easy synchronization provided by the extension (no Maven or File Vault required) increases developer efficiency and helps front-end developers with limited AEM knowledge to work in projects. This extension also provides some support for the HTML Template Language (HTL), which makes component development easier and more secure.

The features of AEM Brackets extension are:

- Automated synchronization of changed files to the AEM development instance
- Manual bidirectional synchronization of files and folders
- Full content-package synchronization of the project
- HTL code completion for expressions and data-sly-* block statements

Configuring Your Project

You already learned how to create a package using the Package Manager. You can import the same package to Brackets and begin working on it. If you already have a project package, you must ensure it has a:

- jcr_root folder
- filter.xml file

To set up your Brackets project based on an existing front-end development project in your repository:

1. Generate a package of your project work in the CRX Package Manager.
2. Extract the contents of the package.
3. Open the jcr_root folder of the package.
4. Configure the project settings to connect to AEM development author instance.

Exercise 1: Install Brackets and the AEM Brackets extension

To install Brackets:

1. If you have an Internet connection, download the latest version of Brackets from: <http://brackets.io/>. If you do not have network connectivity, you can obtain the standalone installation files from the **Distribution_Files** folder for this course.



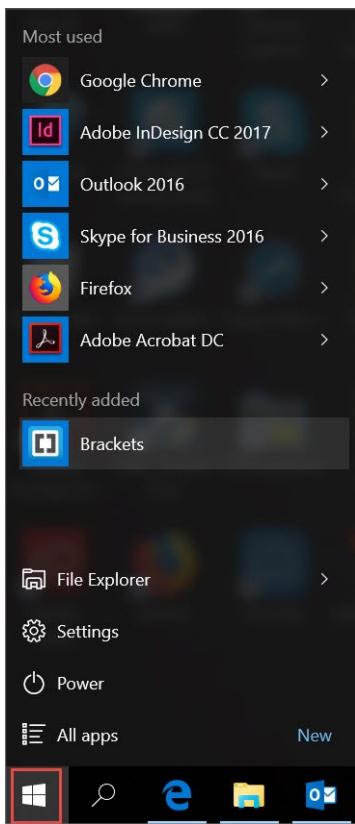
Note: The Distribution_Files folder is different than the Exercise_Files you are using for this course. Installers are available in the Distribution_Files for both Mac and Windows (Windows: Brackets.Release.1.12.msi, Mac: Brackets.Release.1.12.dmg).

2. Based on your Operating System (OS), double-click the **Brackets.Release.1.12.msi** or **Brackets.Release.1.12.dmg** installation file. The Brackets Installer dialog box opens.
3. Ensure both checkboxes are selected, and then click **Next**.
4. Click **Install**. The **Progress** bar appears. The installation may take 3-5 minutes to complete.
5. Click **Finish** to complete the installation.

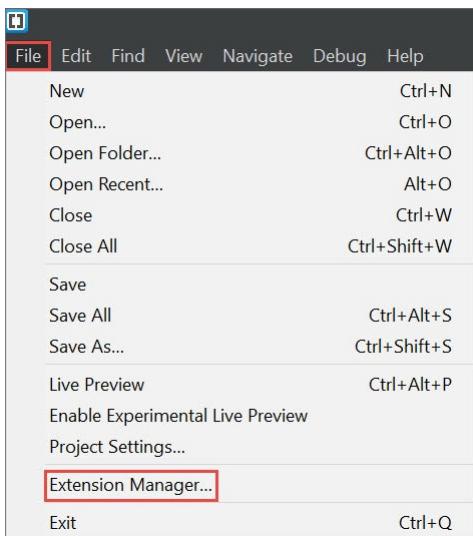


Note: If you are attending a VILT class using ReadyTech, the installation of Brackets has already been performed for you. You may skip ahead to step #2 of the Exercise 2.

6. Click the **Windows** logo on your task bar and click **Brackets** as shown. The **index.html (Getting Started) - Brackets** page opens.



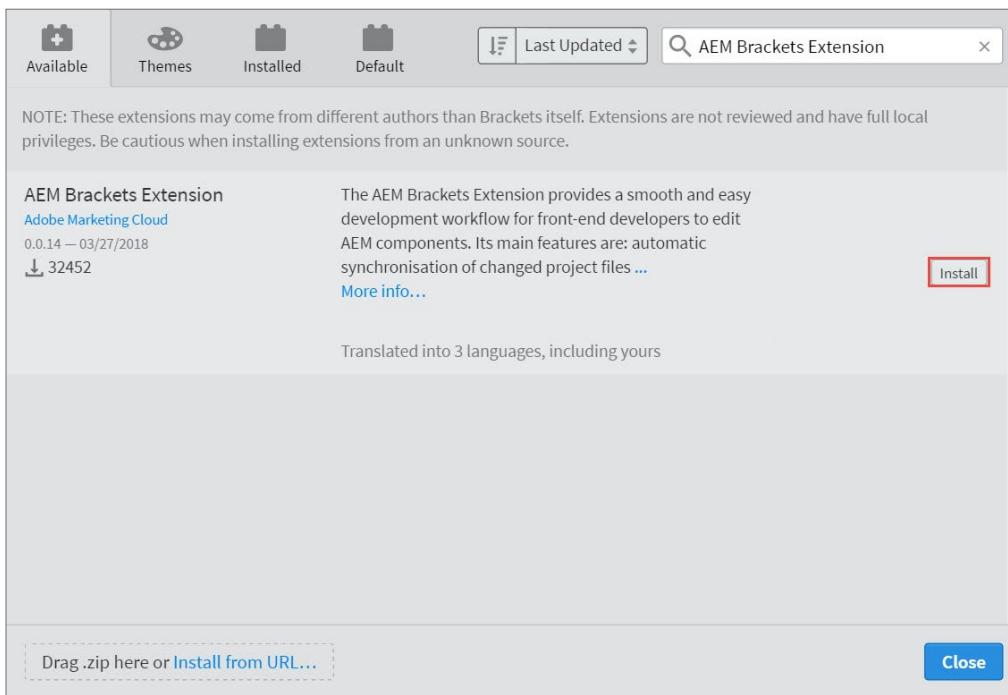
7. Click **File** and select **Extension Manager** from the list, as shown. A new wizard opens.



8. Ensure you are on the **Available** tab.

9. Enter **AEM Brackets Extension** in the **Search** field, and press Enter.

10. Click **Install** as shown. The **Install Extension** dialog box opens.



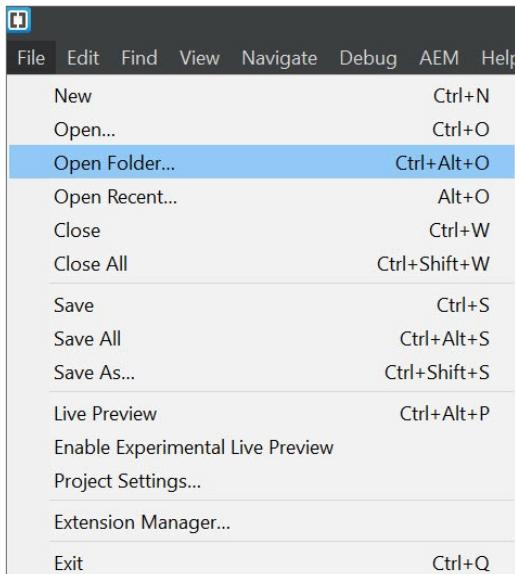
11. You can also drag the extension zip file (**aem-brackets-extension-master.zip**) from the **Distribution_Files** to the area displayed at the bottom of the pop-up window. Notice, you need to drag the file only if you do not have the internet connectivity.
12. Click **Close**.
13. Click X in the top-right corner to close the Brackets instance.

Exercise 2: Make changes to the repository by using Brackets

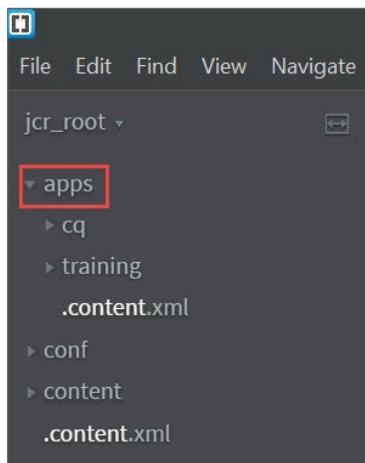
In this exercise, you will edit the contents of your project in Brackets. After completing this task, you can update the package content and share it with team members so they also see the same content in their AEM instances.

You will import the code content package into Brackets to observe how your repository can be configured in Brackets, and then examine the changes propagated to the repository in one of your We.Train pages.

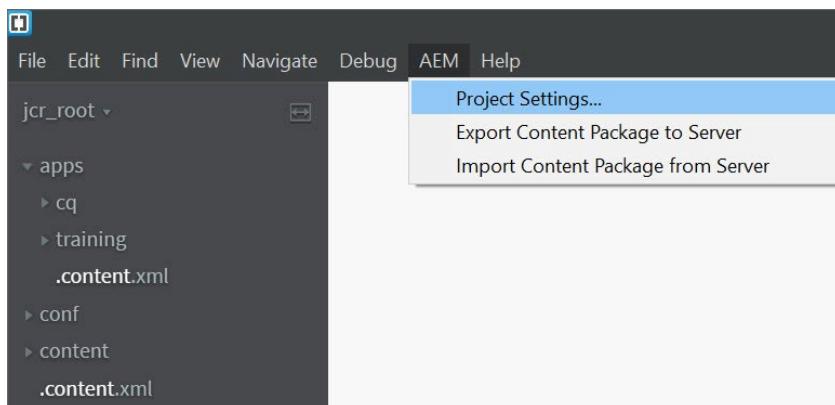
1. Navigate to the Downloads folder on your file system and look for the code content package (for example, chuckgrantDevWebsites-v1.0.zip) that you created in the previous exercise.
2. Right-click the **chuckgrantDevWebsites-v1.0.zip** and select **Extract to chuckgrantDevWebsites-v1.0.** to extract its contents to your local file directory.
3. Open the Brackets instance.
4. Click **File** and select **Open** from the list as shown. The **Choose a folder** dialog box opens.



5. Locate the **jcr_root** folder of the unzipped (extracted) package, and click **Select Folder** (or click **Open** on a Mac system). The contents of the folder are listed in the left panel.
6. Expand the **apps** folder as shown:



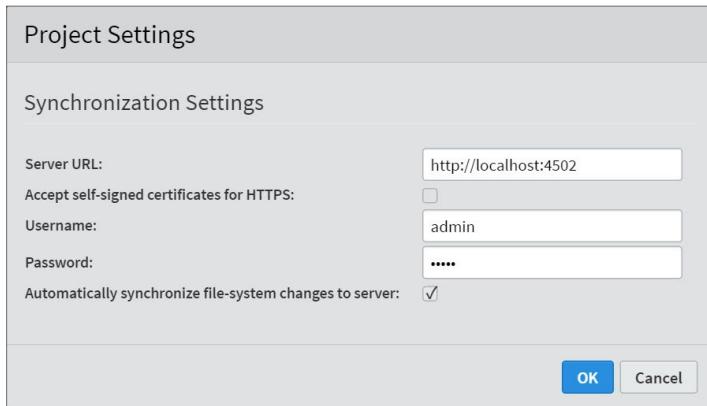
7. Click **AEM** and select **Project Settings** from the list as shown. The **Project Settings** dialog box opens.



8. Enter the following details:

Field	Value
ServerURL	http://localhost:4502
Username	admin
Password	admin

9. Select the **Automatically synchronize file-system changes to server** checkbox, and click **OK** as shown:



These details help Brackets to interact with the AEM author development instance. In a real-time scenario, your URL and credentials might differ.

10. In the left panel, expand the apps node and navigate to **/apps/training/components/structure/title/title.html**. The file opens on right panel.

11. Add a line of simple HTML: `<h2>Hello, I'm from Brackets!</h2>` at line 5 as shown:

```

File Edit Find View Navigate Debug AEM Help
apps/training/components/content/title/title.html (jcr_root) - Brackets
Working Files
• title.html
jcr_root
  ▾ apps
    ▾ cq
      ▾ training
        ▾ clientlibs
          ▾ components
            ▾ content
              ▾ breadcrumb
              ▾ image
              ▾ list
              ▾ stockplex
              ▾ text
              ▾ title
                ▾ _cq_dialog
                  .content.xml
title.html
1  <h1 data-sly-use.title="com.adobe.cq.wcm.core.components.models.Title"
2  data-sly-use.template="core/wcm/components/commons/v1/templates.html"
3  data-sly-element="${title.type}">
4  ${title.text}</h1>
5  <h2>Hello, I'm from Brackets!</h2>
6  <!--/* Subtitle added to HTL */-->
7  <h2 data-sly-test="${properties.subtitle}">${properties.subtitle}</h2>
8
9  <sly data-sly-call="${template.placeholder @ isEmpty=!text}"></sly>
10
Line 5, Column 39 — Selected 35 columns — 10 Lines
INS UTF-8 HTML Spaces: 4

```

12. Press **Ctrl+S** (Windows)/**Cmd+S** (Mac) or click **File** and select **Save** from the list.

13. Notice, the green AEM logo as shown. This confirms that the files are synchronized with AEM successfully.



To test the synchronization:

14. Navigate to the tab where the AEM author instance is running.
15. Navigate to the tab where the English page of We.Train site is open or open a new tab in your browser, add <http://localhost:4502/editor.html/content/we-train/en.html> in the address bar, and then press Enter. The **English** page opens.
16. Notice, the code that you added in Brackets will appear on the page as shown:

A screenshot of a web browser displaying the 'English' page. The page title is 'English'. Below it, there is a red-bordered box containing the text 'Hello, I'm from Brackets!'. Further down, there are two input fields labeled 'Image' and 'List'. At the bottom of the page, the text 'ADBE' is displayed in red, followed by 'Current Value: No Data'.

References

You can use the following links for more information on:

- How to Develop AEM Projects Using Eclipse:

<https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/howto-projects-eclipse.html>

- How to Build AEM Projects using Apache Maven:

<https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/ht-projects-maven.html>

- Creating an Adobe Experience Manager project using Lazybones:

https://helpx.adobe.com/experience-manager/using/aem_lazybones.html

- Creating an Adobe Experience Manager project using Lazybones:

https://helpx.adobe.com/experience-manager/using/aem_lazybones.html

- AEM Developer Tools for Eclipse:

<https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/aem-eclipse.html>

Appendix

In this section, you will learn about static templates, dialog coversion tool, and AEM and General Data Protection Regulation (GDPR) Compliance.

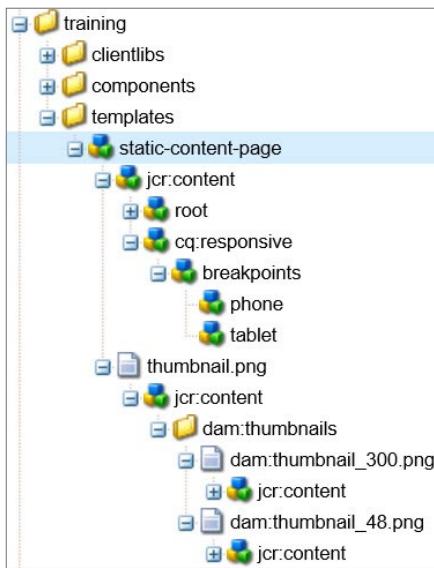
Static Templates

A static template is a hierarchy of nodes that has the same structure as the page to be created, but without any actual content.

You can create Web pages based on your requirement using static templates and achieve the same output as editable templates that you created in previous course - Creating Editable Templates and Pages in AEM.

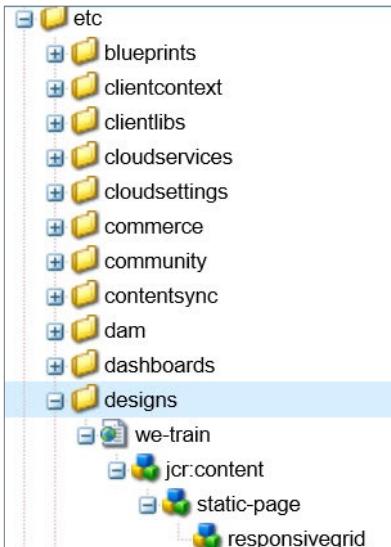
To understand how the static templates work, you need to install **we-train-static-template.zip** on your AEM instance. The package contains static template for We.Train site that was created for you.

The static templates typically exist under `/apps/<my-app>/templates` folder, where as the editable templates exist under `/conf/<tenant>`. A page component for a static template must include all structure components that are hard coded in the HTL as shown in the below screenshot. If you have installed the package, open CRXDE Lite, and navigate to the `/apps/training/templates/static-content-page` node and observe the changes.



On the other hand, the page component for an editable template, typically has a layout container to allow template authors drag and drop the structure components onto the template

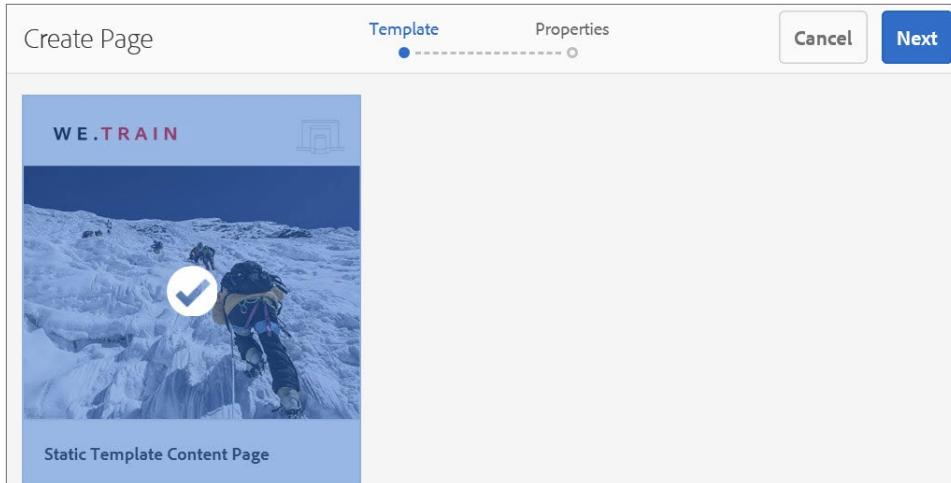
Any component configurations (Design Mode) for static templates exist under `/etc/designs/<my-design>` as shown in the below screenshot. Any component configurations (Content Policies) for editable templates exist under `/conf/<tenant>/settings/wcm/policies`.



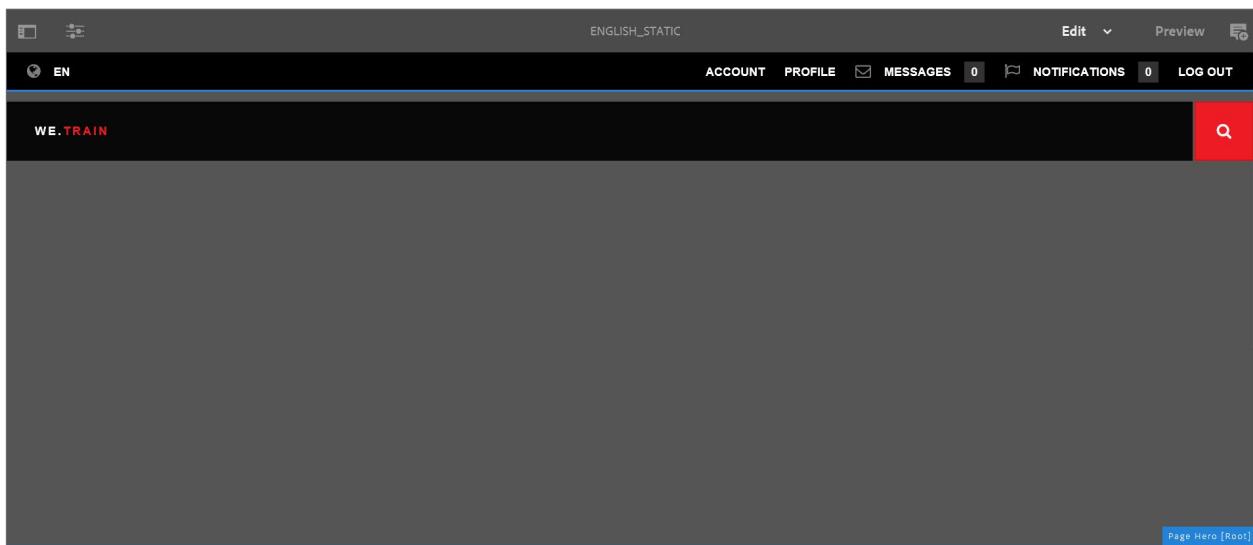
To use the static template for creating pages, you need to add the following properties to **We.Train** from the Sites console.

- Design: /etc/designs/we-train
- Allowed Templates: /apps/training/templates/*

After, defining the above properties, when creating a new page in **We.Train** site, the **Static Template Content Page** appears in the **Create Page** dialog box as shown:



The page created from Static Template Content Page has the same structure as the page created from editable template as shown:



Dialog Conversion Tool

The dialog conversion tool is provided to convert Classic UI dialogs or dialogs based on Coral 2 to Coral 3. The tool uses the original dialog to create a duplicate dialog designed for the Touch UI, based on Granite UI and Coral 3.

Although the Dialog Conversion tool will create a new dialog, it will skip what it cannot convert. Therefore, the resulting dialog might contain nodes from the original dialog copied as-is, if no rule matched that specific component. In addition, a converted component might have some unconverted properties, because there was no appropriate rule to convert them.

The tool cannot cover every scenario, as its conversion rules are non-exhaustive and operate on a best-effort basis. It converts the most frequently used elements and properties, but the conversion will be incomplete when dealing with customizations or highly-specialized dialogs. Converted dialogs may require additional adjustments and all conversions must be reviewed.

You can use the following links for more information on:

- Dialog Conversion Tool:

<https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/dialog-conversion.html>

AEM and GDPR Compliance

The European Union's General Data Protection Regulation on data privacy rights takes effect as of May 2018:

"The EU General Data Protection Regulation (GDPR) replaces the Data Protection Directive 95/46/EC and was designed to harmonize data privacy laws across Europe, to protect and empower all EU citizens data privacy and to reshape the way organizations across the region approach data privacy."

The regulation will apply to any company doing business with individuals in the EU. Adobe recognizes that this presents a new opportunity for companies to strengthen their brand loyalty by focusing on consumer privacy while delivering amazing experiences.

AEM instances and the custom applications that might process Personally Identifiable Information (PII) data are owned and operated by AEM customers. This means that the Data Processor and Data Controller as defined in GDPR are both owned and managed by the AEM customer, so AEM 6.4 does not include any out-of-the-box service to handle GDPR requests. You can use the following links for more information on:

- GDPR Compliance:

<https://helpx.adobe.com/experience-manager/6-4/managing/using/gdpr-compliance.html>

- AEM Foundation GDPR support:

<https://helpx.adobe.com/experience-manager/6-4/sites/administering/using/handling-gdpr-requests-for-aem-platform.html>