Develop Websites and Components in Adobe Experience Manager

STUDENT WORKBOOK

ADOBE DIGITAL LEARNING SERVICES

Develop Websites and Components in AEM

04302020

Playlist

Develop Websites and Components in Adobe Experience Manager

This course is a combination of modules from two different courses. You will receive two student guides and two exercise folders, one for each course:

- Adobe Experience Manager Technical Basics
- Develop Websites and Components in Adobe Experience Manager

This table shows the order in which these courses, and the modules within them, flow:

Course Name	Module Number	Module Name
AEM Technical Basics	1	AEM Architecture Overview
	2	AEM Installation
	3	Developer Tools
	4	AEM Sites Authoring Basics
Develop Websites and Components in Adobe Experience Manager	All modules	1. Introduction to Content Rendering 2. Introduction to HTML Template Language 3. AEM Sites Development Key Concepts 4. Creating Editable Templates and Pages 5. Creating Client-Side Libraries 6. Core Components: An Overview 7. Working with Components 8. Creating Custom Components 9. Preparing for Production
AEM Technical Basics	10	Front-End Development Using aemfed
	11	Cloud Manager Basics (Cloud only)

Contents

Introduction to Content Rendering

Introduction

Adobe Experience Manager (AEM) is a content management system that helps organizations build custom websites and apps across different customer touch points. To better utilize AEM features, you should understand the key concepts such as Java Content Repository (JCR), base page component, Sling resolution, and inheritance process.

Objectives

After completing this module, you will be able to:

- Explain JCR nodes, properties, and namespaces
- Explain the JCR folder structure
- Create a project folder structure in the JCR
- Describe the structure of a component
- Create a component
- Describe dialogs
- Create an edit dialog
- Explain the Sling resolution process
- Search for a rendering script
- Customize selectors
- Explain how Sling inheritance works
- Implement Sling inheritance through resourceType
- Modularize content by including other scripts

JCR Nodes, Properties, and Namespaces

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

Nodes

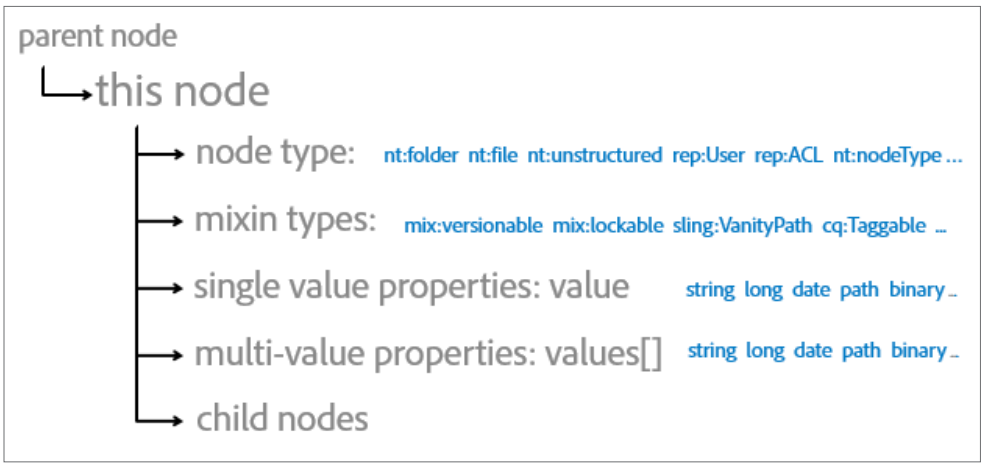
Nodes provide the structure and properties to store the data. The nodes of JCR can have:

- Parent and child nodes that are represented with paths.
- A type that sets the rules about the kind of properties and child nodes that 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
- Any number of properties.
- Zero or more mixin types that specify additional properties and child nodes that 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 a value
- Single or multi-valued property. A multi-valued property is notated and behaves like an array (values []).



Namespaces

All nodes and JCR properties 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	Supports any combination of child nodes and properties. It also supports client-orderable child nodes, stores unstructured content, and is 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 node of cq:EditConfig.

JCR Folder Structure

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

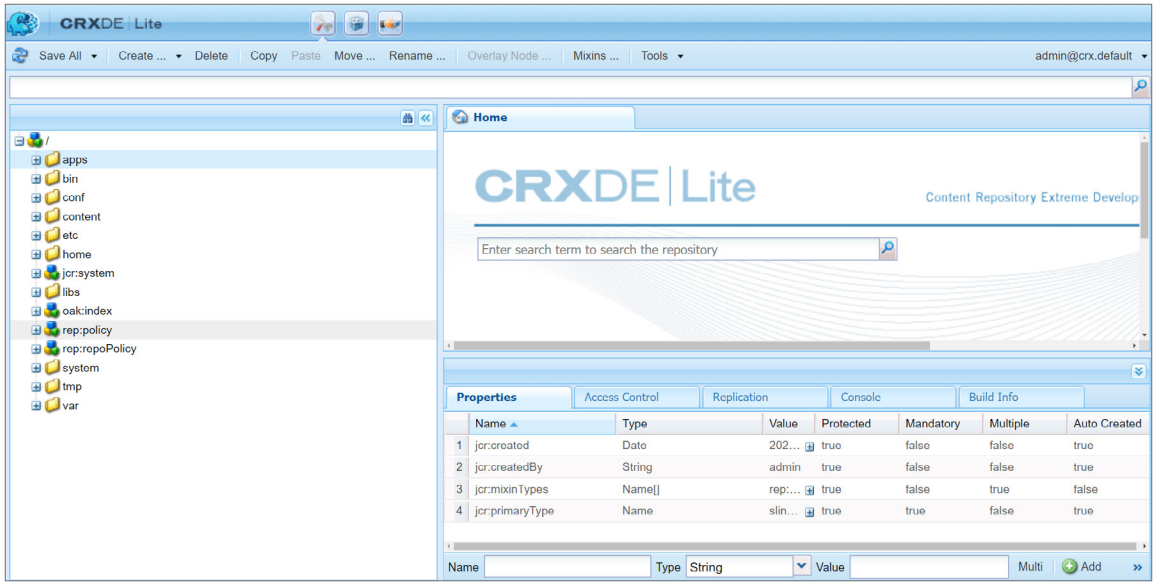
Folder	Description
/apps	Contains all project codes such as components, overlays, client libraries, bundles, i18n translations, and static templates created by an organization.
/conf	Contains all configurations for your website. This folder is used to store the dynamic templates and policies for your 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 the files that 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 are 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. Most changes will occur in /apps during development.

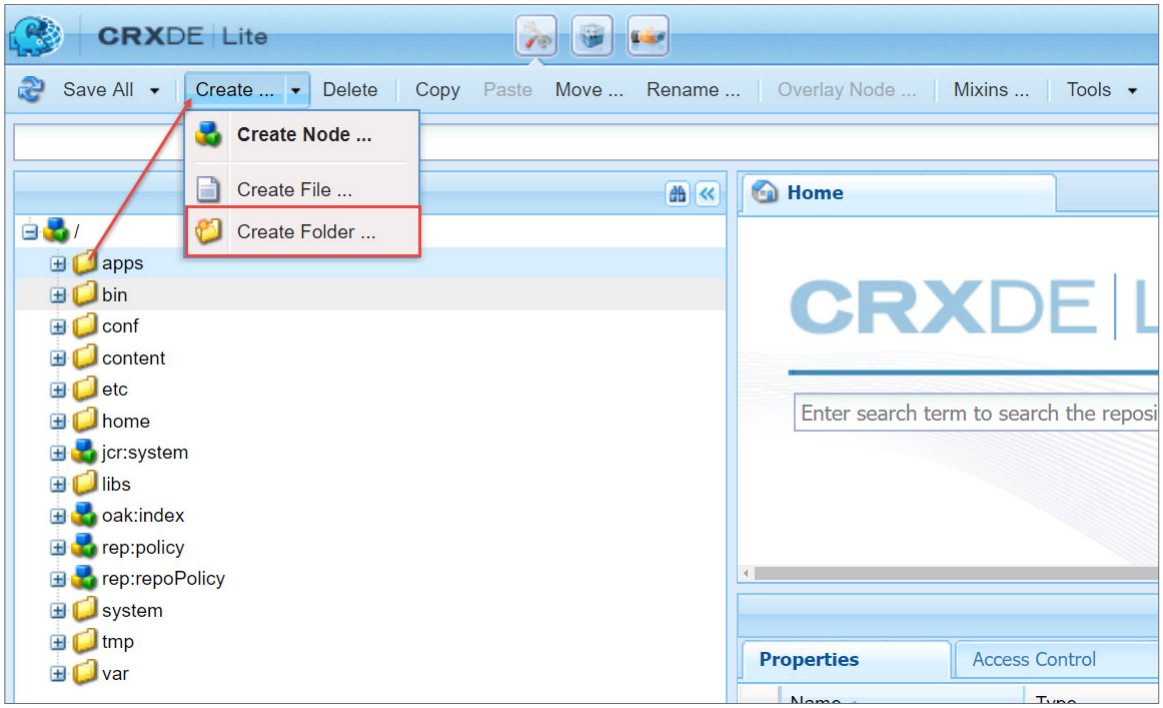
Exercise 1: Create a project structure in JCR

To create a project structure in JCR:

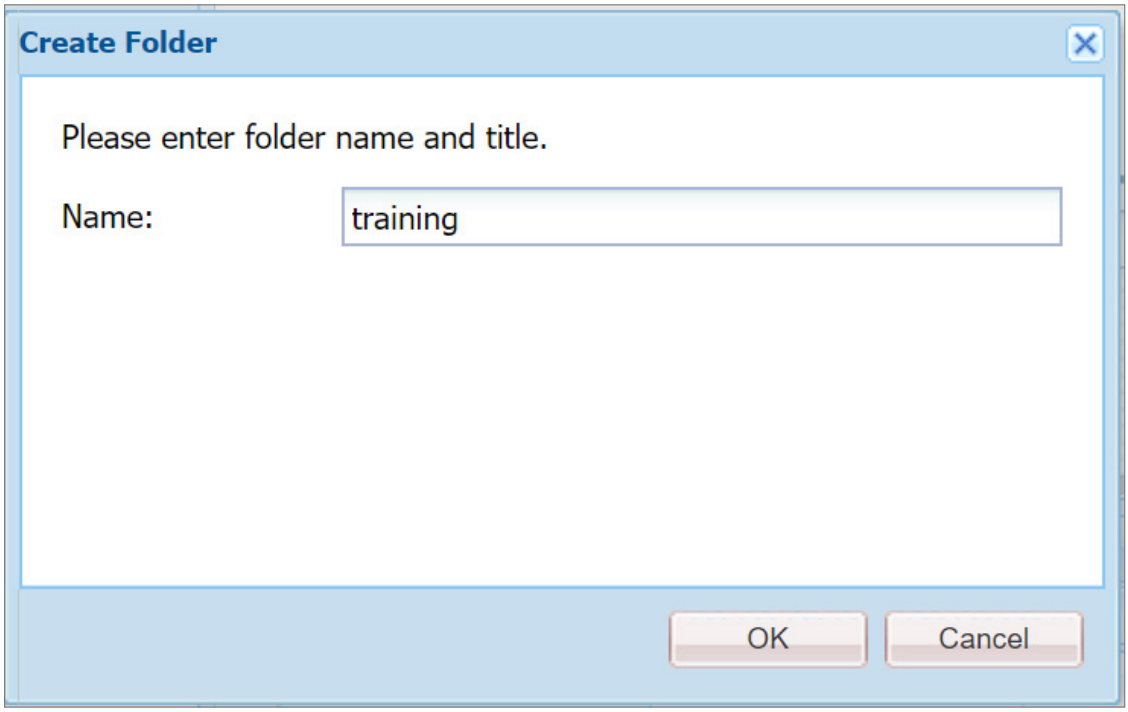
1. In the AEM author service, click **Adobe Experience Manager** from the header bar and click the **Tools** icon. The **Tools console** opens.
2. Click **CRXDE Lite**. The **CRXDE Lite** page opens on a new browser tab, as shown. (You can also open http://localhost:4502/crx/de in a browser to access the **CRXDE Lite** page.)



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

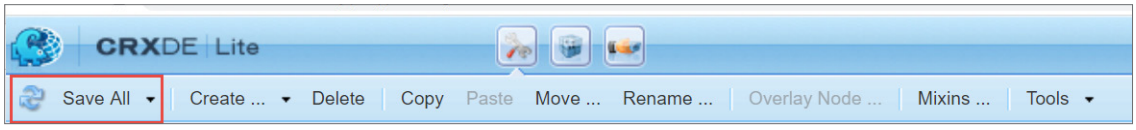


4. Enter **training** in the **Name** box and click **OK**, as shown. The training folder is created under the /apps folder.



Note: Node names are a restricted set. In node names, ensure there are no whitespaces or special characters, and use 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 corner of the actions bar, as shown:

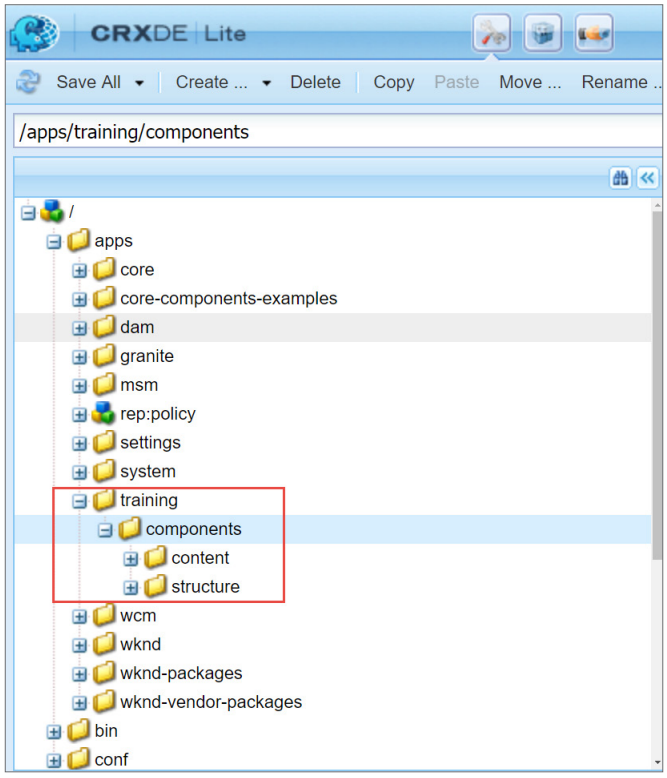


6. Under **/apps**, select the **training** folder you created now.
7. Click **Create Folder** from the **Create** drop-down menu on the actions bar to create a new folder. The **Create Folder** dialog box opens.
8. Enter **components** in the **Name** box and click **OK**. The components folder is created under the training folder.
9. Click **Save All**.

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

10. Navigate to the **/apps/training** folder, select the **components** folder, and create two child folders named **content** and **structure**. The two folders are created under the components folder.

11. Click **Save All**. Your project structure should look similar to the one shown in the below screenshot:



Structure of a Component

The important elements of the component structure are:

- Root node
- Vital properties
- Vital child nodes

Root Node

It is the hierarchy node of the component and is 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 tooltip in the components browser.
- cq:icon: Is the string property pointing to a standard icon in the Coral UI library, which is displayed in the component browser.

Vital Child Nodes

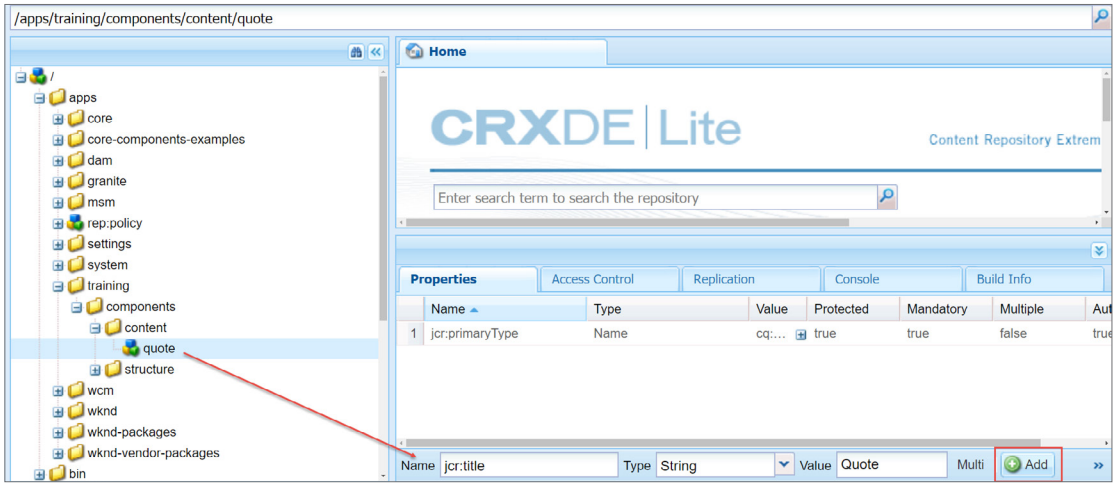
The vital child nodes of a component are:

- 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 a component and defines the interface, which enables the user to configure the component and/or edit content.
- cq:design_dialog (nt:unstructured): Helps edit the design of a component.

Exercise 2: Create an HTL component

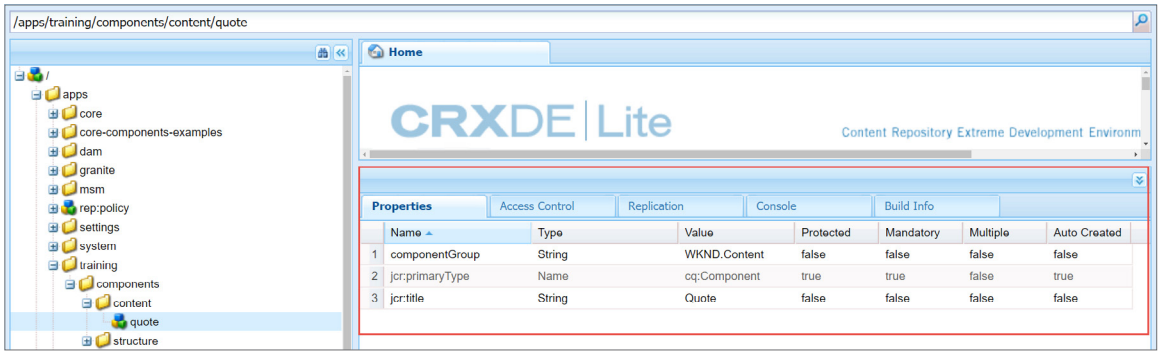
In this exercise, you will create a simple component that will be added to a page on the WKND website.

1. In **CRXDE Lite**, navigate to **/apps/training/components/content**.
2. Right-click the **content** folder and click **Create** > **Create Node**. The **Create Node** dialog box opens.
3. In the **Name** box, type **quote**.
4. From the **Type** drop-down menu, select **cq:Component**.
5. Click **OK**. A node of component type is created.
6. Click **Save All** to save the changes.
7. Select the quote component you created to add properties:
   a. In the **Name** box, type **jcr:title**.
   b. In the **Type** box, retain **String**.
   c. In the **Value** box, type **Quote**.
   d. Click **Add**, as shown. The property is added.



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

9. Repeat steps 7 and 8 to add another property:
   a. **Name**: **componentGroup**
   b. **Type**: **String**.
   c. **Value**: **WKND.Content**.
   d. Click **Add**. The property is added.
10. Click **Save All** to save the changes. The quote component properties should look, as shown:

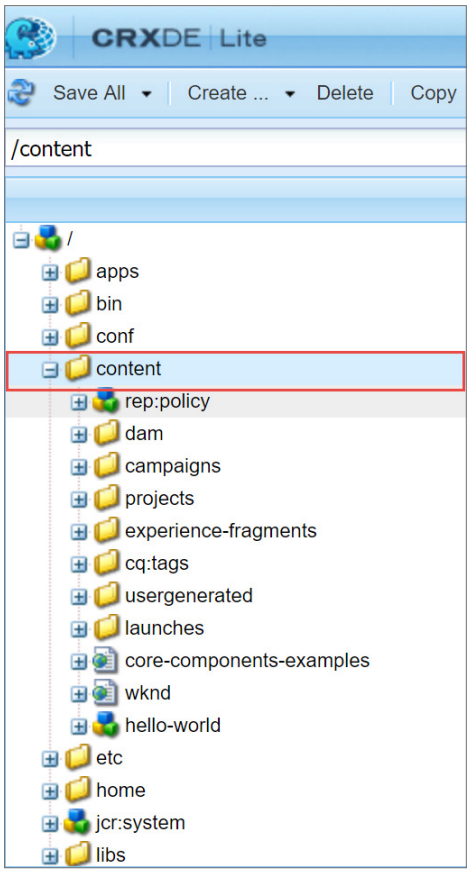


11. Right-click the **quote** component and click **Create** > **Create File**. The **Create File** dialog box opens.
12. In the **Name** box, type **quote.html** and click **OK**. The file is created.
13. Click **Save All** to save the changes. Notice that the **quote.html** editor opens on a separate tab on the right in **CRXDE Lite**.
14. In the Exercise Files folder provided to you by your instructor, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/content/quote** folder.
15. Open the **start-quote.html** file in Notepad++ (or any editor of your choice), copy the content of the file and paste it in the **quote.html** editor in **CRXDE Lite**.
16. Click **Save All** to save the changes.

Exercise 3: Create content to render the component

In this exercise, you will create a content node that will render the 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. You saw an example of this with the quote component and the WKND website. Use CRXDE Lite to quickly create a testing content node.

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

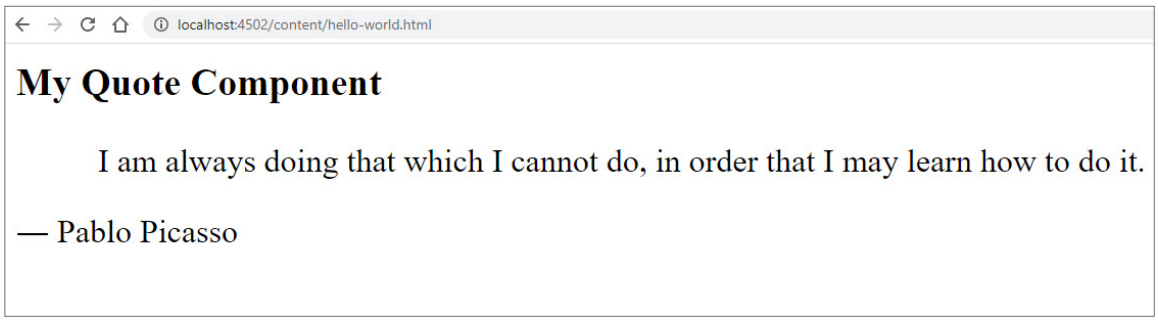


Note: Select the /content folder that is at the root, as shown above, and not the one within your /training project structure.

2. Right-click the **/content** folder and click **Create** > **Create Node**. The **Create Node** dialog box opens.
3. Enter the following details:
   a. **Name**: **hello-world**
   b. **Type**: **nt:unstructured**
4. Click **OK**. The **hello-world** node is created.
5. Click **Save All**.
6. Select the **hello-world** node and add the following property on the **Properties** tab:

Name	Type	Value
sling:resourceType	String	training/components/content/quote

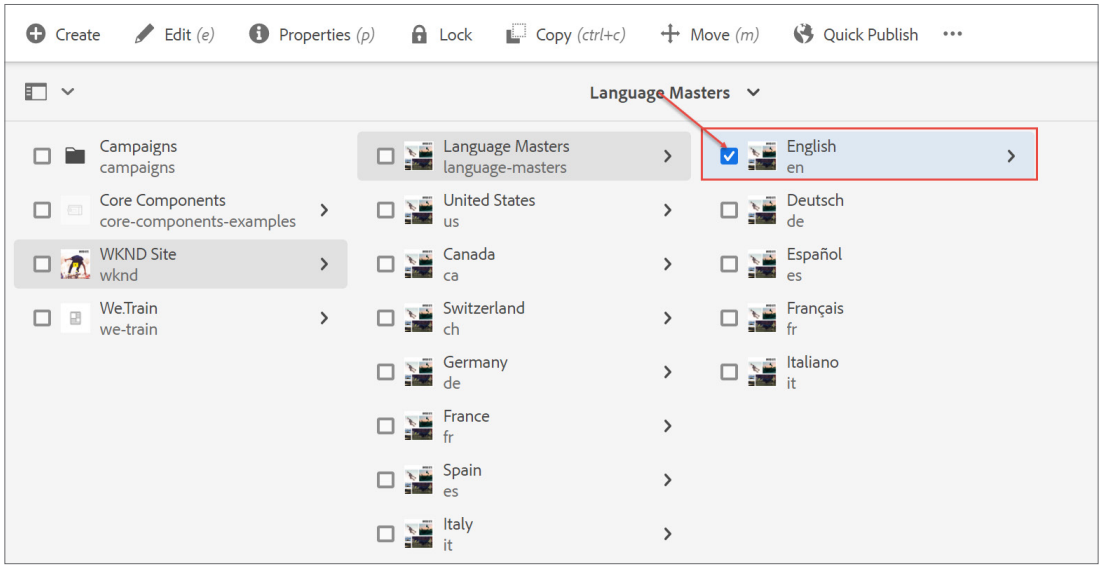
7. Click **Add**. The property is added to the node.
8. Click **Save All**.
9. In a browser, open http://localhost:4502/content/hello-world.html.
10. Observe how the property `sling:resourceType` renders the content from **quote.html** on the hello-world.html page:



Exercise 4: Use the authoring interface to add content

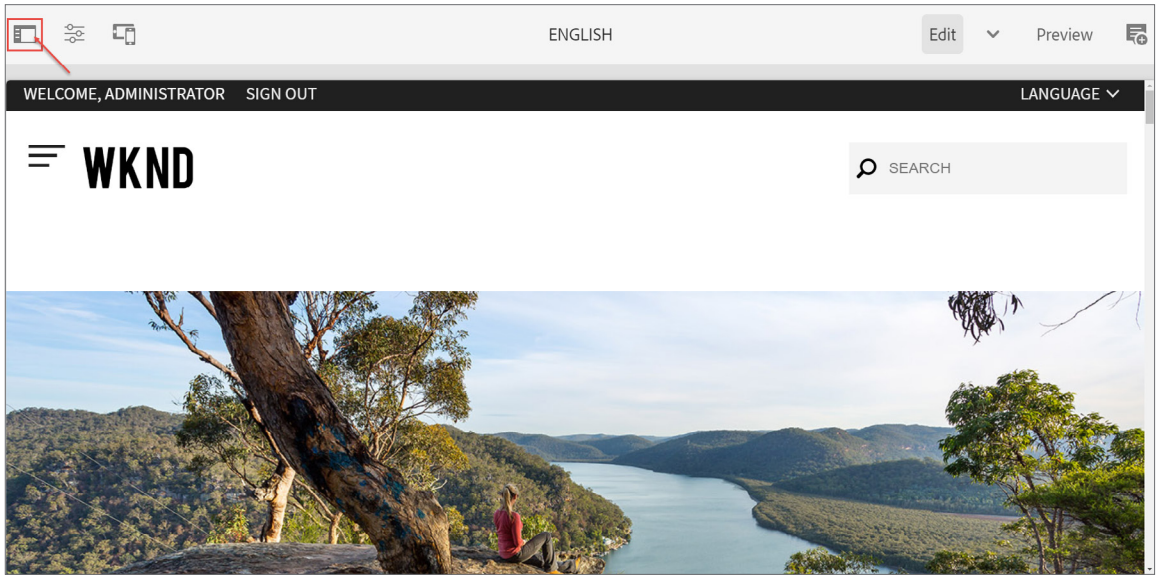
To drag a component onto a page, the page must have a dialog box—even if the box is empty. In this exercise, you will create an empty dialog to test the quote component on a page.

1. In **CRXDE Lite**, right-click the **quote** component and click **Create** > **Create Node**. The **Create Node** dialog box opens.
2. In the **Name** box, type **cq:dialog**.
3. From the **Type** drop-down menu, select **nt:unstructured** if it is not selected already.
4. Click **OK**. The node is created
5. Click **Save All** to save the changes.
6. In the AEM author service, navigate to **Sites** > **WKND Site** > **Language Masters**.
7. Select the **English** page by clicking the check box, as shown:

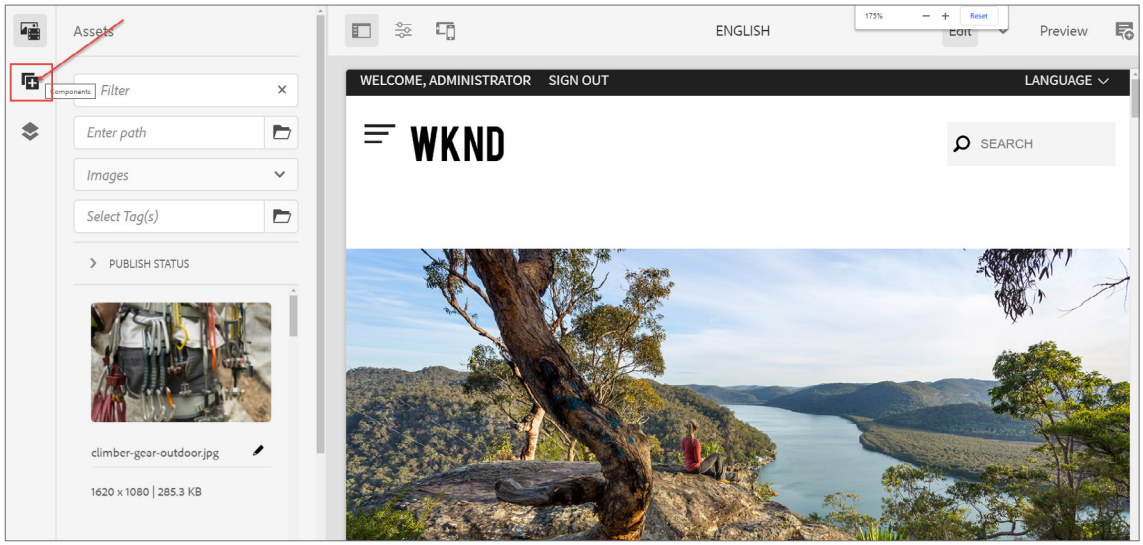


8. With the English page selected, click **Edit (*e*)** from the top menu bar to open the **English** page in **Edit** mode.

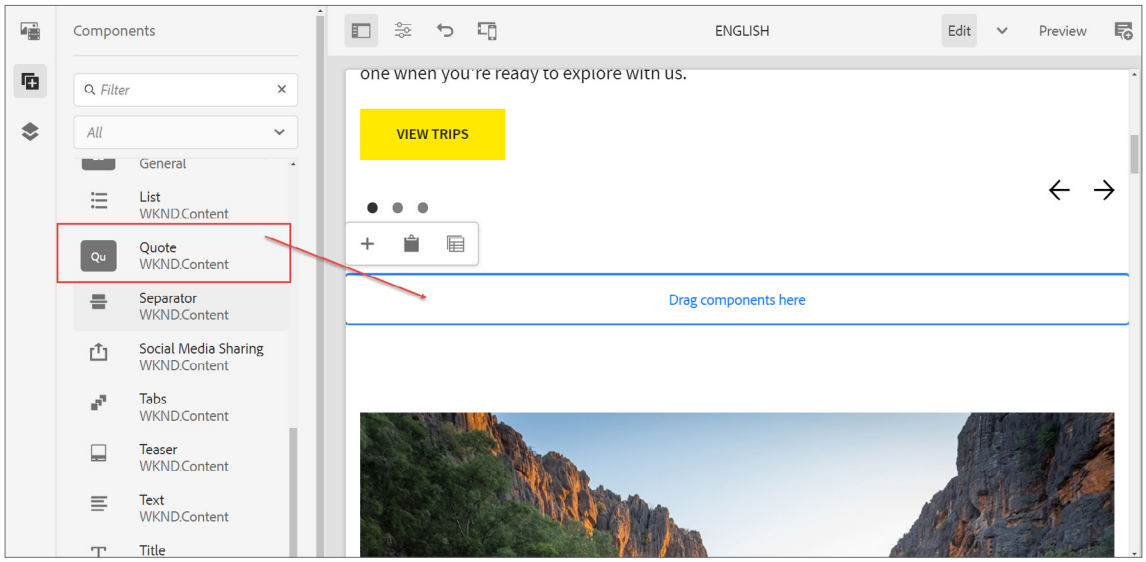
9. On the upper left, click the **Toggle Side Panel** icon, as shown, if you do not see the left panel.



10. On the left panel, click the **Components** icon, as shown. A list of available components is displayed.

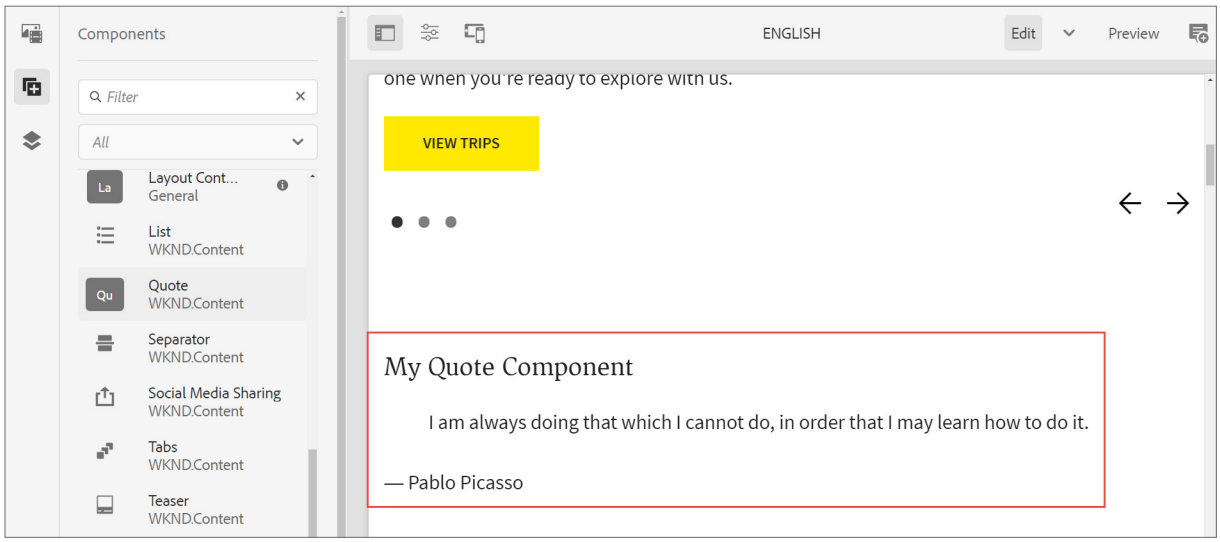


11. Look for the **Quote** component and drag the component, as shown, onto the layout container.



Note: Even though the component was created in the /apps/training code base, you will still be able to see and use the component on the WKND website. This is because the componentGroup property is WKND.Content on the /apps/training/components/content/quote node.

Your page should look similar to the screenshot below:



Dialogs

Dialogs 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 tab keeps the dialog short and sorts the input fields.

Coral UI and Granite UI

The Coral UI and the Granite UI define the look and functionality of AEM. The Granite UI provides a large range of the basic components (widgets) needed to create a 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.

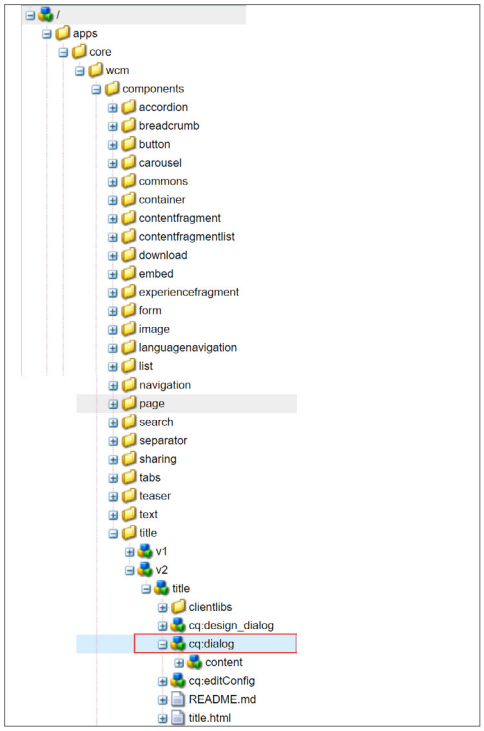
The cq:dialog node:

- Defines the dialog for editing the content of the 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.

Exercise 5: Create an edit dialog

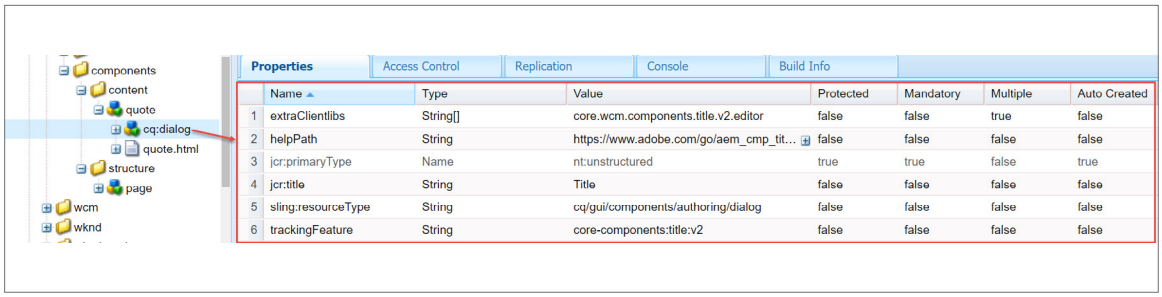
In this exercise, you will create a dialog box that enables an author to add a value to a form field and then have that value display on the component.

1. In **CRXDE Lite**, navigate to **/apps/training/components/content/quote/cq:dialog**.
2. Right-click the **cq:dialog** node you created in the previous exercise and click **Delete**. The node is deleted. In this exercise, you will copy an existing dialog from a core component and modify it as per your requirement.
3. Click **Save All** to save the changes.
4. Navigate to **/apps/core/wcm/components/title/v2/title/cq:dialog**, as shown:



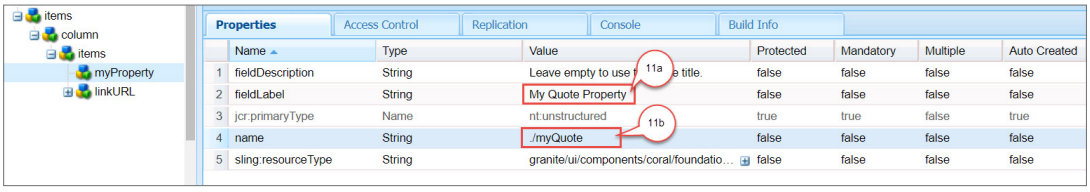
5. Right-click the **cq:dialog** node and click **Copy**. The node is copied.
6. Navigate to **/apps/training/components/content/quote**.
7. Right-click the folder and click **Paste** to paste the node under the quote folder.

8. Select the **cq:dialog** node. You will see the properties of the **cq:dialog** node on the **Properties** tab on the right, as shown:



You must change the dialog title as this is a different component

9. On the **Properties** tab, double-click **jcr:title** and change the value to **Quote**.
10. Under **quote/cq:dialog/content/items/tabs/items/properties/items/columns/items/column/items**:
    a. Right-click the **types** node and click **Delete** to delete the node.
    b. Right-click the **defaulttypes** node and click **Delete** to delete the node.
    c. Right-click the **title** node and click **Rename** to rename it.
    d. Rename it as **myProperty**.
    e. Click **Save All** to save the changes.
11. Select the **myProperty** node to update its properties:
    a. **fieldLabel**: **My Quote Property**
    b. **name**: **./myQuote**
    c. Click **Save All** to save the changes. The Properties tab should look, as shown:

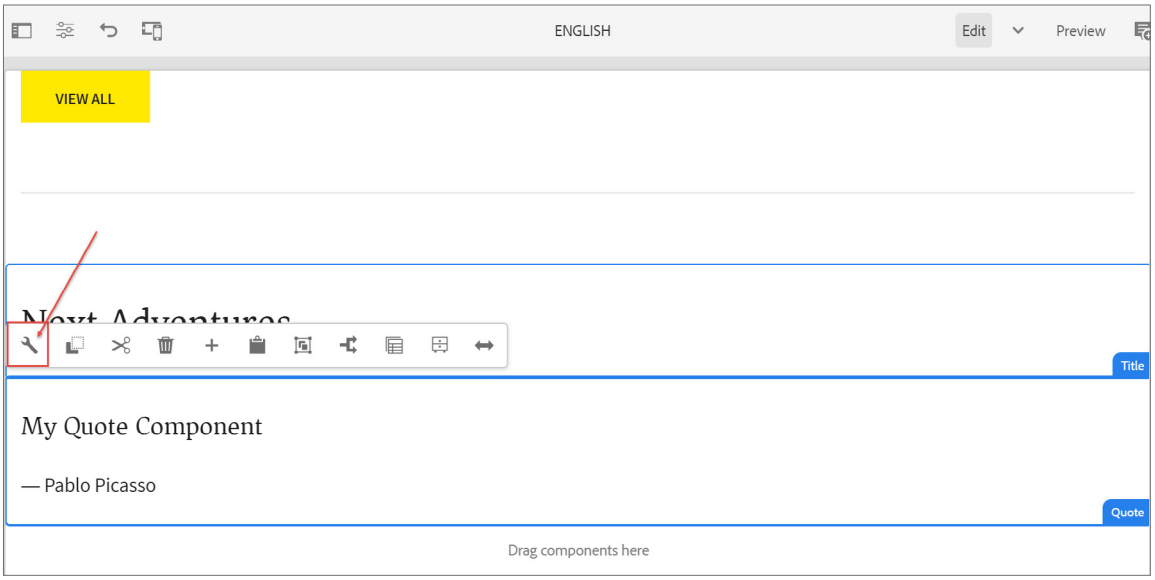


When an author uses this dialog and adds a value to the new formfield, AEM saves it under the current node on a property called myQuote. This value can then be accessed by the HTL script in the quote component.

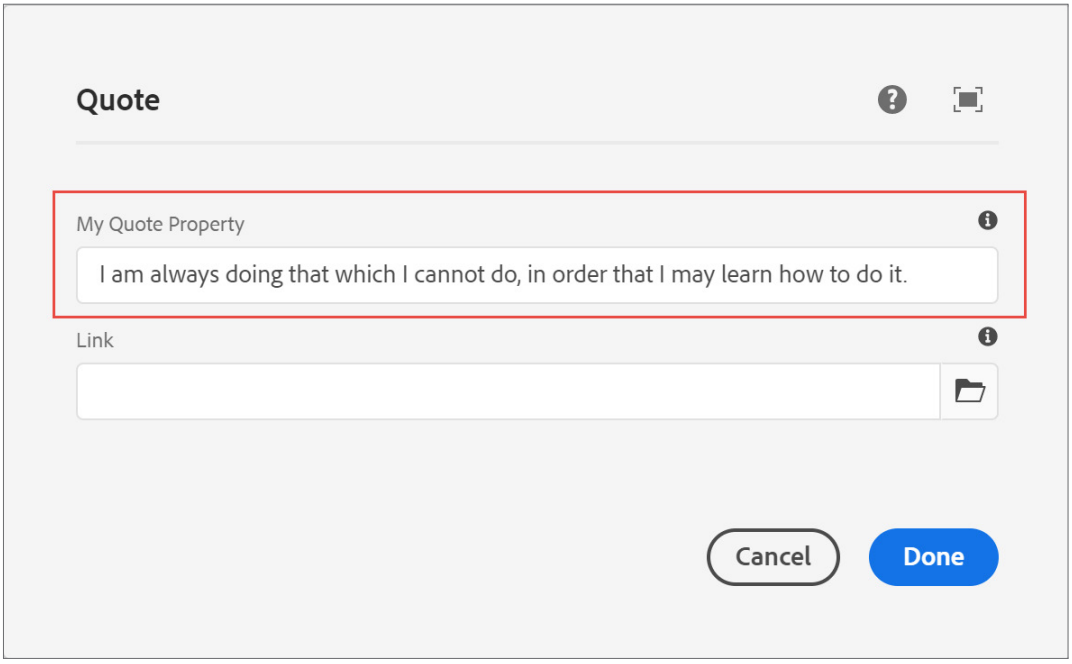
12. Double-click **quote.html**. The editor opens on the right. Delete its content.
13. In the Exercise Files folder provided to you by your instructor, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/content/quote** folder.
14. Open the **dialog-quote.html** file in Notepad++ (or any editor of your choice), copy the content of the file, and replace the existing content in the **quote.html** editor in **CRXDE Lite** with the new content.

Notice the code `${properties.myQuote}`. By using `${}` expression, you are able to insert an object into the HTML code. The properties is a global object that gives access to all properties on the current resource.

15. Click **Save All** to save the changes.
16. In the AEM author service, navigate to **Sites** > **WKND Site** > **Language Masters** > **English**.
17. With the English page selected, click **Edit (*e*)** from the top menu bar to open the English page. on a new tab.
18. Click the **Quote** component on the page and click the **Configure** icon, as shown. The **Quote dialog box** opens. If your page does not have a Quote component, drag it on the page.

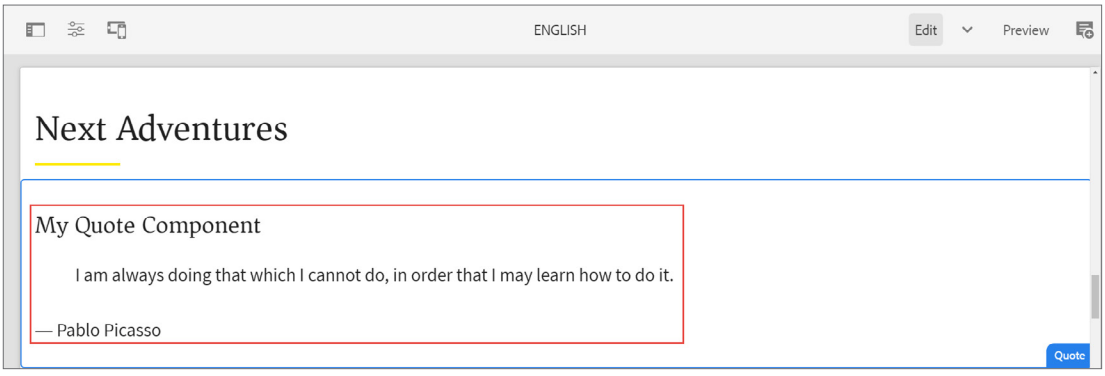


19. Add the following value to the dialog box:
    a. **My Quote Property**: I am always doing that which I cannot do, in order that I may learn how to do it.



20. Click **Done**.

21. Observe the rendered dialog values on the page.



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 the resource to a file system or a 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 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

Processing Requests: Steps

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:

- The properties of the content item
- The HTTP method used to make the request
- The simple naming convention within the URL that provides the secondary information

The steps involved in resolving a URL request are:

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: Is used to specify additional information
- Param(s): Are any parameters required for dynamic content

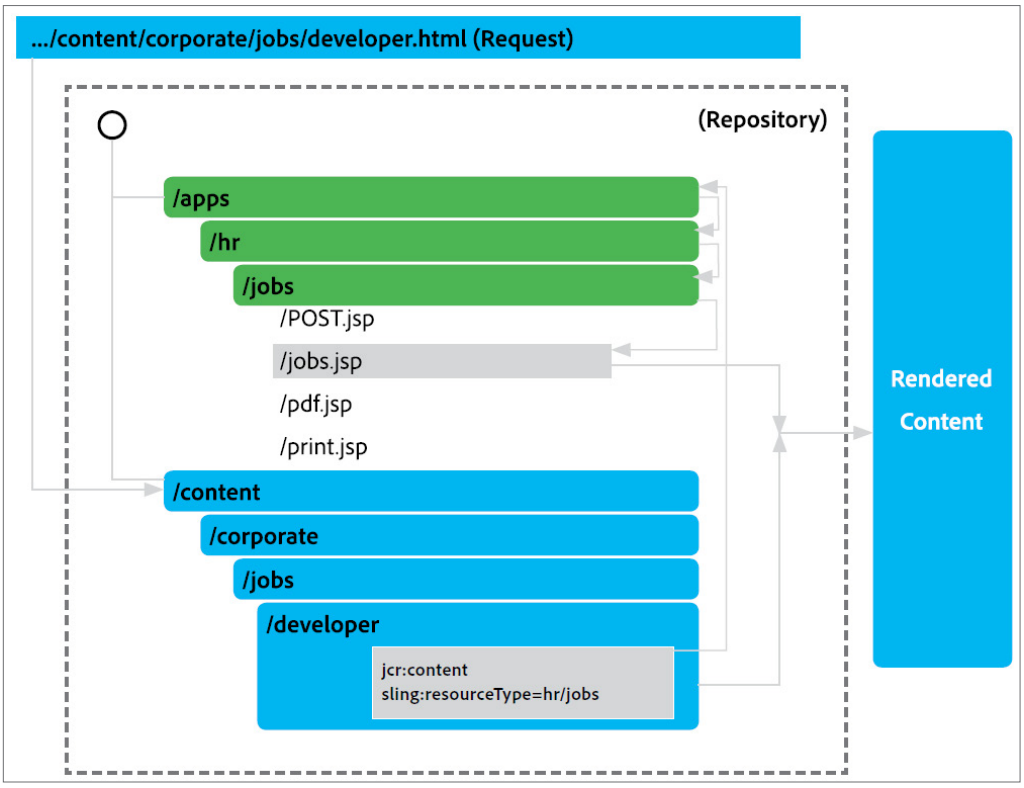
Resolving Requests to Resources

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, 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.

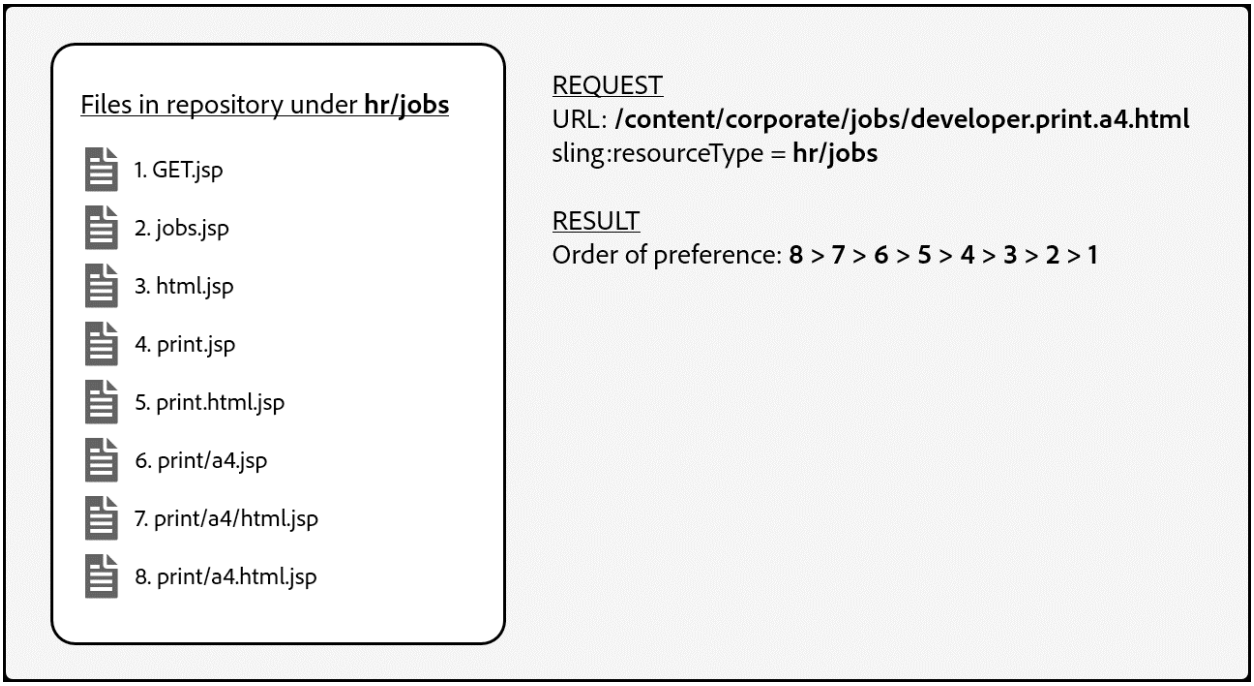
4. If no node is found, Sling will return the http code 404 (Not Found).

Example 2: Consider the URL request, http://myhost/content/corporate/jobs/developer.html and the corresponding diagram. Notice how the properties of the developer node are used to provide the rendered content:



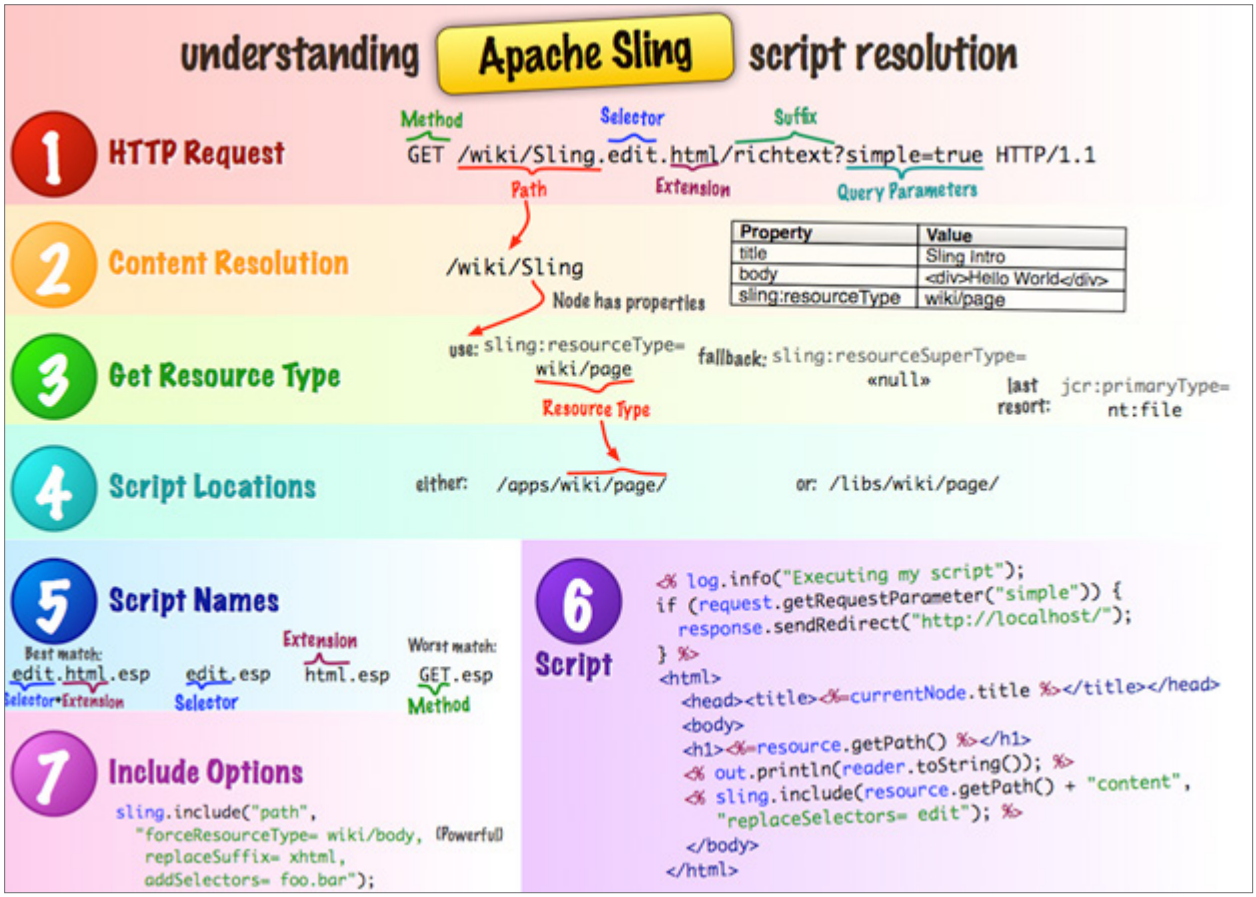
Locating and Rendering Scripts

When the resource is identified from the URL, its resource type property is located, and the value is extracted. This value is either an absolute or a relative path that points to the location of the script to be used for rendering the content. 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, 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:



URL Decomposition

Each content item in the JCR is exposed as an HTTP resource, and the request URL addresses the data to be processed. After the content is determined, the script or the servlet to be used to handle the request is determined. In the 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.



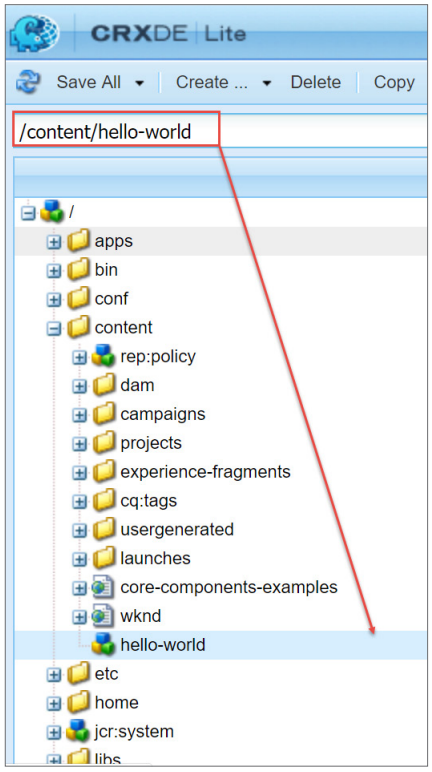
Exercise 6: Search for a rendering script

When presented with a request (URL), Sling will perform 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.

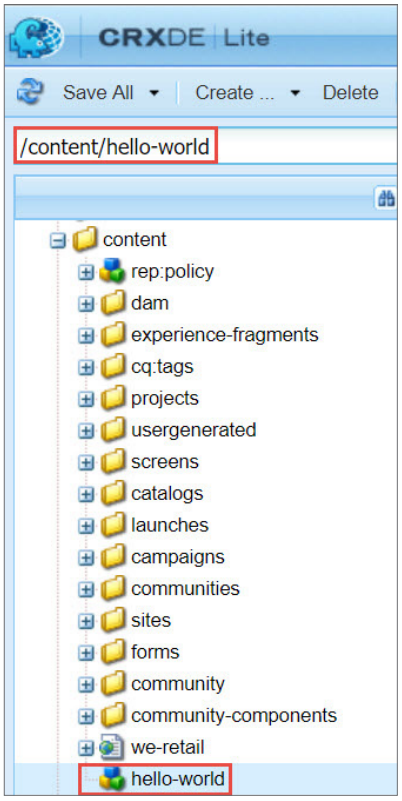
In this exercise, you will follow how Sling resolves a URL into resources and scripts to render. Now, let us follow the Sling's actions to see how the hello-world resource was rendered in the previous exercise.

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, click http://localhost:4502/crx/de to open the **CRXDE Lite** page.
4. Navigate to **/content/hello-world.html**. Notice that the path does not exist in the repository.



At this point, Sling backs off the remaining part of the request until the first period (.) and everything after the **.** is considered the extension.

5. In **CRXDE Lite**, navigate to the **/content/hello-world** node, as shown. You should find a node in the repository that matches the resource in the request. What you just did is called *resolving the resource*. You resolved a request URL to a resource that is represented by a node in the repository.

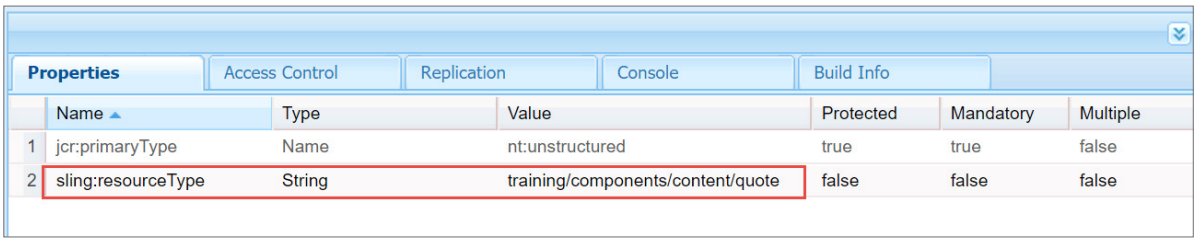


Note: You can map resources to the items in the relational databases and/or the file system. The Apache Sling specification does not mandate a JCR database.

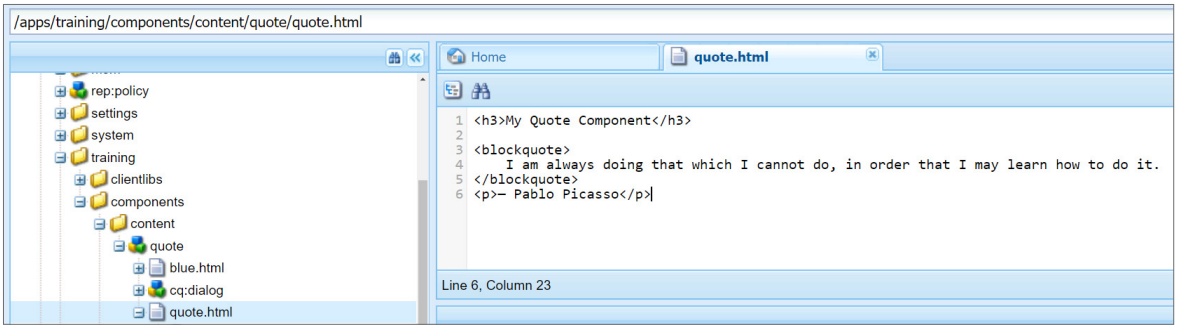
Now, you must find the rendering script.

6. Navigate to the **/content/hello-world** node and select the **sling:resourceType** property. The value of the **sling:resourceType property** is what Sling uses to begin its search for a rendering script.

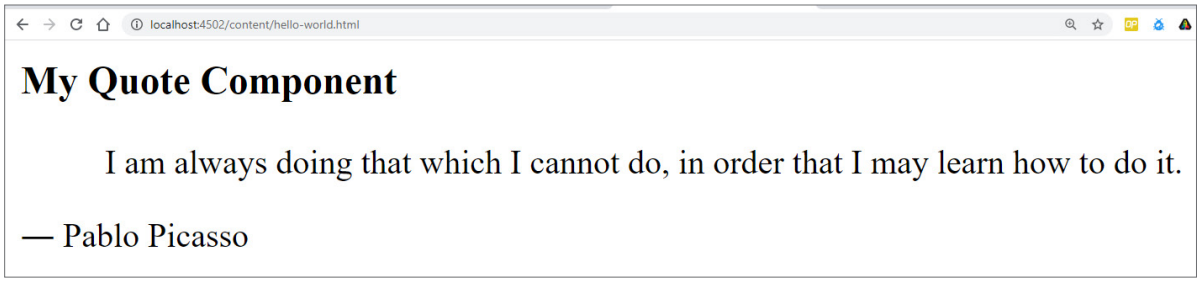
Notice that the **sling:resourceType** property points to the quote node you created previously:



7. Navigate to the **/apps/training/components/content/quote** node.
8. Notice the **quote.html** script. This is the default script because the script's name matches the name of the folder in which the script resides.



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 7: Customize 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 customize Sling to choose the script that you want to render.

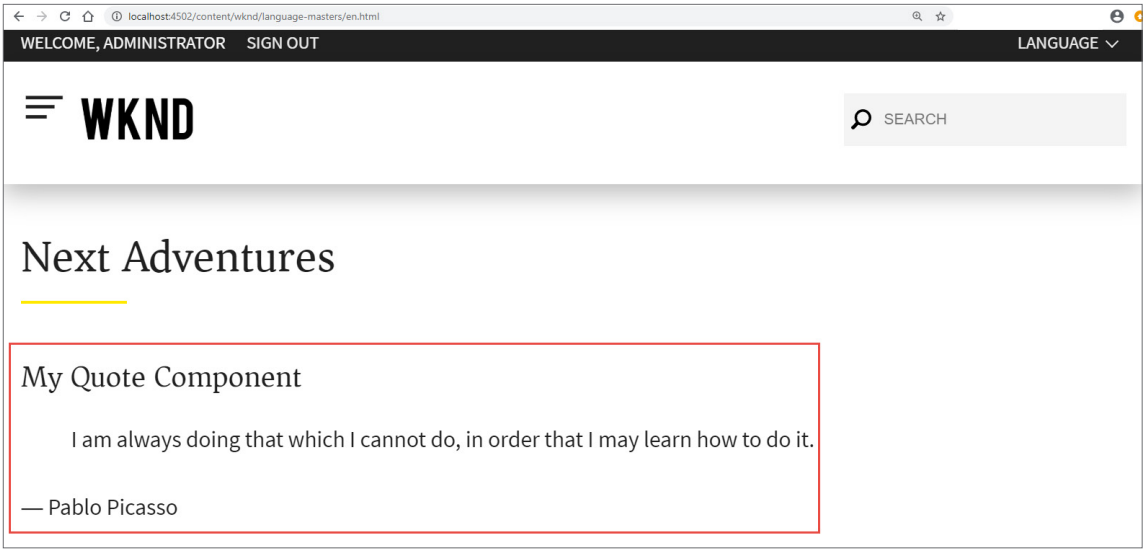
1. From **CRXDE Lite**, navigate to **/apps/training/components/content/quote** node.
2. Select the **quote** node, right-click it and click **Create** > **Create File**. The **Create File** dialog box opens.
3. Enter **blue.html** in the **Name** box and click **OK**. The **blue.html** file is created.
4. Click **Save All**.

To add code to the **blue.html** page:

5. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/content/quote** folder.
6. Open the **blue.html** file in Notepad++ (or any editor of your choice), copy the content of the file and paste it in the **blue.html** editor in **CRXDE Lite**.
7. Click **Save All**.

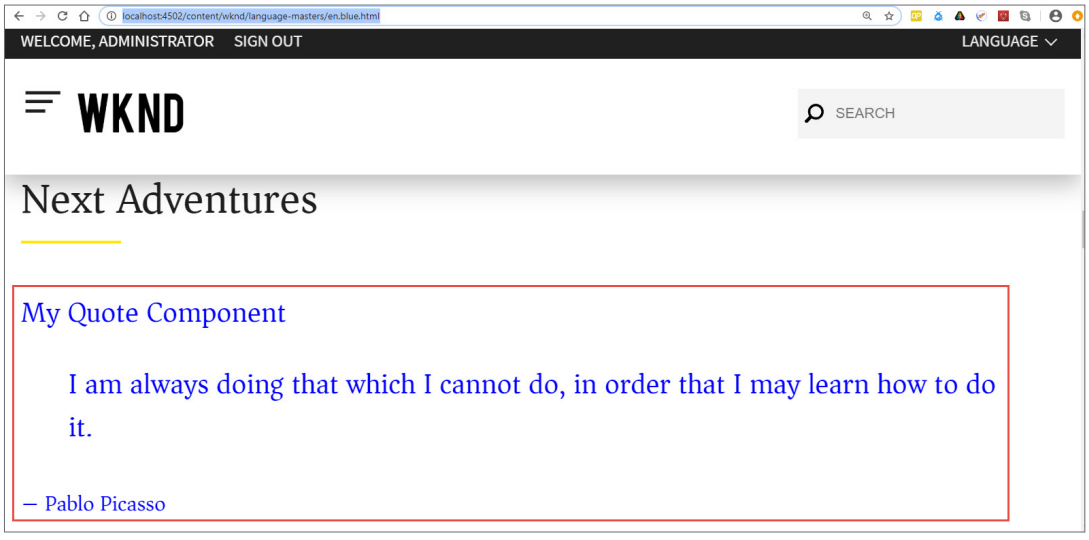
To test the selector:

8. Navigate to the WKND English page (http://localhost:4502/content/wknd/language-masters/en.html). You added the Quote component to this page in one of the previous exercises. If you do not see the Quote component, add it now. Notice that the text is in black font.



9. Add blue to the URL (http://localhost:4502/content/wknd/language-masters/en.blue.html) and refresh the page.

10. Notice the difference in font color. The code from **blue.html** is chosen as the script name that 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, 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 hierarchies:

- Resource Type
- Container
- Include

Resource Type Hierarchy

The resource type hierarchy is used to extend components using the `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 the 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, etc.), and dialog box layout (inline, floating, etc.) on the parent component, which are then propagated 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. This hierarchy can be used to separate a logic within a component further for a more modularized approach. Breaking a component's logic into multiple scripts and rendering them with a series of includes help extend and customize a component.

Exercise 8: Inheritance with resourceSuperType

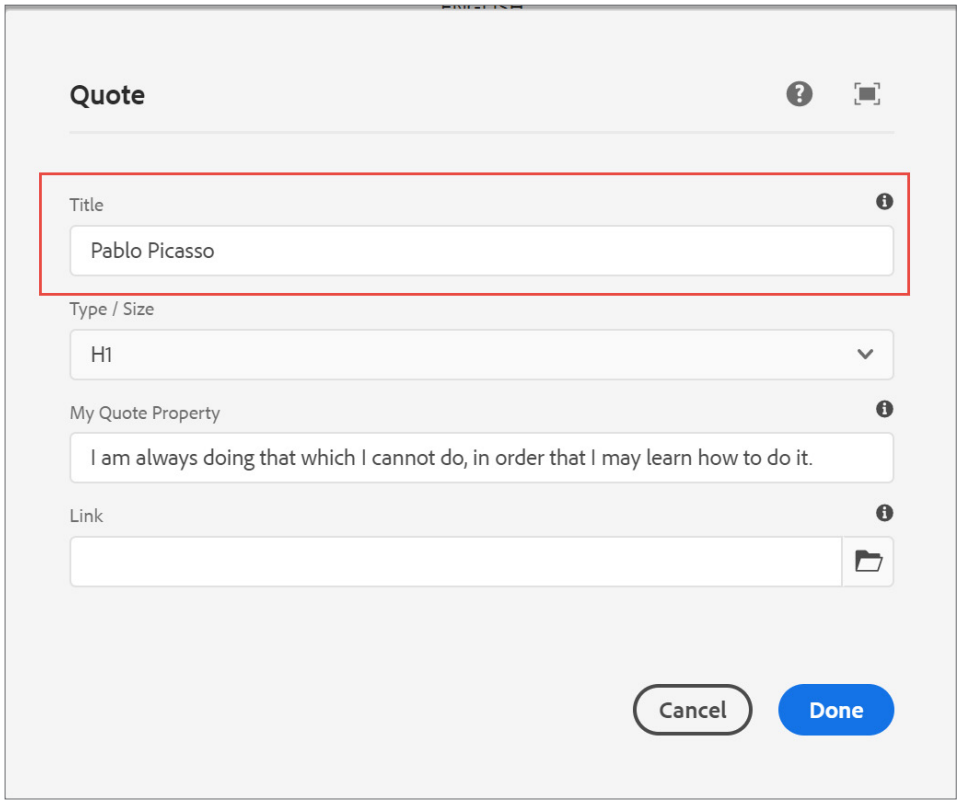
In this exercise, you will extend the quote component to inherit from the core title component. The sling:resourceSuperType property will be used to inherit from the core component rather than recreating code and dialog boxes.

1. In **CRXDE Lite**, navigate to the **/apps/training/components/content/quote** folder.
2. On the **Properties** tab, add the following property:

Name	Type	Value
sling:resourceSuperType	String	core/wcm/components/title/v2/title

3. Click **Add** to add the property.
4. Click **Save All** to save the changes.
5. To check if the quote component is inheriting from the Core title component, navigate to **Sites** > **WKND Site** > **Language Masters** in the AEM author service and open the **English** page in Edit mode.
6. Click the **quote** component you added earlier and click **Configure** icon (wrench icon). The **Quote** dialog box opens. Notice that the dialog now has four formfields rather than the single formfield. This is because, the cq:dialog node structure is same as the Core title component, and you can inherit the other formfields from the core title component. You can also use these fields in **quote.html** if needed.

7. In the **Title** box of the **Quote** dialog, enter **Pablo Picasso**, as shown:



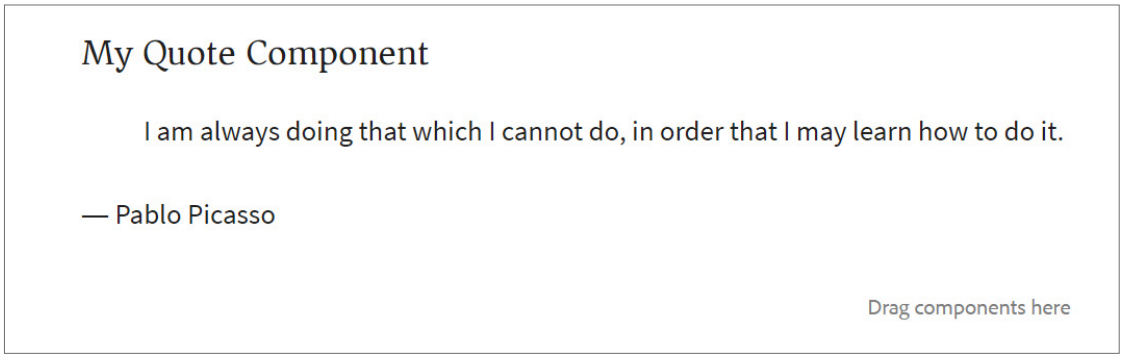
8. Click **Done**.

Note: Notice how you added a title to the dialog box, but nothing showed up on the page. This is because the **quote.html** script does not use this inherited formfield value. You must update the **quote.html** script to include the formfield value.

---

9. Navigate back to **CRXDE Lite**.
10. Navigate to **apps/training/components/content/quote**.
11. Double-click **quote.html**. The editor opens on the right. Delete the existing content.
12. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/content/quote** folder.
13. Open the **supertype-quote.html** file in Notepad++ (or any editor of your choice), copy the content of the file and replace the existing content in the **quote.html** editor in **CRXDE Lite** with the new content.

14. Click **Save All** to save the changes.
15. In AEM, navigate to **Sites** > **WKND Site** > **Language Masters** > **English**.
16. With the English page selected, click **Edit (*e*)** from the top menu bar to open the page**.** The component is updated, as shown:

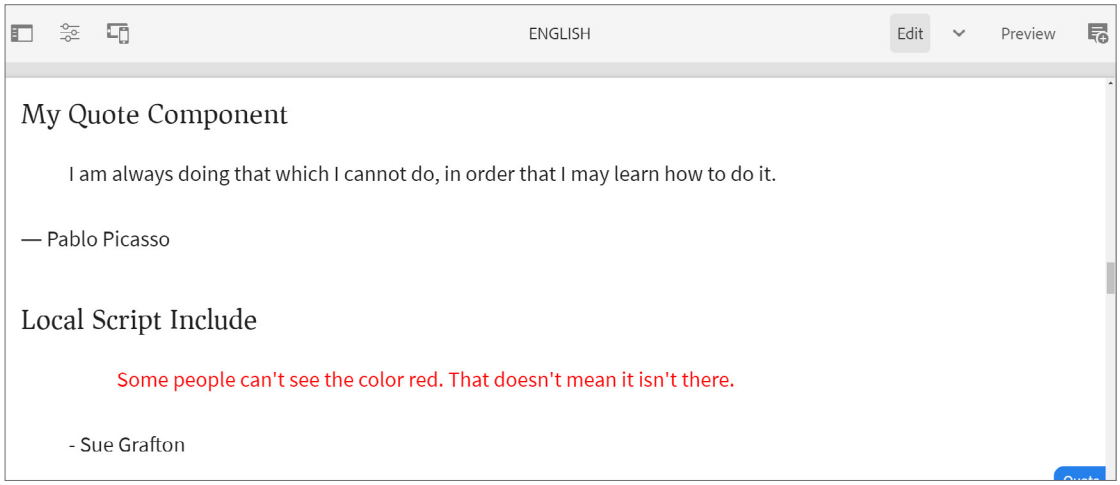


Exercise 9: Modularize the content by using multiple scripts

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

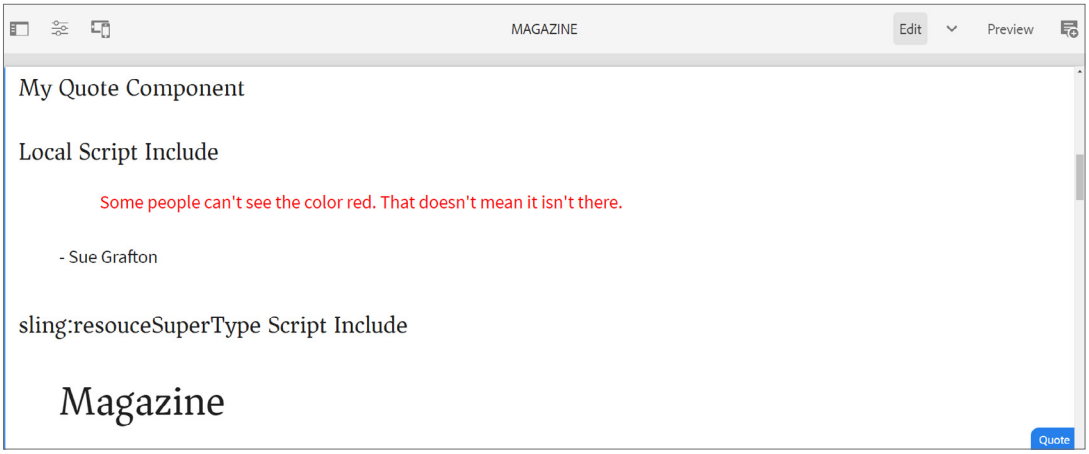
1. In **CRXDE Lite**, navigate to /**apps/training/components/content/quote**.
2. Right-click the **quote** component and click **Create** > **Create File**. The **Create File** dialog box opens.
3. In the **Name** box, enter **red.html** and click **OK**. The file is created and the editor opens on the right.
4. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/content/quote** folder.
5. Open the **red.html** file in Notepad++ (or any editor of your choice), copy the content of the file and paste it in the **red.html** editor in **CRXDE Lite**.
6. Click **Save All** to save the changes.
7. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/content/quote** folder.
8. Open the **includelocal-quote.html** file in Notepad++ (or any editor of your choice), copy the content of the file, and replace **quote.html** editor's existing content in **CRXDE Lite** with the new content.

9. Click **Save All** to save the changes.
10. To view the newly included script, navigate to **Sites** > **WKND Site** > **Language Masters** > **English** page and open it in the Edit mode. Notice how both the **quote.html** and **red.html** scripts are rendered with the Quote component, as shown. The script modularization is useful in code reuse. To utilize the Core title component in the Quote component, include a title.html script that does not exist in Quote and it will automatically be included from the Title component because of the sling:resouceSuperType property on the Quote component.



11. Navigate back to **CRXDE Lite**. Now that you have included local scripts to the quote component, include an inherited script from the core title component.
12. Open **quote.html** and delete its content.
13. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/content/quote** folder.
14. Open the **include-quote.html** file in Notepad++ (or any editor of your choice), copy the content of the file and replace the existing content in **quote.html** editor in **CRXDE Lite** with the new content.
15. Click **Save All**.
16. To view the newly included script, navigate to the **Sites** > **WKND Site** > **Language Masters** > **English** > **Magazine** page.

17. With the **Magazine** page selected, click **Edit (*e*)** from the top menu bar to open the page in Edit mode.
18. Click the **Toggle Side Panel** icon if the left panel is not visible.
19. From the **Components** tab on the left panel, drag and drop the **Quote** component onto the **Drag components here** area. Notice how both the **quote.html** and the inherited title.html script from the core title component are included in the Quote component.



References

- Script Resolution
- URL Decomposition
- Sling Cheat sheet

Introduction to HTML Template Language

Introduction

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

Objectives

After completing this module, you will be able to:

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

HTL

HTL is the recommended language for developing components in AEM. 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 the 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).

HTL: Goals

The main goals of HTL are to:

- Provide increased security through automated and context-sensitive cross-site scripting protection.
- Simplify development by helping HTML developers to write 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 types 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 ${ and }. 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 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
- HTL 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}`

Exercise 1: Render page content by using AEM global objects

In this exercise, you will explore different Java-backed global objects available in HTL and the ways to call their methods.

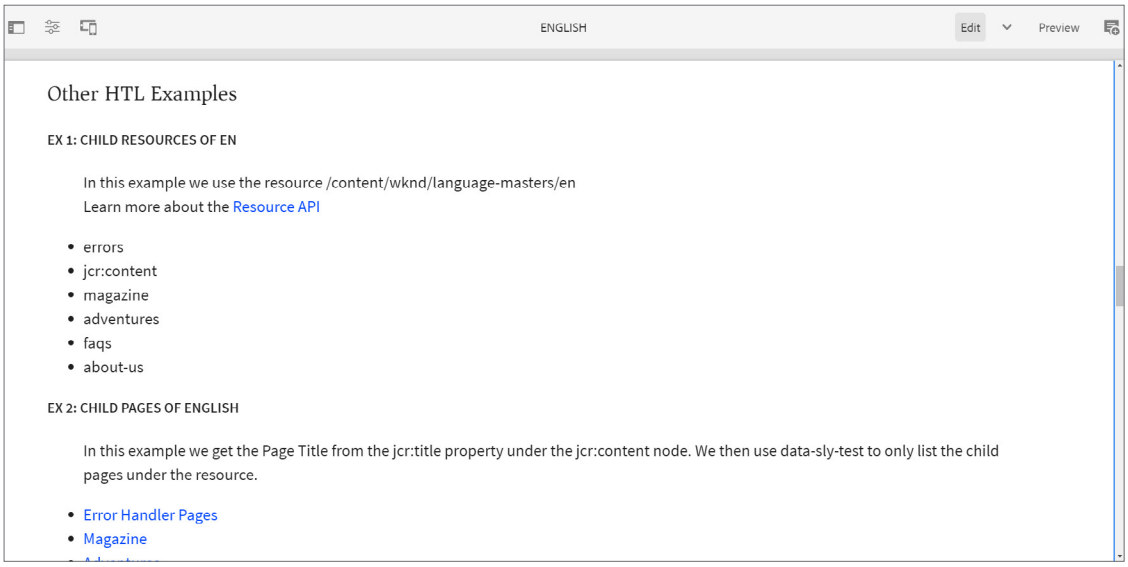
1. Open **CRXDE Lite** (http://localhost:4502/crx/de) if it is not already open.
2. Navigate to the **/apps/training/components/content/quote** component.
3. Double-click the **quote.html** file. The editor opens on the right. Delete the existing content.
4. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/content/quote** folder.
5. Open the **globalobjects-quote.html** file in Notepad++ (or any editor of your choice), copy the content of the file and replace the existing content in the **quote.html** editor in **CRXDE Lite** with the new content.
6. Click **Save All** to save the changes.
7. In the AEM author service, navigate to **Sites** > **WKND Site** > **Language Masters** > **English**.
8. With the English page selected, click **Edit** (***e***) on the actions bar to open the page in edit mode.

9. Examine the Quote component on the page. If you do not see the Quote component on the English page, drag the **Quote** component from the **Components** tab on the left panel on the page. Notice how different global objects are used for different content within AEM. Depending on what you are trying to accomplish, different objects can be used for desired values.

Exercise 2: Render page content by using HTL attributes

In this exercise, you will add a new script to the quote component, observe different HTL attributes and what the attributes are accomplishing.

1. In CRXDE, navigate to **/apps/training/components/content/quote**.
2. Double-click **quote.html** to update the code. The editor opens on the right. Delete its content.
3. In the Exercise Files folder provided to you, navigate to the ui.apps/src/main/content/jcr_root/apps/training/components/content/quote folder.
4. Open the **htl-quote.html** file in Notepad++ (or any editor of your choice), copy the content of the file, and replace the existing content in the **quote.html** editor in **CRXDE Lite** with the new content.
5. Click **Save All**.
6. In the AEM author service, navigate to **Sites** > **WKND Site** > **Language Masters** > **English**.
7. Open the English page in edit mode and observe the newly rendered quote script. Notice how the different HTL attributes allow differently rendered HTML. Carefully inspect the comments in your new **quote.html** script to better understand what these attributes are accomplishing.



References

- Experience Manager HTL Help
- HTL Specifications:

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

AEM Sites Development: Key Concepts

Introduction

To use Adobe Experience Manager (AEM) capabilities effectively, you must understand the key concepts such as templates, core and proxy components, responsive pages, and context-aware configurations.

Objectives

After completing this module, you will be able to:

- Explain AEM templates
- Explain core components and proxy components
- Create a page component
- Explain responsive layout editing
- Explain context-aware configuration
- Create a context-aware configuration

Templates

AEM helps organizations build a website and add content to webpages. Based on business requirements, authors can create pages by using a specific template and adding 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 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 define and configure static templates. Note that the static template design information stored under /apps cannot be edited from the UI. Additionally, configuration information must come from Git through the CI/CD pipeline.
- Editable templates: An editable template retains a dynamic connection to any pages created from the template. This ensures that any changes to the template are reflected on the pages. Template authors create and manage editable templates. Editable templates are the preferred template type in AEM.

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 the required information. They should have experience in working with development environment.
- Template author/Power author: Creates templates and configures the use of components. To accomplish their tasks successfully, they must be a part of the *template-authors* group and may need additional technical information from developers. Template authors must have some experience in handling technical tasks, such as using patterns when defining paths.

Core Components and Proxy Components

Core Components

In AEM, components are the structural elements that constitute the content of the pages being authored. Core components make page authoring simple, 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:

- 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:
  - Compatibility within a version with the flexibility for components to evolve.
  - Multiple versions of one component can coexist on the same environment.
- Modern implementation:
  - Markup defined in HTML Template Language (HTL)
  - Content model logic implemented with Sling Models
- Lean markup:
  - 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 various Layout Containers and Paragraph Systems.
- For the 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 Component Guidelines

Proxy Components

To use Core components in your project, you must download and install the core components package on your AEM author service, 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 Core components directly in the template editor.

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 refer 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.

Best Practice: Create proxy components from core components and use the proxy components in your project.

Create a Page Component

AEM has a set of node types to make content rendering 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 though 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 realizes 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 use this single page component.

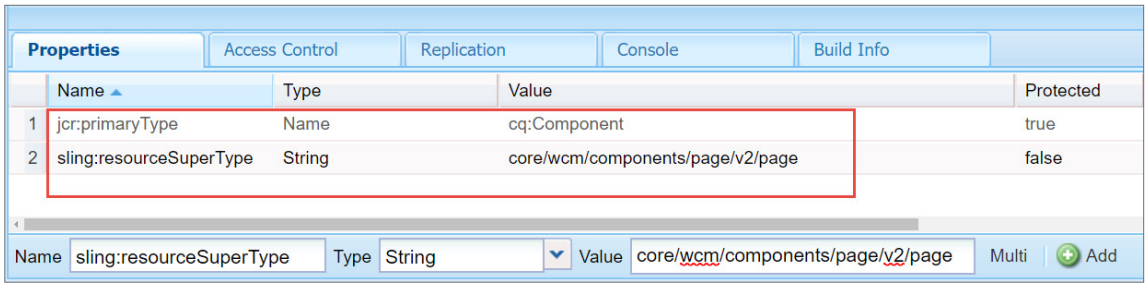
Exercise 1: Create a proxy page component

To create a proxy page component:

1. In **CRXDE Lite**, navigate to the **/apps/training/components/structure** folder.
2. Right-click the **structure** folder and click **Create** > **Create Node**. The **Create Node** dialog box opens.
3. In the **Name** box, type **page**.
4. From the **Type** drop-down menu, select **cq:Component**.
5. Click **OK**. The node is created.
6. Select the page node you created now and add the following property:

Name	Type	Value
sling:resourceSuperType	String	core/wcm/components/page/v2/page

7. Click **Save All** to save the changes. The page node properties should look as shown:



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
- An 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

In AEM, you can 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 author 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:

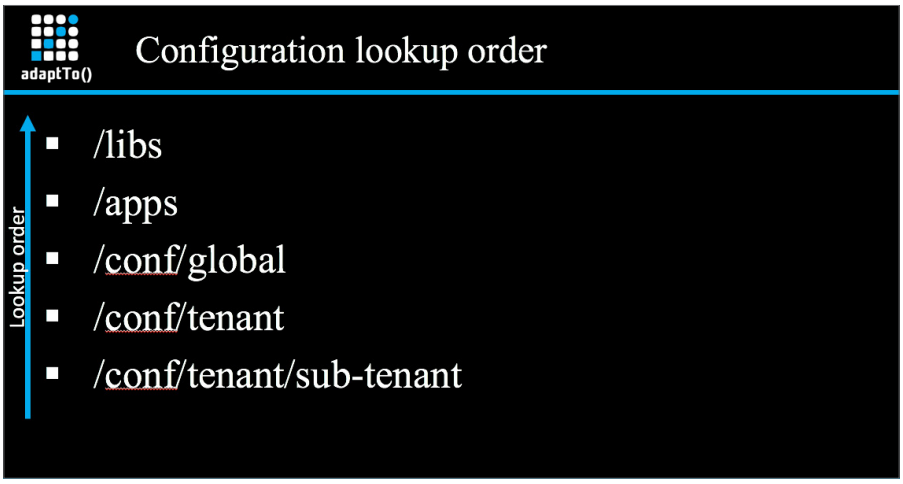
- Define differing content layouts based on device width (related to device type and orientation) by using breakpoints.
- Ensure that the content is responsive to the size of the browser window on the desktop by using the same breakpoints and content layouts.
- Place components in the grid, resize as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

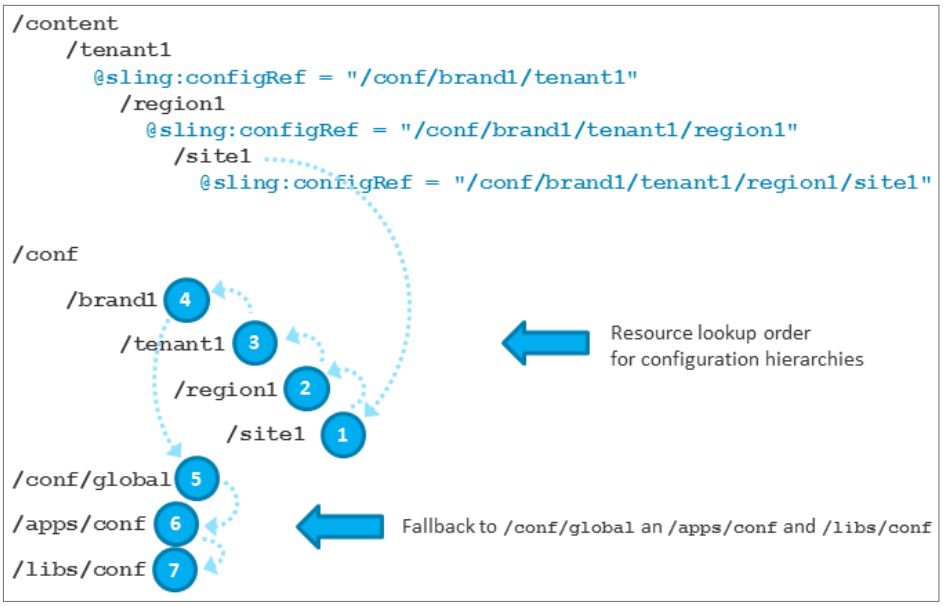
Currently, context-aware configurations support:

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

As new features are added or updated in AEM, this list will expand. 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:



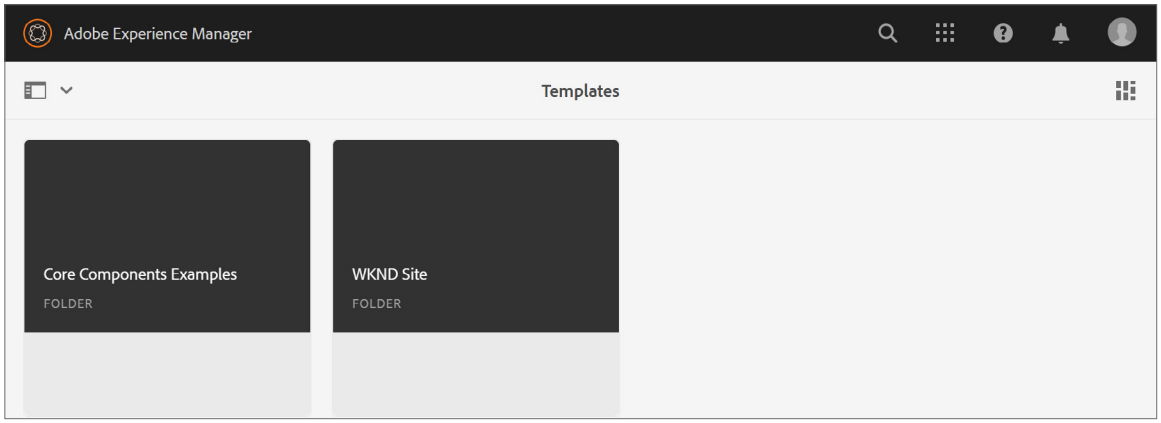
The rules you must follow in resource lookup are:

- 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 2: Create a context-aware configuration

Before you create a context-aware configuration, check if you can create a template folder in the **Templates** console.

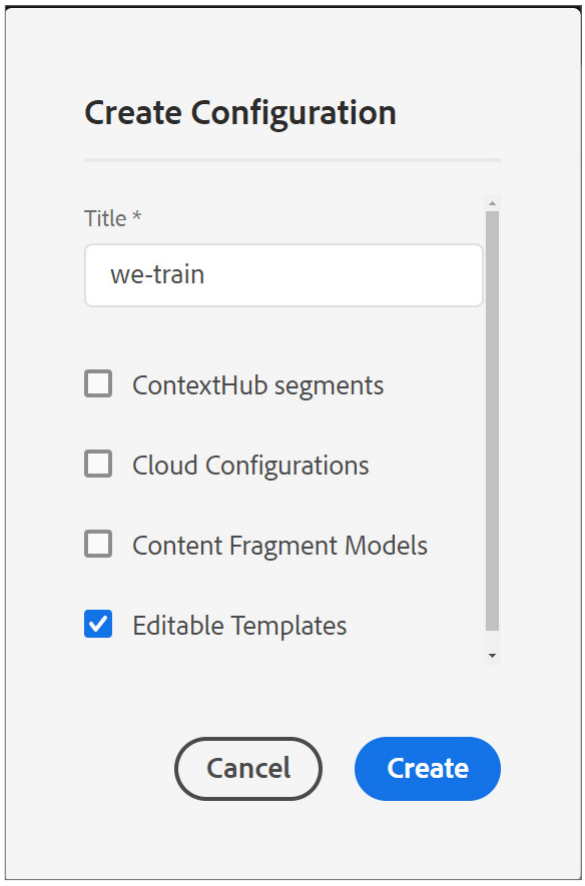
1. In AEM author service, click **Adobe Experience Manager** from the header bar to open the navigation pane.
2. Click the **Tools** icon and click **Templates**. The **Templates** console opens. Notice that there is no way to create a new template folder in the **Templates** console.



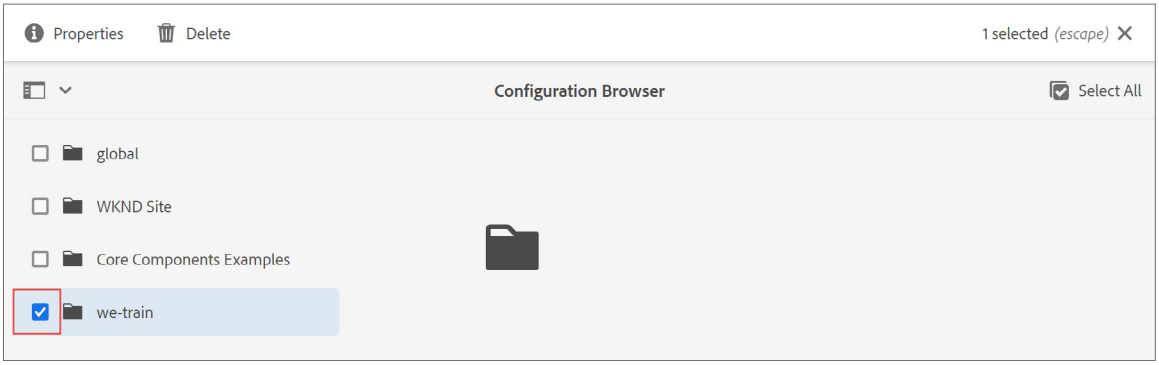
To create a new folder for editable templates, you must create a context-aware configuration.

3. Click **Adobe Experience Manager** > **Tools** > **Configuration Browser**. The **Configuration Browser** console opens.
4. Click **Create** from the actions bar. The **Create Configuration** dialog box opens.

5. In the **Title** box, enter **we-train**.
6. Select the **Editable Templates** check box, as shown:

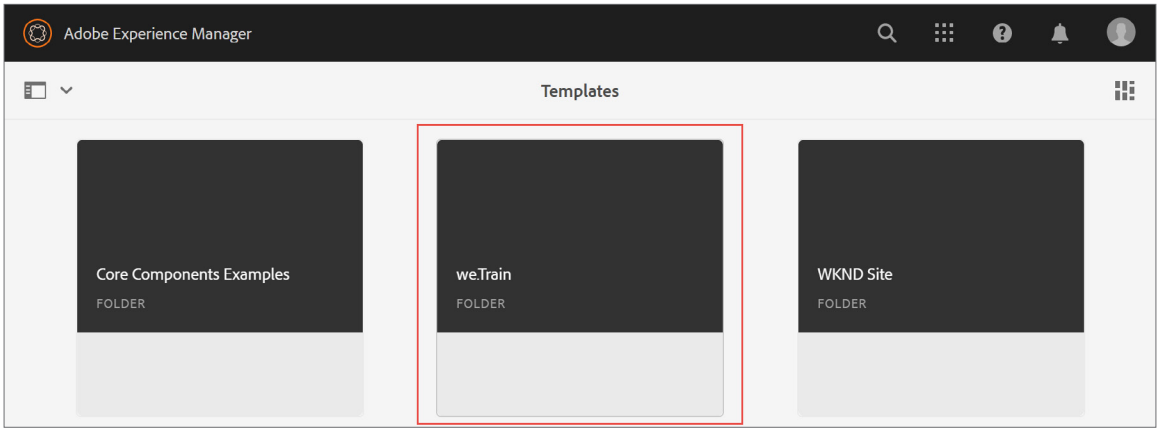


7. Click **Create**. A success message appears confirming the creation of the configuration.
8. Click the thumbnail beside the **we-train** configuration, as shown, and click **Properties** from the actions bar. The **Configuration Properties** page opens.



9. Update the title to **We.Train**.

10. Click **Save & Close**. A success message appears confirming the change.
11. Navigate to **Adobe Experience Manager** > **Tools** > **Templates**. The **Templates** console opens. Notice that the **We.Train** folder is now available. You will use this folder in a later exercise to create an editable template.



References

- AEM Sites as a cloud service
- AEM Core components: GitHub

Creating Editable Templates and Pages

Introduction

In Adobe Experience Manager (AEM), template 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 must set up the required environment and create client libraries and components to be used on the page.

Objectives

After completing this module, you will be able to:

- Explain templates in AEM
- Create editable templates
- Create pages from editable templates

Templates

Content authors can easily create pages and content from the AEM Sites console.

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

Types of Templates

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

Templates Console

The Templates console enables template authors or power authors to:

- 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 enables template authors to:

- 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: The structure enables template authors to define the components, such as header, footer, and layout container for a page. Template authors can add a paragraph system to the template, which helps page authors to add and remove the components on the resultant pages. When components are locked, page authors cannot edit the content. In Structure mode, any component that is the parent of an unlocked component cannot be moved, cut, or deleted.
- Initial Content: Power authors can 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 pages created from the template. Page authors can edit these unlocked components on the resultant pages. 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: The Layout enables power authors to predefine the template layout for the required device formats.

Content Policies

You can define content policies for templates and components from the template editor.
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.
- Enable you to add the allowed components and default components to the template.
- Define the number of columns (responsive settings) in the template.

Creating Editable Templates

The steps to create an editable template are:

1. Create a context-aware configuration folder and enable editable 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.
7. Make the template available for the required page or the branch of your website.
8. Publish the template so that it is available on the publish environment.

Creating the Template Folder

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

- global: In a standard AEM author service, 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).
- site-specific: The site-specific folders help organize templates specific to your site.

Best Practice: Create a new folder to save your customized templates instead of using the global folder. A user with admin rights can create folders.

Creating a Template Folder from the Configuration Browser Console

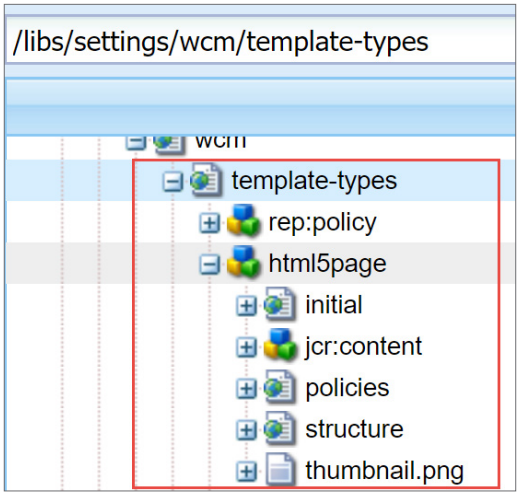
You can access the **Configuration Browser** console from the **Tools** console. You can create a multi-tenant environment where tenants can have different editable templates, configurations, cloud services, and data models.

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.
Template types help you define the:

- Resource type of the page component
- 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:



Template Definitions

The definitions for editable templates are stored under 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

The jcr:content 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 on 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 or deleted from any resultant pages. After a component is unlocked, the editable property is set to true. After a component that already contains the content is unlocked, the content is moved to the initial branch. The **cq:responsive** node holds the 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 the 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 the 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 the resultant pages.

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. 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 must enable it:

- From the **Templates** console

Or

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

To ensure you can access the template, you must:

1. Define the **Allowed Template paths** on the **Page Properties** of the appropriate page or root of the page of a sub-branch.
2. 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 must 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 an empty template type
2. Create a development template
3. Enable and publish the template

Task 1: Create an empty page template type

Before you create a template type, check the template types that are available by default.

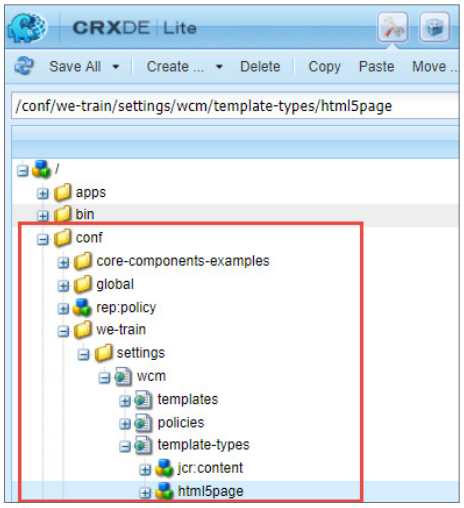
1. Navigate to **Adobe Experience Manager** > **Tools** > **Templates** if you are not in the **Templates** console already.
2. Click the **We.Train** folder and click **Create** from the actions bar. The **Create Template page** opens. Notice that template types are already available. These are default template types and if you use one of these template types, you would use the default components. You must create your own components and design so that you can create your own template type based on the HTML5 Page type.
3. Click **Cancel** to close the **Create Template** page.

Template types connect to a base page component. The page components are responsible for ensuring that the global areas of the site are consistent. This includes loading of global CSS and Javascript as well as the inclusion of the code that will ensure the page can be edited by using AEM authoring tools. You will use the training/components/structure/page component you created earlier.

To create a custom template type:

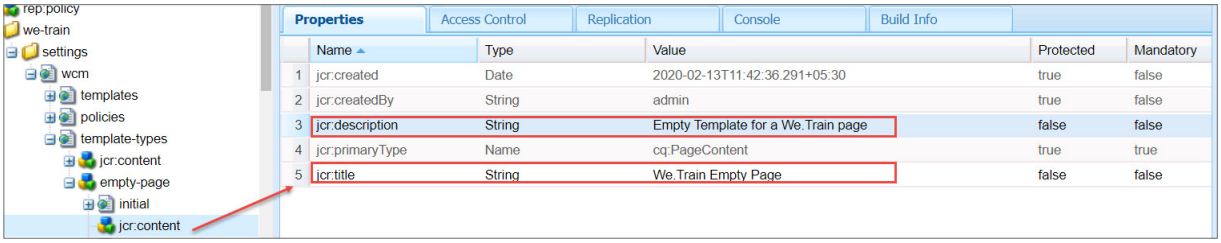
4. Click **Adobe Experience Manager** from the header bar. The **Tools** console opens.
5. Click **CRXDE Lite**. The **CRXDE Lite** page opens.
6. Navigate to the **/libs/settings/wcm/template-types/html5page** node.

7. Right-click the **html5page** node and click **Copy**. The node is copied.
8. Navigate to the **/conf/we-train/settings/wcm/template-types** folder.
9. Right-click the **template-types** folder and click **Paste**. The **html5page** node is added to the **template-types** folder, as shown:



10. Click **Save All** from the actions bar.
11. Right-click the **html5page** node and click **Rename**.
12. Rename the node to **empty-page**.
13. Expand the **empty-page** node by clicking the + icon and select the **jcr:content** node under it.
14. Modify the properties under the **Properties** tab:
    a. **jcr:title**: **We.Train Empty Page**
    b. **jcr:description**: **Empty Template for a We.Train page**

The **jcr:content** node **properties** should now look, as shown:



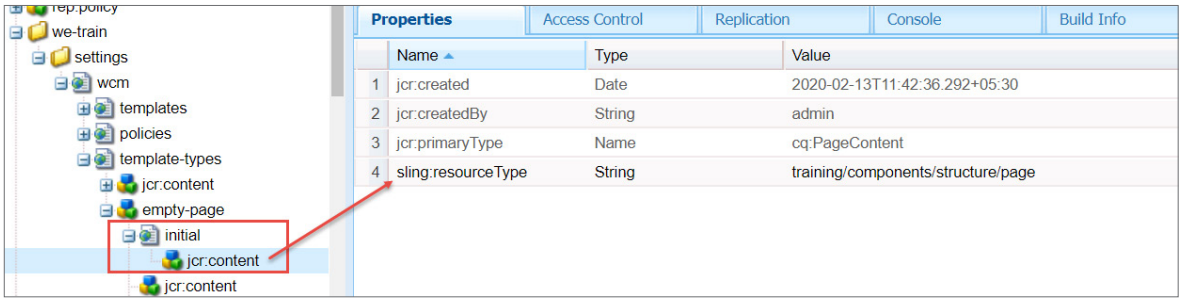
15. Click **Save All** from the actions bar.

16. Select the **empty-page/initial/jcr:content** node.

17. Modify the property under the **Properties** tab:

    a. **sling:resourceType**: **training/components/structure/page**

The empty-page/initial/jcr:content node property should now look, as shown:



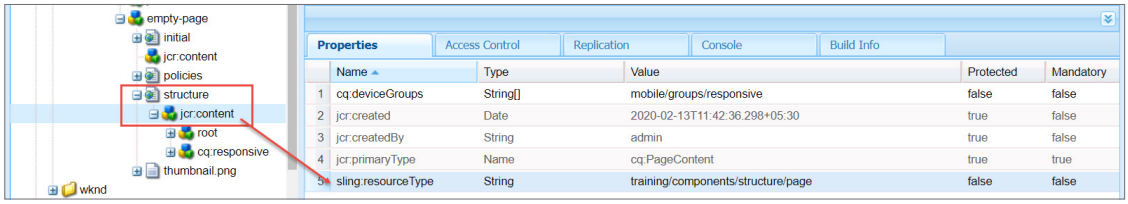
18. Click **Save All** from the actions bar.

19. Select the **empty-page/structure/jcr:content** node.

20. Modify the property under the **Properties** tab:

    a. **sling:resourceType**: **training/components/structure/page**

The **empty-page/structure/jcr:content** node property should now look, as shown:



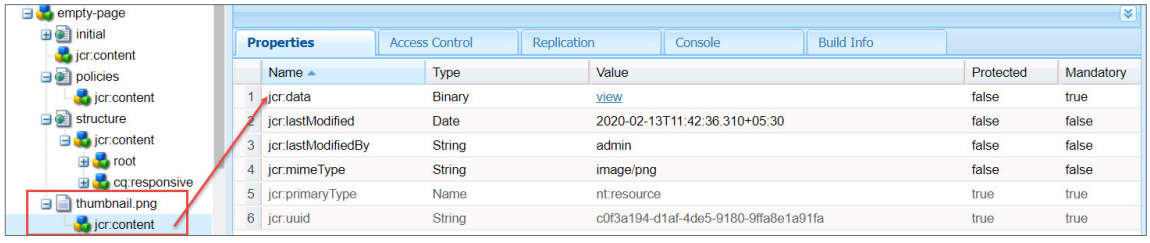
21. Click **Save All** from the actions bar.

22. Navigate to **empty-page/policies/jcr:content**, select the **root** node, right-click it and click **Delete** to delete the node. Now, you can start with an empty root layout container.

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

24. Navigate to the **empty-page/thumbnail.png/jcr:content** node.

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



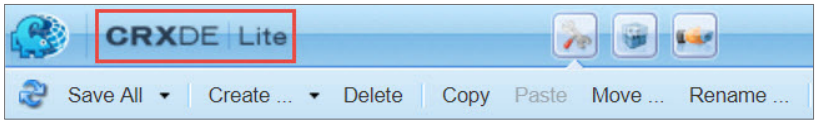
26. Click **Browse**. The **Open** dialog box opens.

27. In the Exercise Files folder provided to you, navigate to the **training files** > **editable templates** folder, select the **We.Train-thumbnail.png** image and click **Open**.

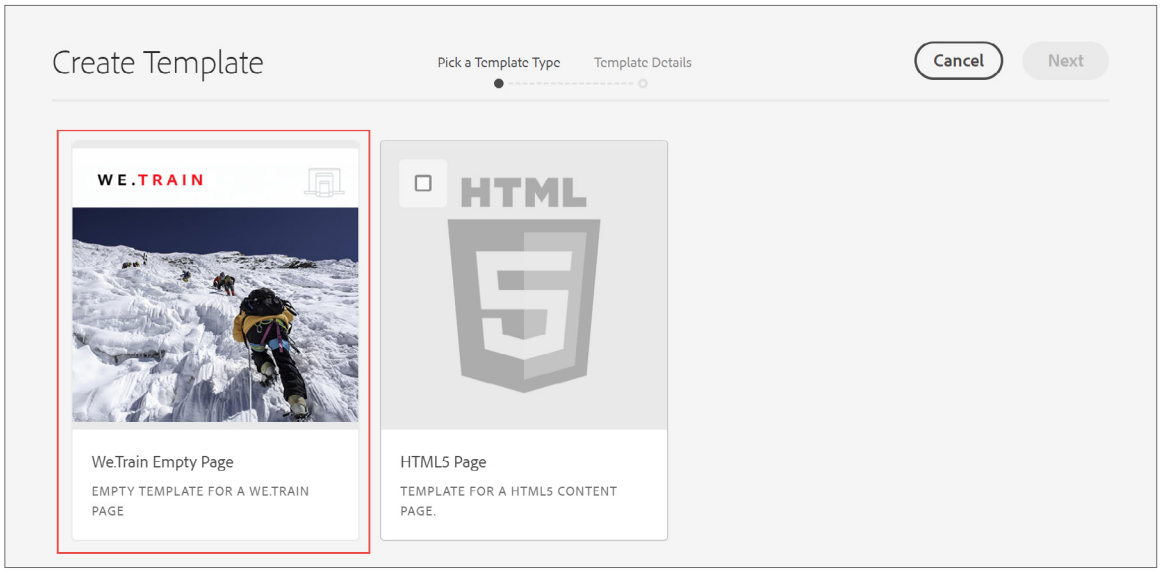
28. Click **OK**. The thumbnail image is added to template-type.
29. Click **Save All** to save changes.

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

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



31. Click the **Tools** icon and click **Templates**. The **Templates** console opens.
32. Click the **We.Train** folder and click **Create** from the actions bar. The **Create Template** page opens. You should now see the **We.Train Empty Page** template type, as shown:



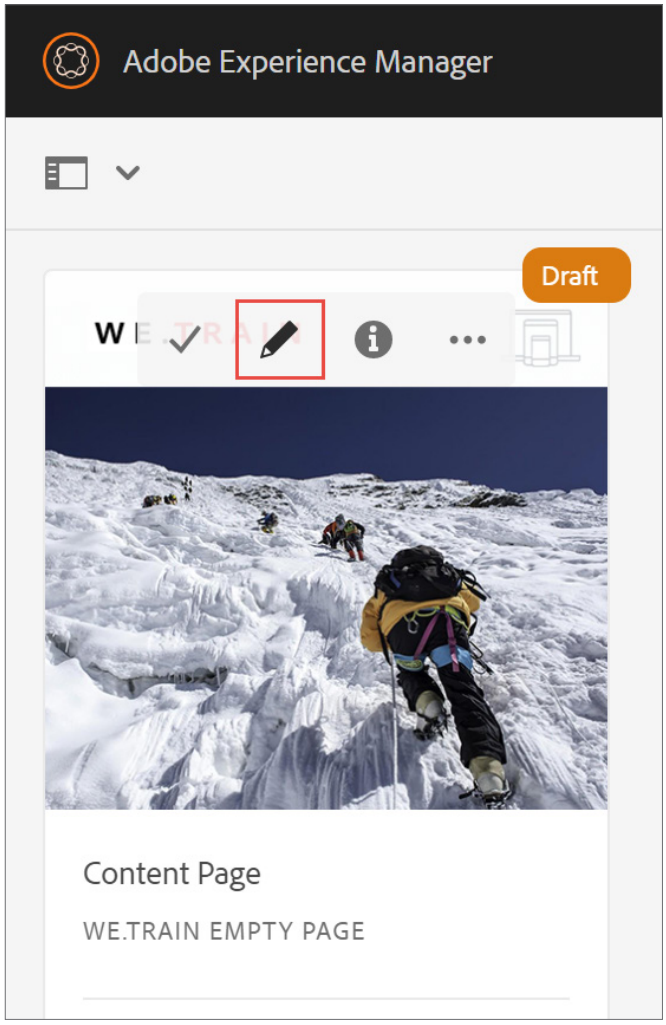
Task 2: Create a development template

After creating the template-type, you must create a development template. The development template enables you to use the template authoring environment and to setup initial template configurations. After the setup is ready in the development template, you will copy it back to the template type.

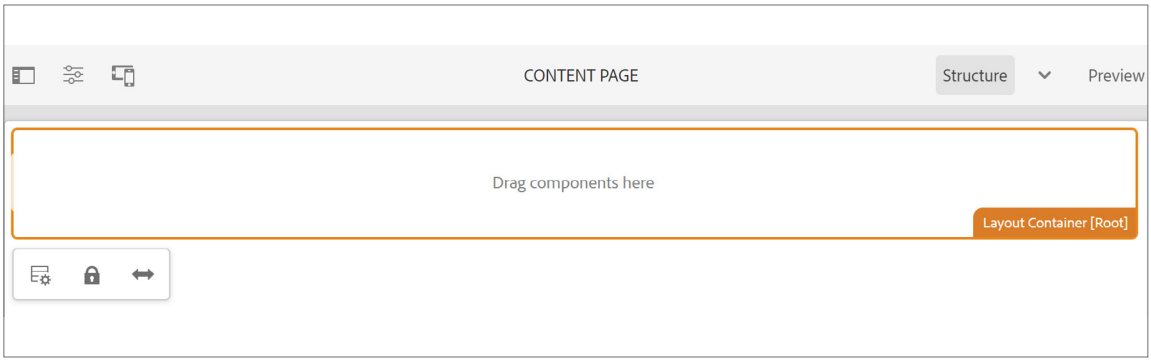
To create a development template:

1. If you are not already in the **Create Template** page, navigate to **Adobe Experience Manager** > **Tools** > **Templates** > **We.Train** folder.
2. Click **Create** from the actions bar. The **Create Template** page opens.
3. Select the **We.Train Empty Page** template and click **Next**.
4. In the **Template Title** box, enter **Content Page** and click **Create**. The **Success** dialog box opens.

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



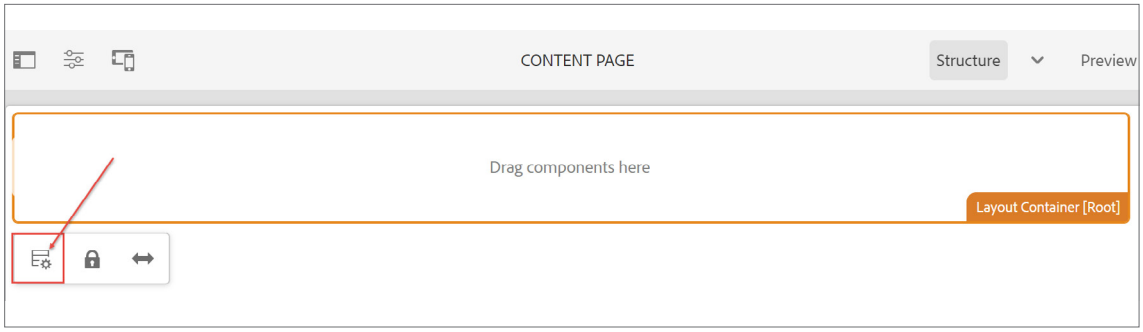
The **Layout Container [Root]** is now visible, as shown. The **Layout Container [Root]** helps configure the components that can be added to the template.



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 must now add a few structure components to the root layout container, so that you can start building the template.

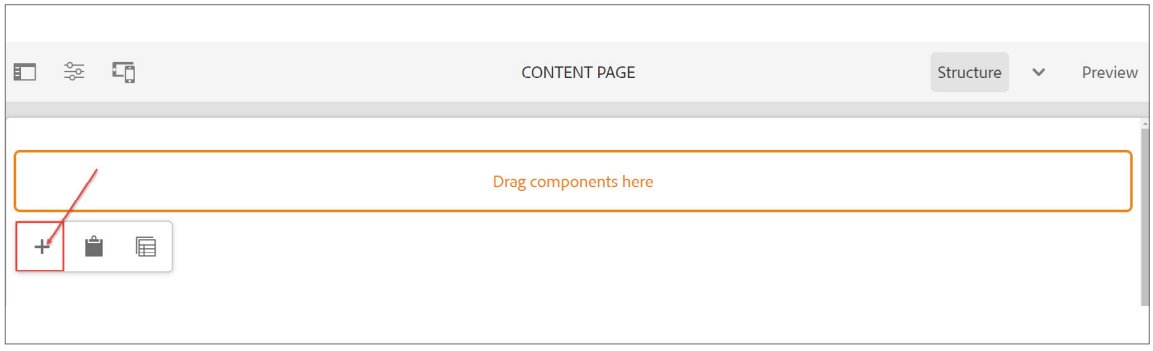
7. Select the **Layout Container [root]** and click the **Policy** icon, as shown. The **Layout Container** page opens.



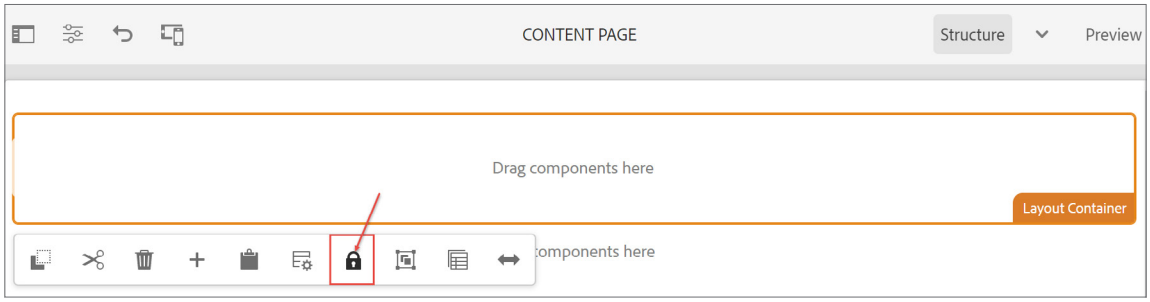
8. On the **Policy** section, in the **Policy Title** box, enter **We.Train Structure Policy**.
9. On the **Properties** section, ensure you are in the **Allowed Components** tab and expand the **General** group. The list of components available in the group is displayed.
10. Select **Content Fragment**, **Experience Fragment**, and **Layout Container** check boxes.
11. Click **Done**. The selected components are added to the policy.

Now that you have structure components in the template, you will add a layout container that will be used in the resulting pages. This layout container will contain approved content components that authors can drag onto a page.

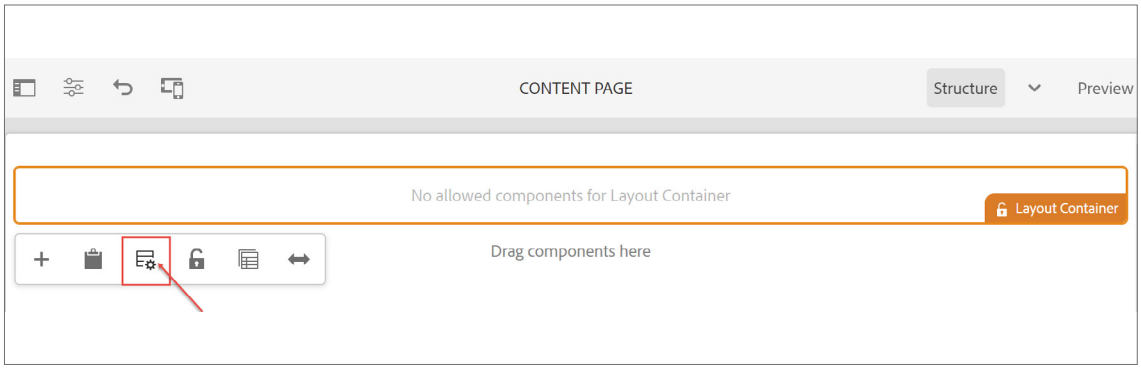
12. Refresh the page.
13. Ensure you are in the **Structure** mode and click within the **Drag components here** layout.
14. Click the **Insert component** icon, as shown. The **Insert New Component** dialog box opens.



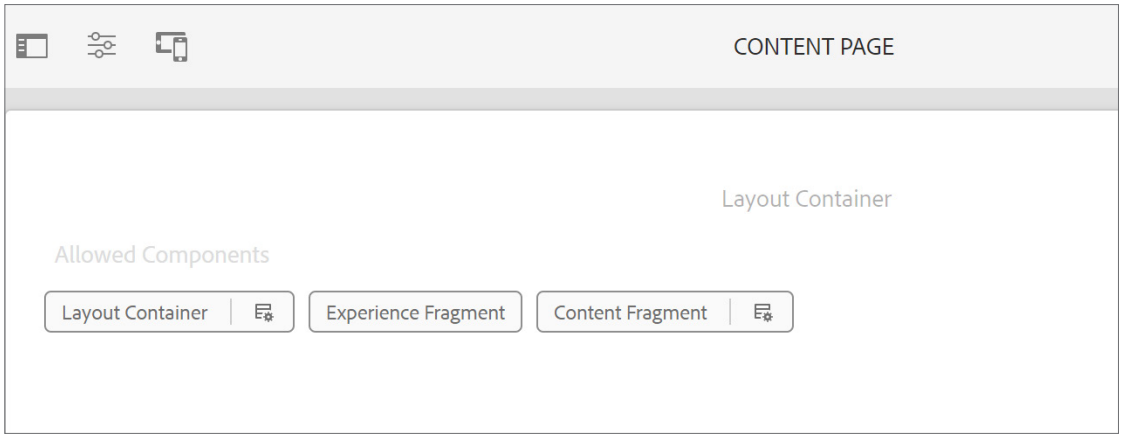
15. Select **Layout Container** from the list of components. The component is added to the template.
16. Select the inner Layout Container (not the root Layout Container) and click the **Unlock Structure Component** icon from the component toolbar, as shown. This enables you to add components to the container.



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

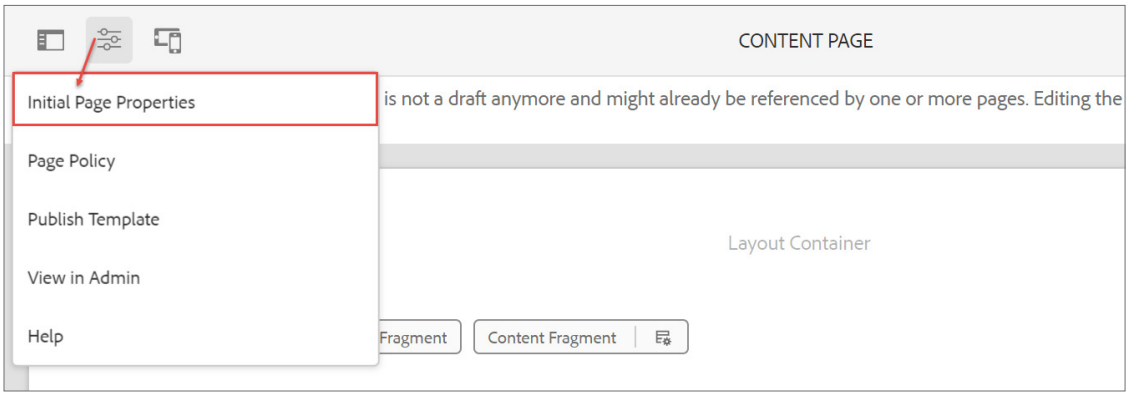


18. On the **Policy** section, in the **Policy Title** box, enter **We.Train Content Policy**.
19. On the **Properties** section, ensure you are in the **Allowed Components** tab and expand the **General** group. The list of components available in the group is displayed.
20. Select **Content Fragment**, **Experience Fragment**, and **Layout Container** check boxes.
21. Click **Done**. The selected components are added to the policy.



To add a thumbnail to the initial page properties:

22. In the template editor, click **Page Information** > **Initial Page Properties**, as shown:



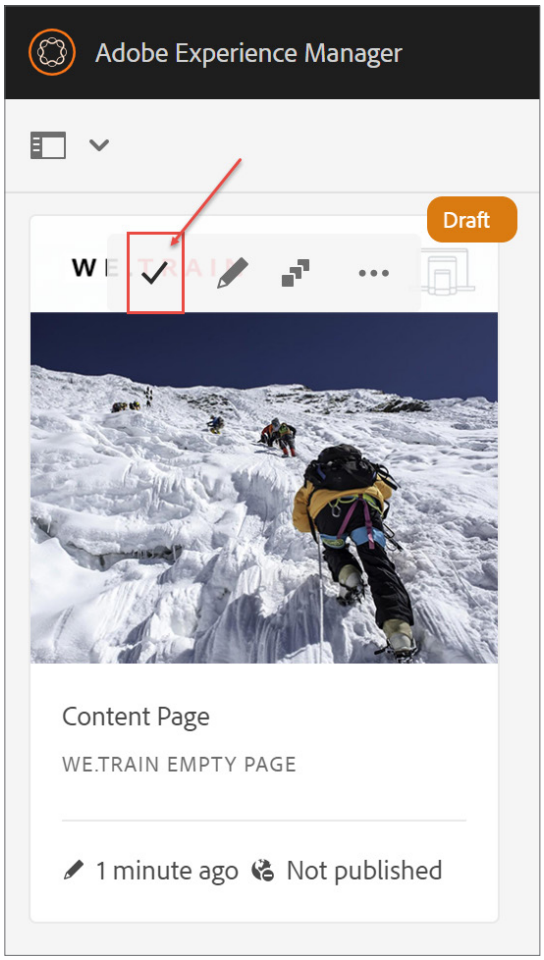
23. On the **Thumbnail** tab, click **Upload Image**.
24. In the Exercise Files folder provided to you, navigate to the **training files** > **editable templates** folder and select **We.Train-thumbnail.png**.
25. Click **Open**. The image is uploaded.
26. Click **Save & Close**.

Task 3: Enable and publish the template

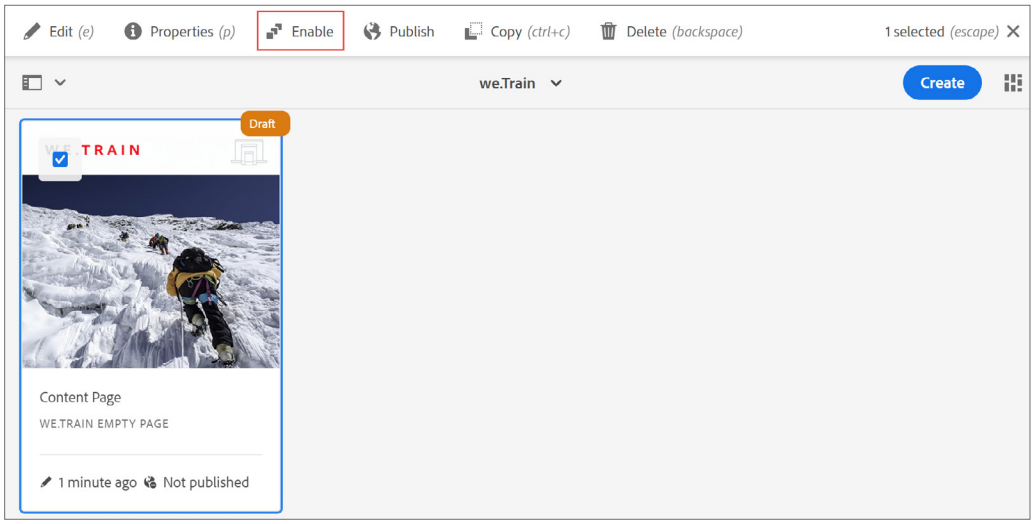
In this exercise, you will first enable the template so that it is available for authors to create pages on the author service. Later, publish the template to make it available on the publish service.

To enable the template:

1. In the AEM author service, navigate to **Adobe Experience Manager** > **Tools** > **Templates**. The **Templates** console opens.
2. Click the **We.Train** folder. The Content page template you created earlier is displayed.
3. Hover the cursor over the **Content Page** template and click the **Select** icon, as shown. The template is selected.



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



5. Click **Enable**. Notice that the status of **Content Page** changes to **Enabled**.
6. Ensure the **Content Page** template is selected and click **Publish** from the actions bar. The **Publish** dialog box opens.
7. Ensure all check boxes are selected and click **Publish**. The message, **The page has been published**, appears.

Creating Pages from Editable Templates

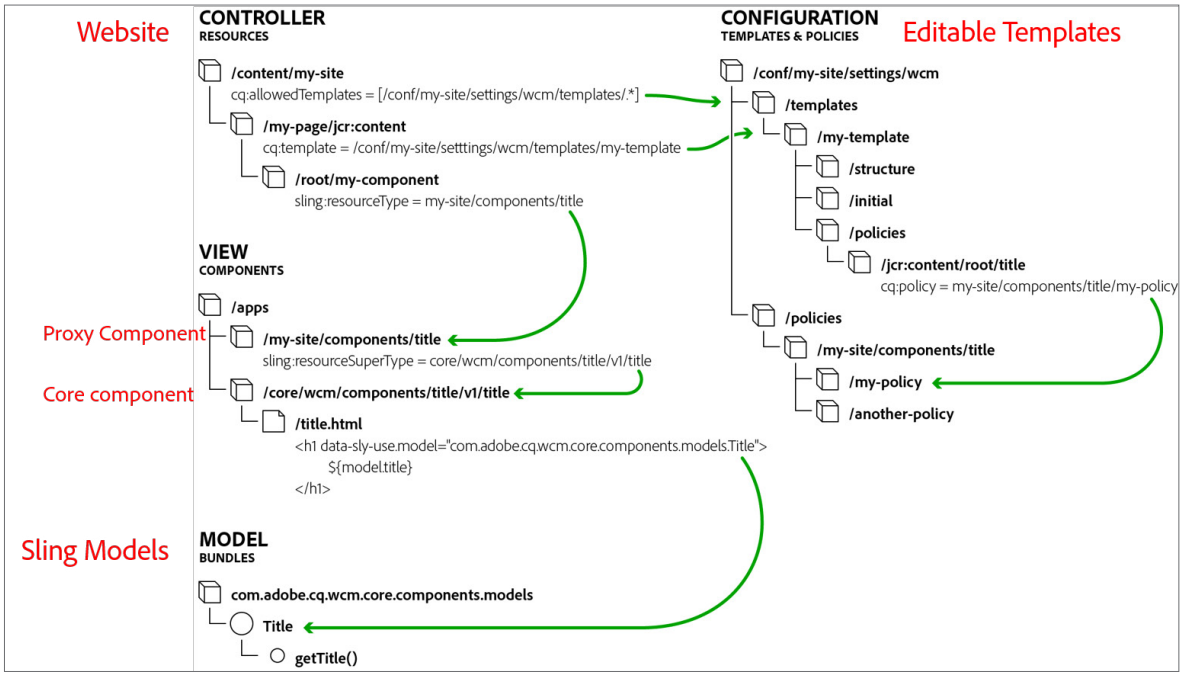
Before creating pages through the UI, a content root needs to be set up for your site. The content root will define the allowed templates for the site and is used to set other global configurations. By convention, the content root is not intended to be the home page for the site but instead it redirects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

Creating Pages

In the **Sites** console, you can create the pages of your website by using a 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 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 is 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 those nodes of the template structure that are flagged as editable.
- The relative path of the component is 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 2: Create a page

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

This exercise includes the following tasks:

1. Create a content root
2. Create a site structure

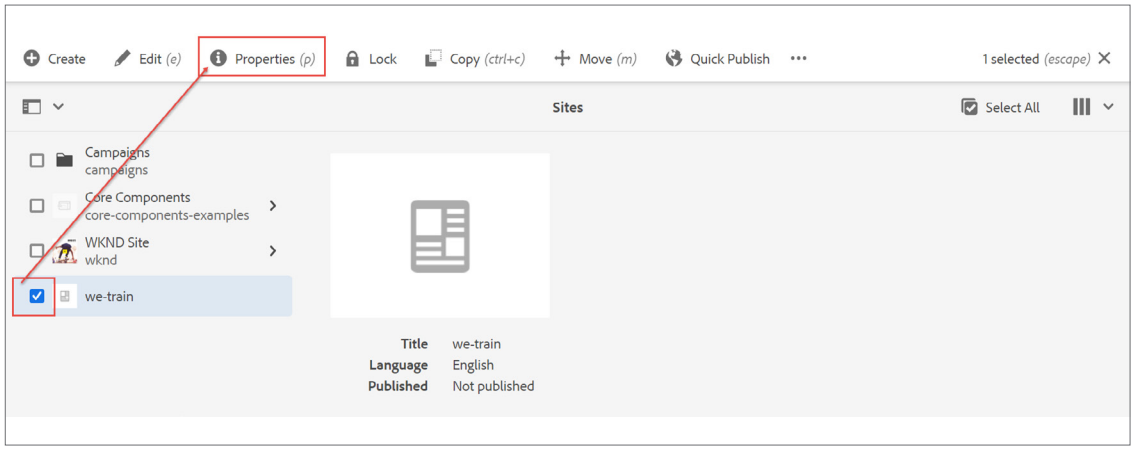
Task 1: Create a content root

1. Click http://localhost:4502/crx/de/index.jsp. The **CRXDE Lite** page opens.
2. Right-click the **/content** folder and click **Create** > **Create Node**. The **Create Node** dialog box opens.
3. In the **Name** box, enter **we-train**.
4. From the **Type** drop-down menu, select **cq:Page**.
5. Click **OK**. The node is created.
6. Click **Save All**.
7. Right-click the **we-train** node and click **Create** > **Create Node**. The **Create Node** dialog box opens.
8. In the **Name** box, type **jcr:content**.
9. From the **Type** drop-down menu, select **cq:PageContent**.
10. Click **OK**. The node is created
11. Click **Save All**.
12. Select the **jcr:content** node that is below the **we-train** node.
13. On the **Properties** tab, add the following property:

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

14. Click **Add** to add the property.

15. Click **Save All** to save the changes.
16. In your AEM author service, click **Adobe Experience Manager**.
17. Click the **Navigation** icon on the left panel and click **Sites**. The **Sites** console opens.
18. Click the check box beside the **we-train** page and click **Properties (*p*)** as shown. The **Properties** page opens.



19. Ensure you are on the **Basic** tab and enter **We.Train** in the **Title** box.
20. On the **Advanced** tab:
    a. In the **Redirect** box, enter **/content/we-train/en**.
    b. Under **Template Settings** > **Allowed Templates,** click **Add**.
    c. Enter **/conf/we-train/settings/wcm/templates/.*** in the **Allowed Templates** box.
21. Click **Save & Close**. **The form has been submitted successfully** message appears.

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

22. Refresh **CRXDE lite**, navigate to the **/content/we-train/jcr:content** node and observe the properties that you just added to the content root.

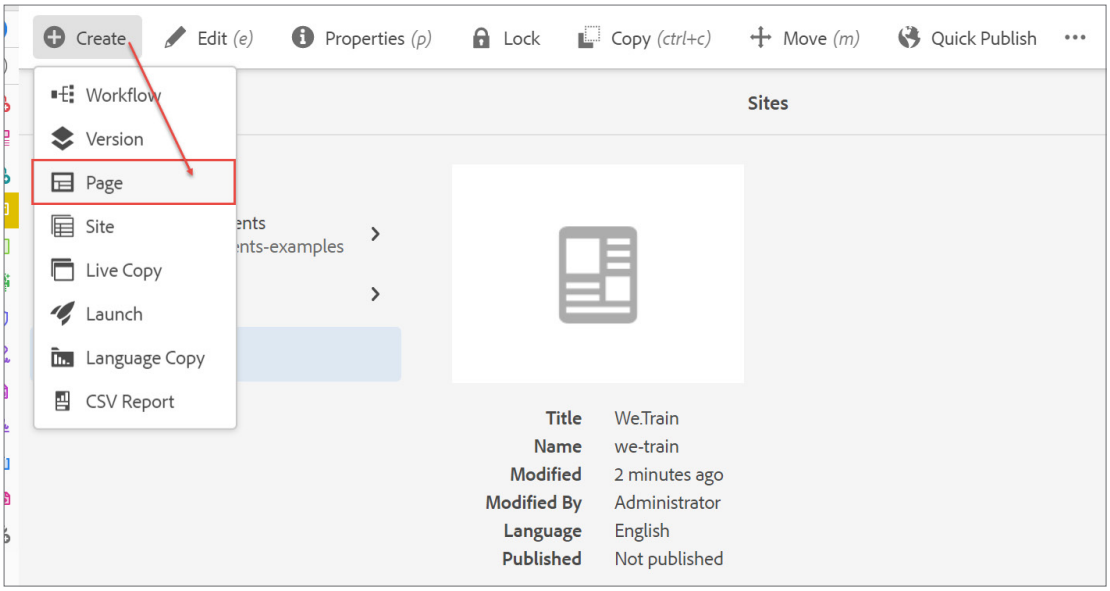
Task 2: Create a site structure

You have enabled and published the Content Page template and created a content root. Next, you will 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

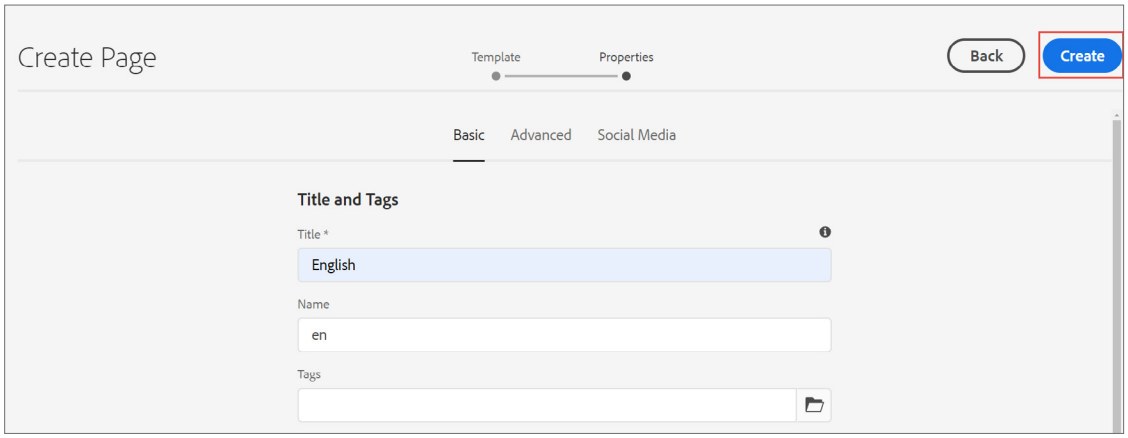
To create a page:

1. Click http://localhost:4502/sites.html/content. The **Sites** console page opens.
2. Select the **We.Train** check box and click **Create** > **Page** from the actions bar, as shown. The **Create Page** page opens.

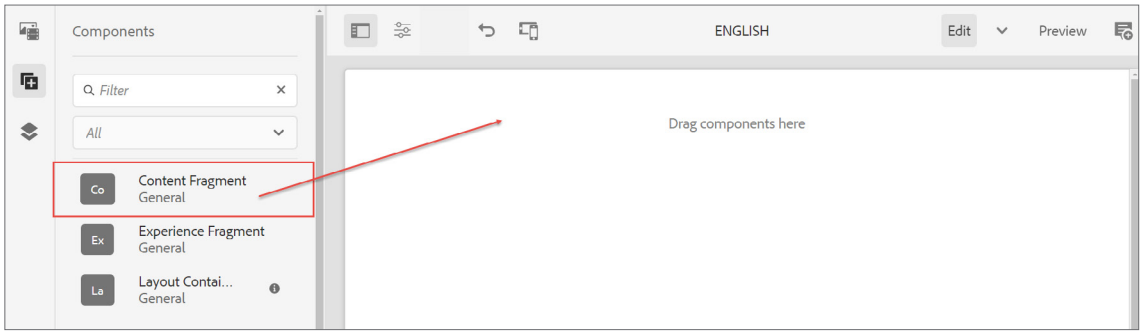


3. Select the **Content Page** template and click **Next**. The **Properties** page opens.

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



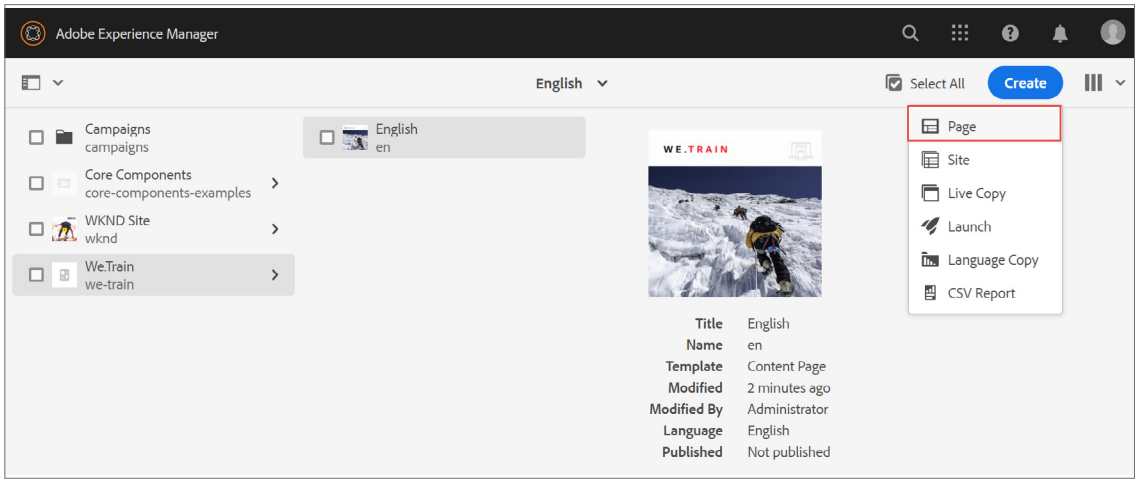
5. Click **Open**. The page opens on a new tab of your browser.
6. Ensure **Edit** is selected on the page toolbar.
7. Click the **Toggle Side Panel** icon from the page toolbar. The side panel opens.
8. Click the **Components** panel. All available components for the page are displayed in the Components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



To create a subpage:

10. Ensure you are in **Sites** console.
11. Navigate to the **We.Train** > **English** page.

12. Click **Create** from the actions bar and select **Page** from the drop-down menu, as shown. The **Create Page** page opens in the **Properties** page.

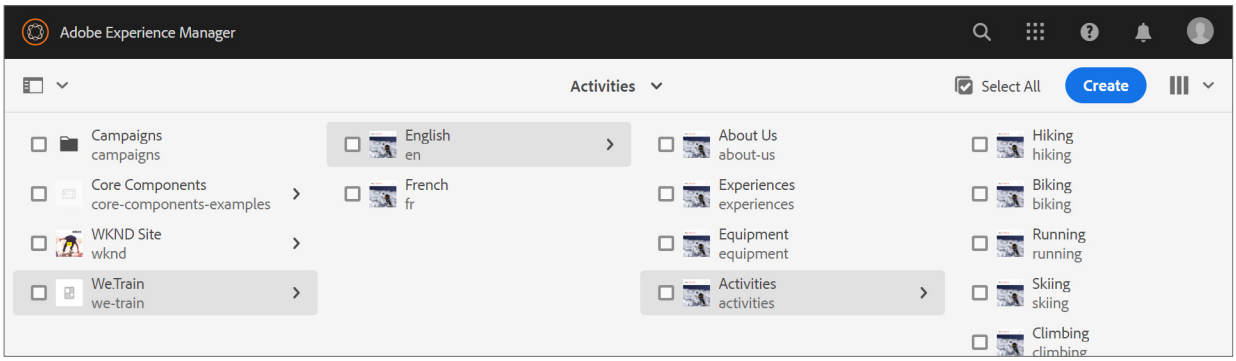


13. Select the **Content Page** template and click **Next**. The **Create Page Properties** page opens.
14. Enter **About Us** in the **Title** box and **about-us** in the **Name** box, and then click **Create**. The **Success** dialog box opens.
15. Click **Done**. A subpage of **English** is created.
16. Perform steps 11 through 15 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)

Best Practice: 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:



17. (Optional) You can upload a thumbnail. Notice that the content root page (/content/we-train) does not have a thumbnail in the Sites console. Add a thumbnail to the page by selecting the **We.Train** page, clicking **Properties**, and uploading 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 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 will 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 page style to the Style system

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 way
- 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 and performance of a site

Structure of Client-Side Libraries

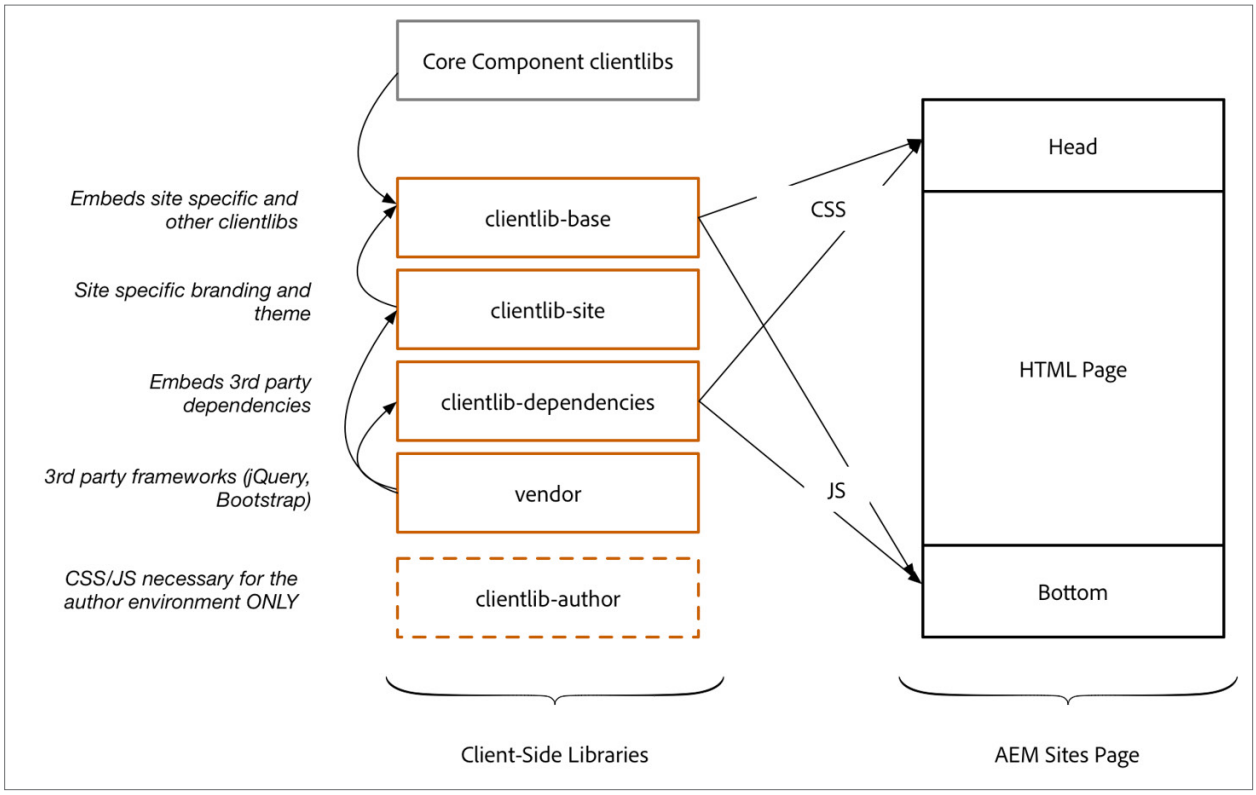
A client-side library folder is a repository node of type **cq:ClientLibraryFolder**. The typical node structure of a client library in AEM is:

```
[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 by using the following properties:

- categories: This property helps identify the categories into which the set of JS and/or CSS files within the cq:ClientLibraryFolder belong. The categories property is multi-valued and enables a library folder to be a part of more than one category.
- dependencies: This property provides a list of other client library categories on which the library folder depends. For example, given two cq:ClientLibraryFolder nodes F and G, if a file in F requires another file in G 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, the allowProxy 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 this convention 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 at the site-level:

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

Component-Level

The clientlibs at 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 templates through `data-sly-use`. The three templates, css, js, and all can be called through `data-sly-call`:

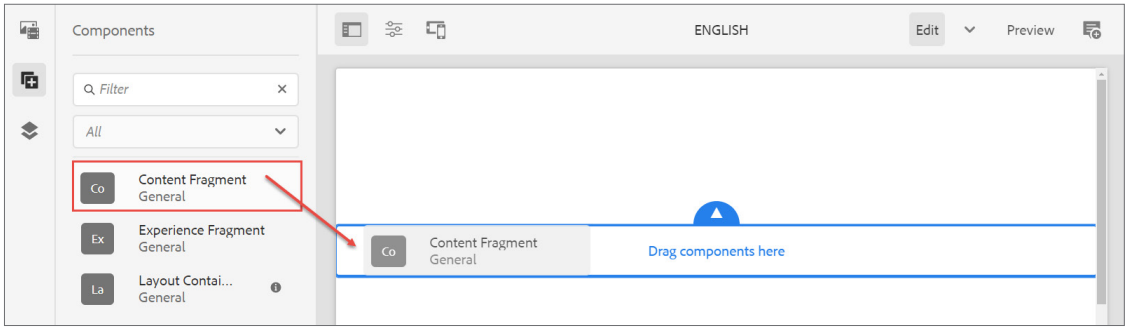
- css: Loads only the CSS files of the referenced client libraries.
- js: Loads only the JavaScript files of the referenced client libraries.
- all: Loads all files of the referenced client libraries (both CSS and JavaScript).

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 (CSV) list.

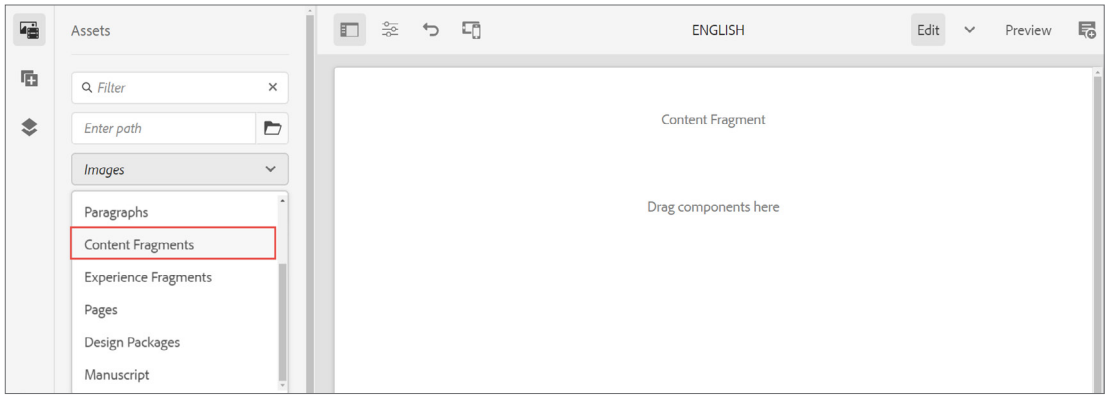
Exercise 1: Add Content to a page

In this exercise, you will add some content to a page. After completing this exercise, you will see the content without a design. In later exercises, you will apply the We.Train design and you will see the content transform to the We.Train design standards.

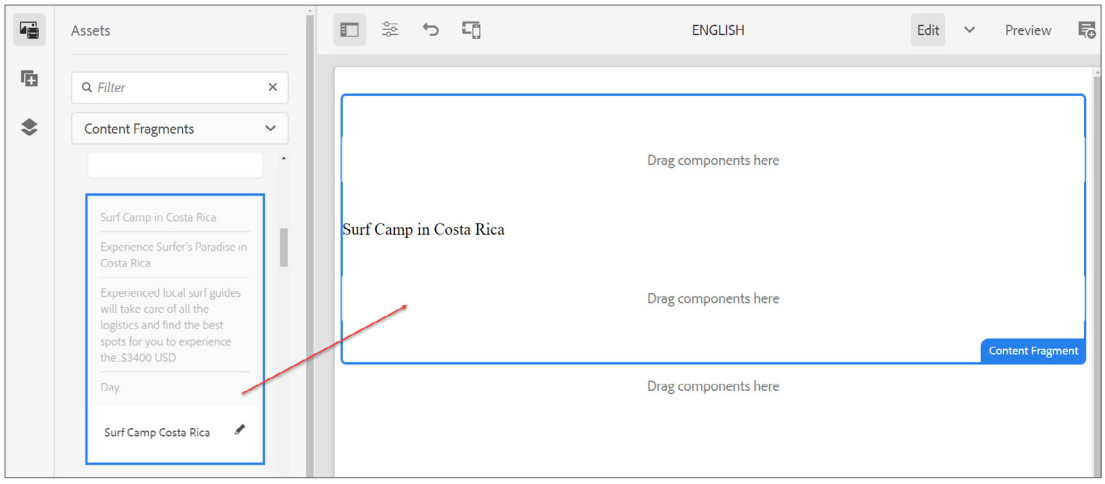
1. In your AEM author service, navigate to **Sites** > **We.Train** > **English** page.
2. Click the **English** page check box and click **Edit (*e*)** on the header bar to open the English page.
3. On the left panel, click the **Components** icon. The available components are displayed.
4. Drag the **Content Fragment** component onto the layout container, as shown.



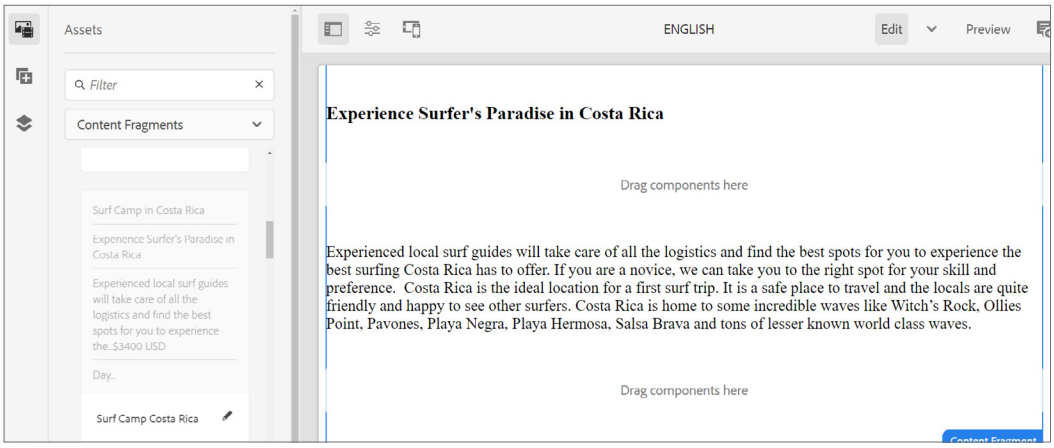
5. On the left panel, click the **Assets** icon. The available assets are displayed.
6. Filter the assets based on Content Fragments by clicking the **Images** drop-down menu and selecting **Content Fragments**, as shown:



7. Drag the **Surf Camp Costa Rica** fragment onto the **Content Fragment** component, as shown. Notice how the content has no design currently.



8. Click the Content Fragment and select the **Configure** icon (wrench icon). The Content Fragment dialog box opens.
9. From the **Element** drop-down menu, select **Description**.
10. Click **Done**. Notice that you can now view the description in the Content Fragment.



Exercise 2: Create a client-side library

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

1. Navigate to **Adobe Experience Manager** > **Tools** > **CRXDE Lite**. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training** folder**.**
3. Right-click the **training** folder and click **Create** > **Create Folder**. The **Create Folder** dialog box opens.
4. Type **clientlibs** in the **Name** box and click **OK**. The folder is created under the training folder.
5. Click **Save All**.
6. Right-click the **clientlibs** folder and click **Create** > **Create Node**. The **Create Node** dialog box opens.
7. Update the following fields:
   a. **Name**: **clientlib-base**
   b. **Type**: **cq:ClientLibraryFolder**
8. Click **OK**. The **clientlib-base** node is created.
9. Click **Save All** from the actions bar.

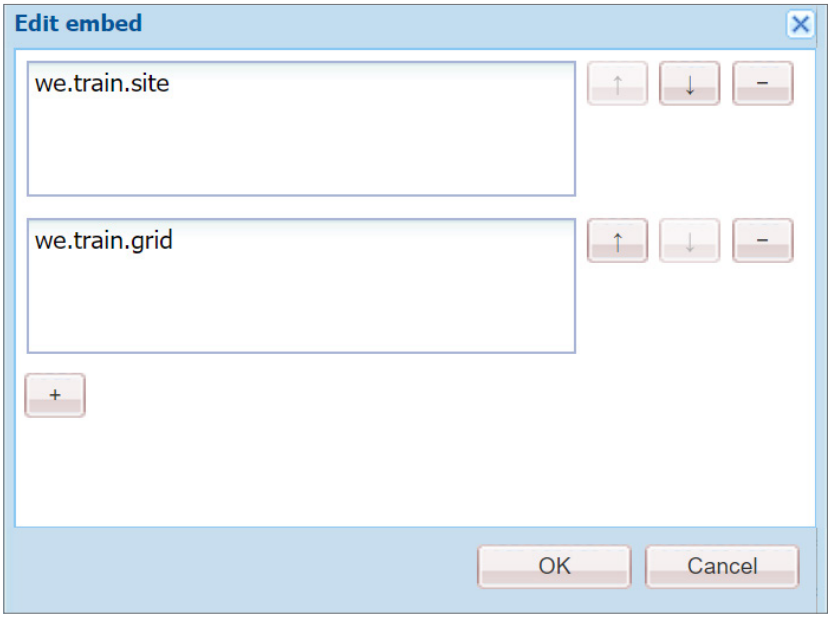
10. Select the **clientlib-base** folder and add the following properties:

Note: To add the below properties, you must enter the property, click **Add**, and then click **Save All.** Follow the same procedure for all the properties.

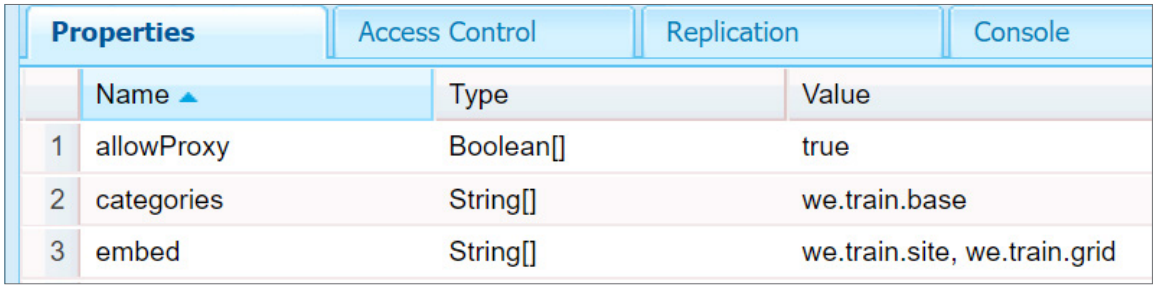
Note: To enter a String Array (String[]), select **String** from the **Type** drop-down menu and then click the **Multi** button. Remember, the **Multi** button will be selected until you deselect it. If the **Edit** dialog box appears with the string array information, click **OK**.

Name	Type	Value
categories	String[]	we.train.base
embed	String[]	we.train.site we.train.grid
allowProxy	Boolean	true

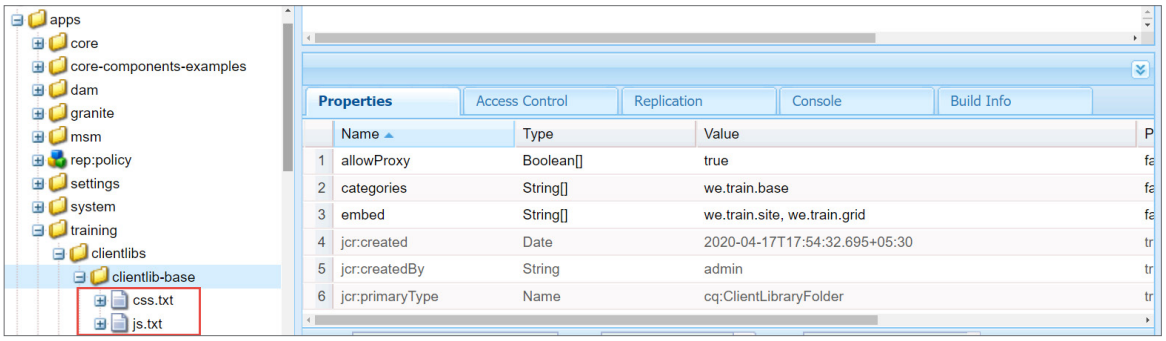
When updating the **embed** property, the **Edit embed** dialog should look as shown:



11. Click **Save All** from the actions bar. The properties should look as shown:



12. Right-click the **clientlib-base** node and click **Create** > **Create File**. The **Create File** dialog box opens.
13. Enter **css.txt** in the **Name** box and click **OK**. The file is created under the clientlibs folder.
14. Click **Save All** from the actions bar.
15. Perform steps 12 through 14 to create the **js.txt** file.



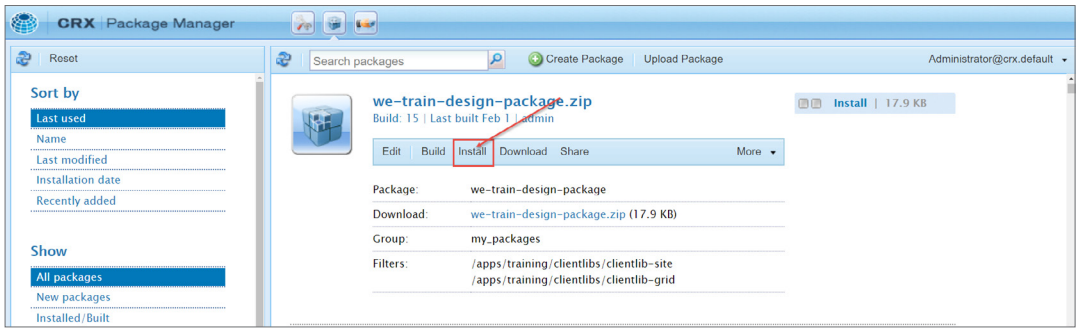
Exercise 3: Install the We.Train design

In this exercise, you will install the we-train-design-package to the JCR. This content package contains the We.Train design that you will use throughout the course. The We.Train design is built to compliment the WKND design, which you will add later.

To install the we-train-design-package:

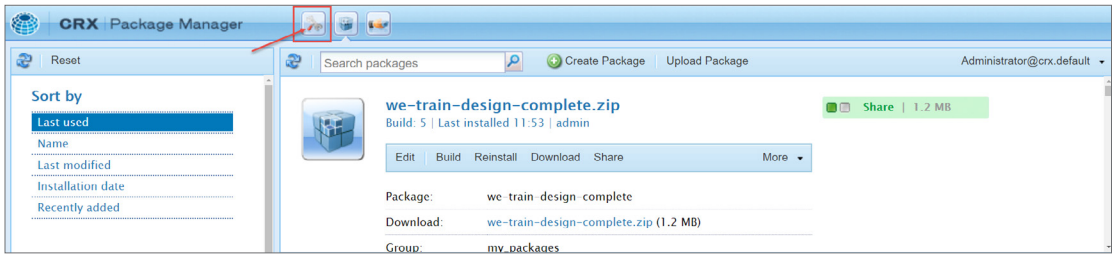
1. From **CRXDE Lite**, click the **Package** icon from the header bar, as shown. The **Package Manager** page opens.

2. Click **Upload Package** from the actions bar. The **Upload Package** dialog box opens.
3. Click **Browse**.
4. In the Exercise Files folder provided to you, navigate to the **training-files/clientlibs** folder and double-click the **we-train-design-package.zip** package.
5. Click **OK** in the **Upload Package** dialog box. The package is uploaded.
6. 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 dialog box shows the files and folders that will be installed in your **/apps/training/clientlibs** folder on JCR.

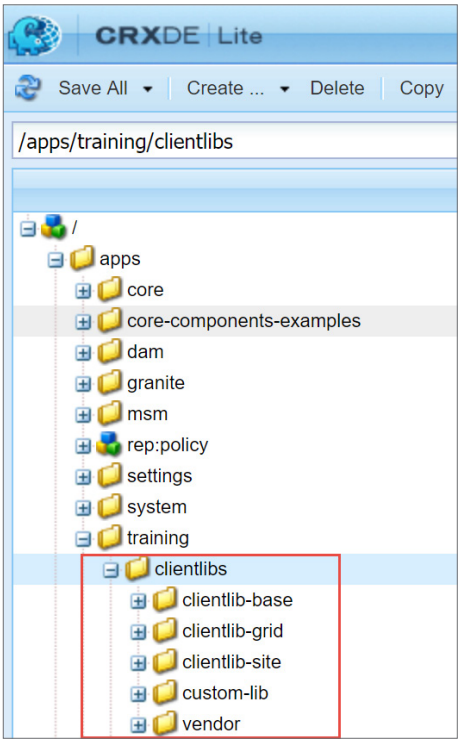


7. Ignore the **Advanced Settings** and click **Install**. The package is installed on your instance.

8. Click the **Develop** icon from the header bar, as shown. The **CRXDE Lite** page is refreshed.



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



Exercise 4: Add a client-side library to a page component

In this exercise, you will include the client libraries directly on the page component with scripts called customheaderlibs.html and customfooterlibs.html. These scripts enable a developer to hard code a mandatory design to the page component.

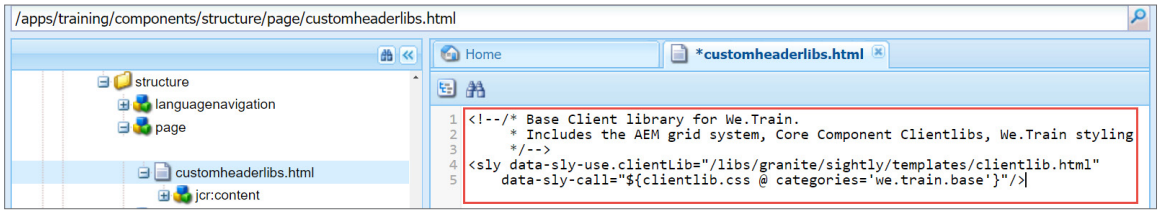
1. Click http://localhost:4502/crx/de/index.jsp if you are not already in **CRXDE Lite**. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/clientlibs/clientlib-base** node. Observe the property **categories=we.train.base**. You will use this category name when adding the clientlib to a page component.

To add the HTML files:

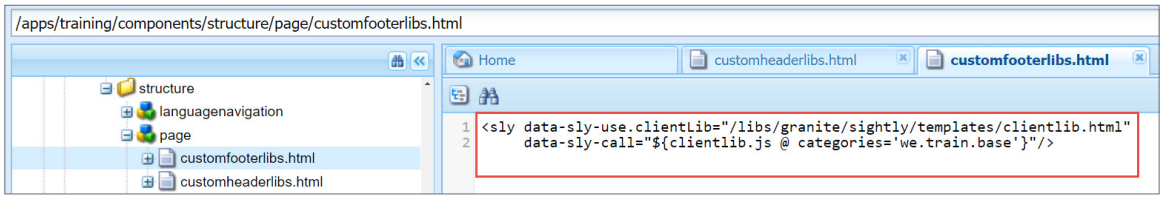
3. In **CRXDE Lite**, navigate to the **/apps/core/wcm/components/page/v2/page** component.
4. Right-click the **customheaderlibs.html** file and click **Copy**.
5. Navigate to the **/apps/training/components/structure/page.**
6. Right-click the **page** node and click **Paste** to paste the file you copied in step 4.
7. Similarly, copy the **customfooterlibs.html** file from **/apps/core/wcm/components/page/v2/page** folder and paste it to the **/apps/training/components/structure/page** node.
8. Click **Save All**.

To add code to the .html files:

9. In the Exercise Files folder provided to you, navigate to the ui.apps/src/main/content/jcr_root/apps/training/components/structure/page folder.
10. Open the **initial_customheaderlibs.html** file in Notepad++ (or any editor of your choice), copy the content of the file and paste it in the **customheaderlibs.html** editor in **CRXDE Lite**, as shown:



11. Click **Save All**.
12. Repeat step 10, copy the script from **initial_customfooterlibs.html** and update it in the **customfooterlibs.html** editor in **CRXDE Lite**.

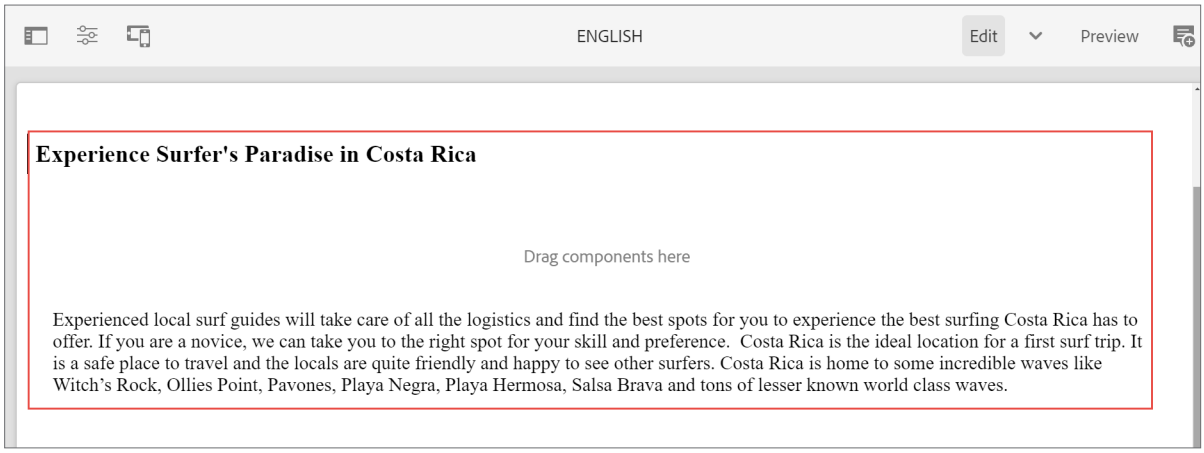


13. Click **Save All**.

You can verify if the CSS and JavaScript client libraries were added to the top and bottom of the page from the HTML source of a page.

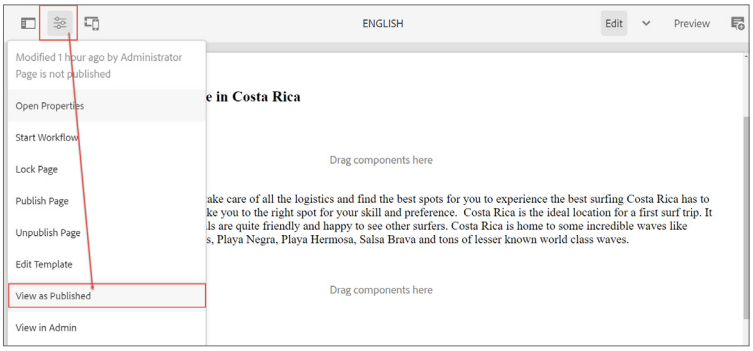
14. Click **CRXDE Lite** from the header bar. You will be taken to the AEM **Navigation** page.
15. Click **Sites** > **We.Train** site.
16. Click the check box on the **English** page and click **Edit** ***(e)*** from the actions bar. The English page opens on a new tab.

On the English page, you should see that a different design is applied to the content fragment that you added earlier.



To verify that the client libraries are being loaded:

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



18. Right-click on the English page and click **View page source**. The source code opens on a new tab of the browser.

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

```
1
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     <meta name="viewport" content="width=device-width, initial-scale=1"/>
11
12
13
14
15
16 <link rel="stylesheet" href="/etc.clientlibs/training/clientlibs/clientlib-base.css" type="text/css">
17
```

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

```
110
111
112 <script type="text/javascript" src="/etc.clientlibs/training/clientlibs/clientlib-base.js"></script>
113
114
115
116
```

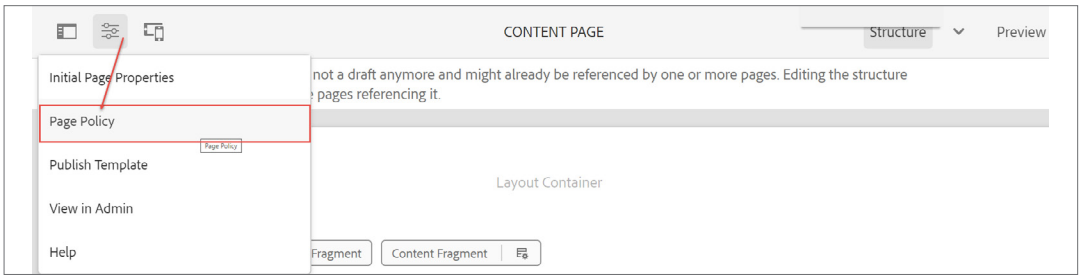
Exercise 5: Add a page style to the Style system

At times, an author needs to incorporate an optional design in a page. You can provide this flexibility to an author with the Style System in content policies. In this exercise, you will create a style at the page level. In later exercises, you will create styles at the component level.

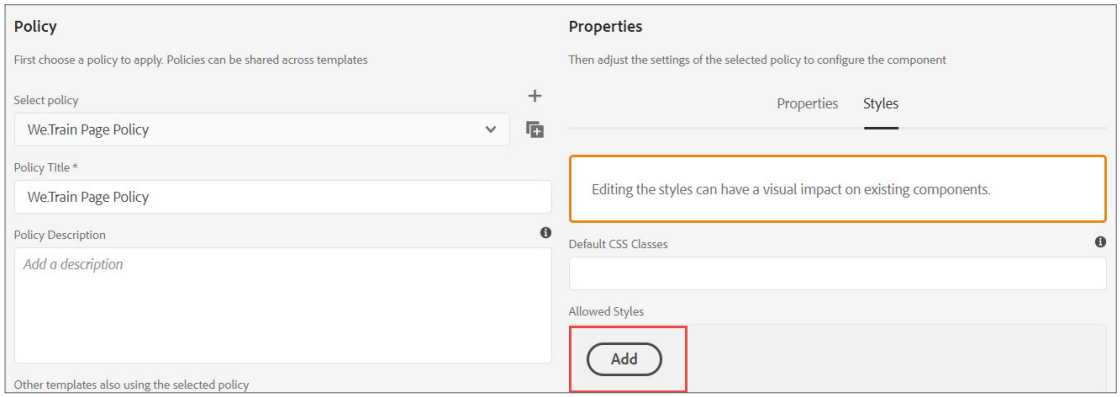
1. In **CRXDE Lite**, navigate to **/apps/training/clientlibs/clientlib-site/components/page/styles/page-center.less.**
2. Double-click the **page-center.less** file to open it. The editor opens on the right. Notice the class name, **wetrain-page--center**.
3. In the AEM author service, navigate to **Sites** > **We.Train** > **English**.
4. Select the **English** page and click **Edit *(e)***. The page opens on a new tab of the browser.
5. Click **Page Information** > **View as Published**. The page opens on a new tab.
6. Right-click on the page and click **View page source**. Notice the **wetrain-page--cente**r class is not rendering currently.

To add this class to the page and render the css, you can use the Style System to optionally add the class to a page based on an author's need.

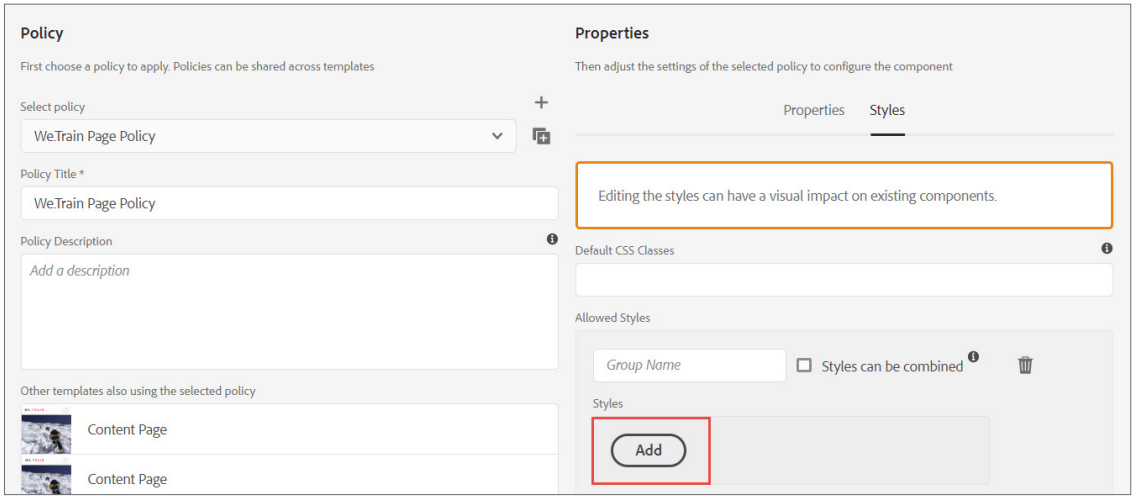
7. Navigate to **Adobe Experience Manager** > **Tools** > **Templates**.
8. Click the **We.Train** folder to open it.
9. Hover the cursor over the **Content Page** template and click the **Open** icon (pencil icon). The **Template editor** opens.
10. Click **Page Information** > **Page Policy**, as shown. The **Page** wizard opens



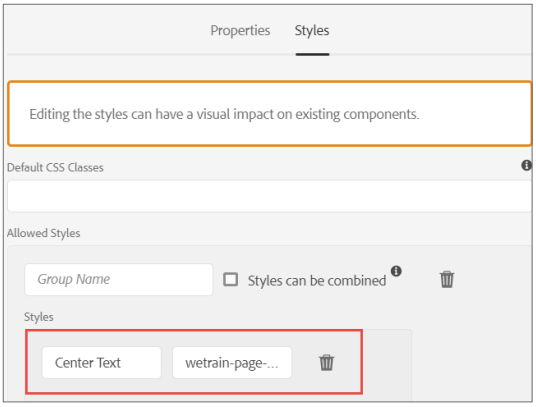
11. In the **Policy** section, in the **Policy Title** box, type **We.Train Page Policy**.
12. In the **Properties** section, on the **Styles** tab, click **Add** under **Allowed Styles**, as shown. The **Styles** section appears.



13. Click **Add** under **Styles**, as shown. The **Style Name** and **CSS Classes** boxes appear.

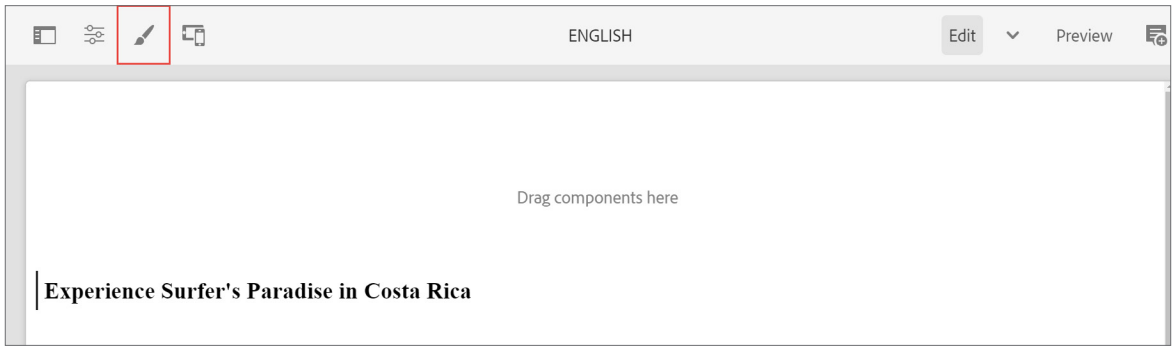


14. In the **Style Name** box, type **Center Text**.
15. In the **CSS Classes** box, enter **wetrain-page--center**
16. Click **Add**. The style is updated, as shown:

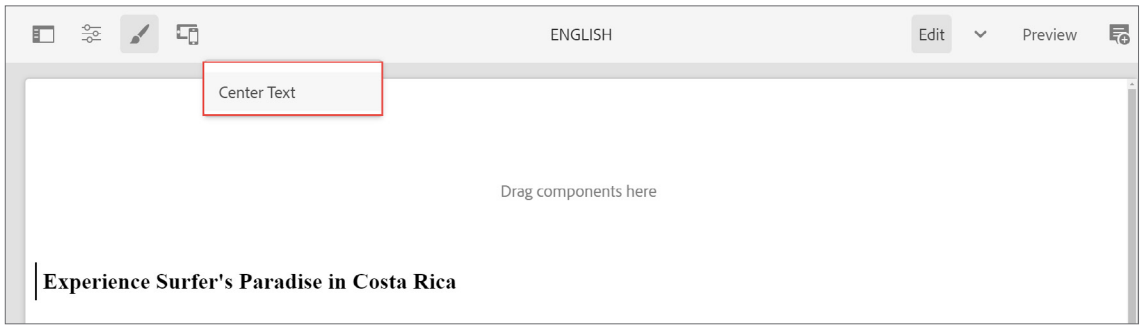


17. Click **Done**.

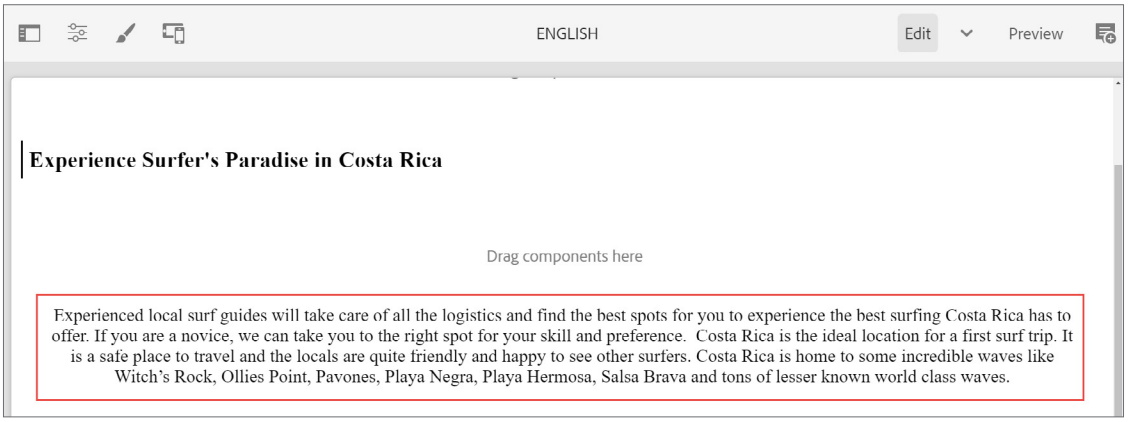
18. Navigate to the **We.Train** > **English** page. Notice there is a **Styles** icon (paintbrush icon) on the toolbar, as shown:



19. Ensure the Content Fragment is selected, click the **Styles** icon and select **Center Text**, as shown:



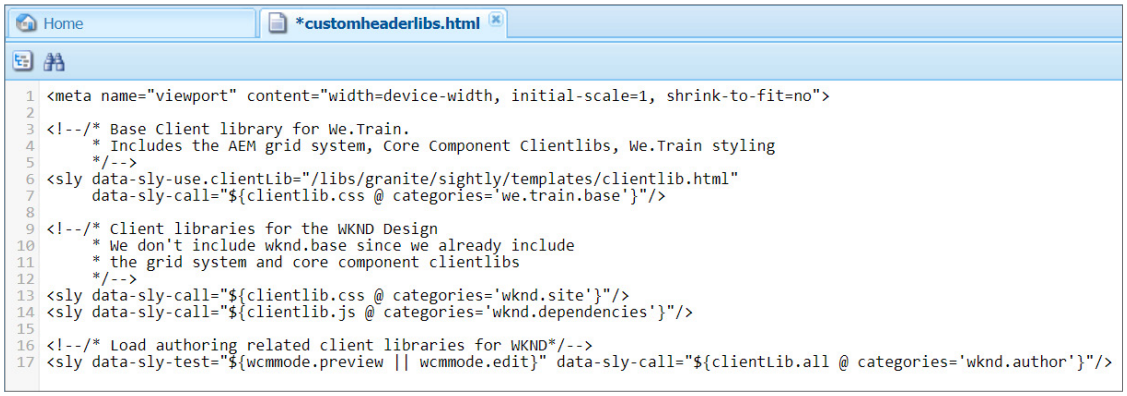
Notice that the description in content fragment is center aligned now.



Exercise 6: Apply the WKND design

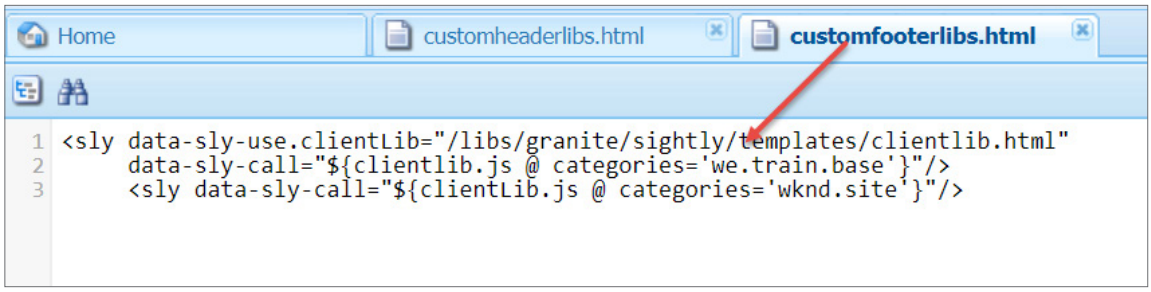
In this exercise, you will incorporate the WKND design to the training design. The WKND client libraries are modular with less code. (You should have already installed the WKND site in your AEM service. If you have not done this yet, refer the Technical Basics module and install the site.)

1. In **CRXDE Lite**, navigate to **/apps/wknd/clientlibs**. Notice the different client libraries WKND uses. Notice, especially, the category names for each client library.
2. Navigate to **/apps/training/components/structure/page**.
3. Double-click **customheaderlibs.html**. The editor opens on the right.
4. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/structure/page** folder.
5. Open the **wknd_customheaderlibs.html** file in Notepad++ (or any editor of your choice), copy the content of the file and replace the content in **customheaderlibs.html** editor in **CRXDE Lite** with the new content.



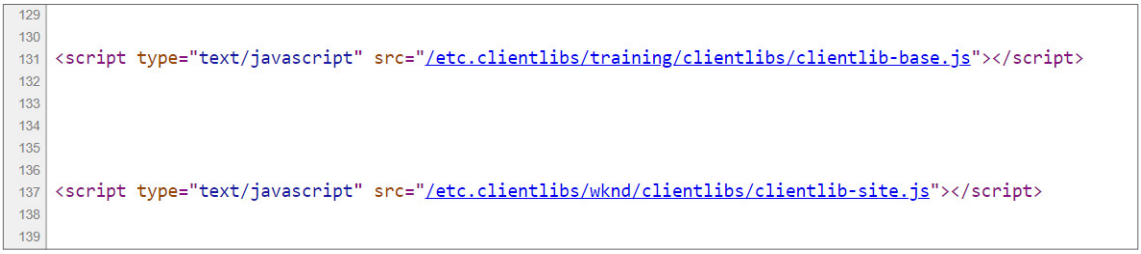
6. Click **Save All**.
7. Double-click **customfooterlibs.html**. The editor opens on the right.
8. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/structure/page** folder.

9. Open the **wknd_customfooterlibs.html** file in Notepad++ (or any editor of your choice), copy the content of the file and replace the content in the **customfooterlibs.html** editor in **CRXDE Lite** with the new content.



10. Click **Save All**.
11. Refresh the English page. Notice the design looks slightly different.
12. To verify that the design is incorporated, click **Page Information** > **View as Published**. The page opens on a separate tab of the browser.
13. Right-click on the page and click **View page Source**. You should now see **wknd clientlib-dependencies**, **wknd clientlib-site**, and **training clientlib-base** at the top of the source, as shown:

14. Scroll down to the bottom and notice the **wknd clientlib-base** java script is also loading at the bottom of the page.



Core Components: An Overview

Introduction

Adobe Experience Manager (AEM) provides a wide range of component implementations on a standard instance by default. Before working with components in AEM, you must understand the key concepts of a component and its types.

Objectives

After completing this module, you will be able to:

- Describe components
- Describe Core components
- Create proxy components
- Add Core component client libraries
- Describe the content policy
- Add proxy components to the template
- Add content policies to proxy components
- Author Core components

Components

In AEM, components:

- Are the structural elements that constitute the content of the pages being authored
- Are resource types
- Contain a collection of scripts that implement a specific functionality to present the content on the website
- 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
- Work anywhere within any AEM system
- Have a standardized user interface
- Have configurable edit behavior
- Use dialogs that are built using subelements 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)

Core Components

In AEM, Core components:

- Are production-ready components that Adobe provides as a base for site creation
- Provide robust and extensible base components built on the latest technology and best practices adhering to accessibility guidelines
- Are compliant with the Web Content Accessibility Guidelines (WCAG) 2.0 AA standard

You can find that the code on Core component development exists outside of AEM code base. This enables you to update and enhance the components more often than the product releases. Developers can extend core components to offer custom functionality.

Core Component Library

The following table describes the current list of Core components :

Core Component	Description
Page	Responsive page working with the template editor
Breadcrumb	Page hierarchy navigation
Navigation	Helps add a site navigation component that lists the nested page hierarchy
Language Navigation	Helps add a language and country switcher that lists the global language structure
Quick Search	Helps add a search component that displays the results as real-time suggestions on a drop-down menu
Title	Creates heading for a page or page sections
Text	Displays rich text paragraph on the page and provides various formatting options
Image	Helps add a single image asset to the page. Images are adaptive. The relevant image width is selected for the screen size, with lazy loading available
Teaser	Helps group an image, title, and description for promoting and linking to site content sections. One or more actions can also be defined
List	Displays a list of pages. They can be defined either dynamically—by search query, tags or from a parent page—or as a static list of items
Experience Fragment	Experience fragment component, written in HTL, renders an experience fragment variation
Content Fragment	Adds a content fragment asset its elements and its variations to a page
Content Fragment List	Renders a list of content fragments. Useful for authoring headless content that can be easily consumed by applications
Separator	Displays a horizontal line on the page for dividing sections of content
Carousel	Displays content panels on a page and helps create slides of varying component type
Social Sharing	Adds Facebook and Pinterest share links to the page. It is often included in page headers or footers
Tabs	Helps switch between panels of related content and create panels of varying component types
Form Container	Adds a responsive form paragraph system
Form Text	Adds a text input field
Form Options	Adds a multiple options input field
Form Hidden	Helps add a hidden input field
Form Button	Helps add a submit or custom button

Note: Visit the GitHub project for Core components to see the latest list of released Core components.

Features of Core Components

The Core components are:

- **Preconfigurable**: Helps page authors define how to use the Core components in templates
- **Versatile**: Helps authors create different types of content
- **Easy-to-use**: Helps authors create and manage content efficiently
- **Production-ready**: Delivers high performance and are tested comprehensively
- **Accessible**: Complies to the WCAG 2.0 standard, provides ARIA labels, and supports keyboard navigation
- **Easy to style**: Implements the Style System, and the markup follows Block Element Modifier (BEM) Cascading Style Sheets (CSS) naming
- **Search Engine Optimization (SEO) friendly**: Provides semantic HTML output and schema.org microdata annotations
- **Extensible**: Helps extend the functionality to meet the custom needs without starting from the beginning
- **Open Source**: Helps developers contribute improvements on GitHub (Apache License)
- **Versioned**: Helps developers and administrators determine the compatibility of Core components with their AEM version

---

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

Proxy Components

Proxy components define the required component name and group to display to page authors and refer to a Core component as their super type. 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 Core components in the template editor.

Best Practice: Create proxy components from Core components, and use the proxy components in your project.

After creating proxy components, you will notice that most components are nodes with properties to make them unique proxy components. Some proxy components have node structures for further customization. The table below describes some of the node structures.

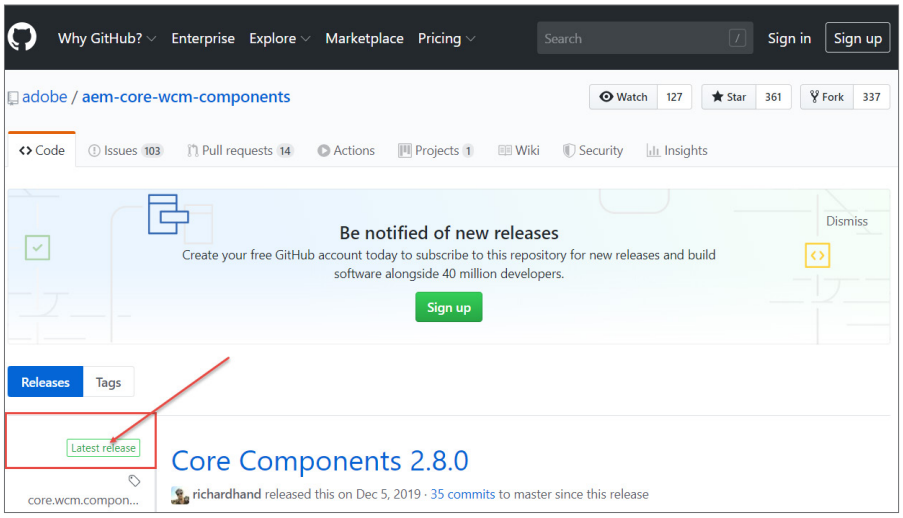
Components	Node Structure
Carousel component	cq:template: Represents the initial setup of the carousel. This node makes it easier for an author to understand how the component works with two example items.
Image component	cq:editConfig: Determines how AEM renders the image that is dragged onto the component.
Tabs component	cq:template: Represents the initial setup for tabs.
Teaser component	imageDelegate property: Delegates the image rendering to the core image component.
Forms Container component	new: Inserts a layout container so you can add form components to the container.

Note: The examples above are specific to each component in their respective row, but the nodes could be applied to any proxy component depending on the design requirements of the proxy component.

(Optional) Exercise: Install latest Core components

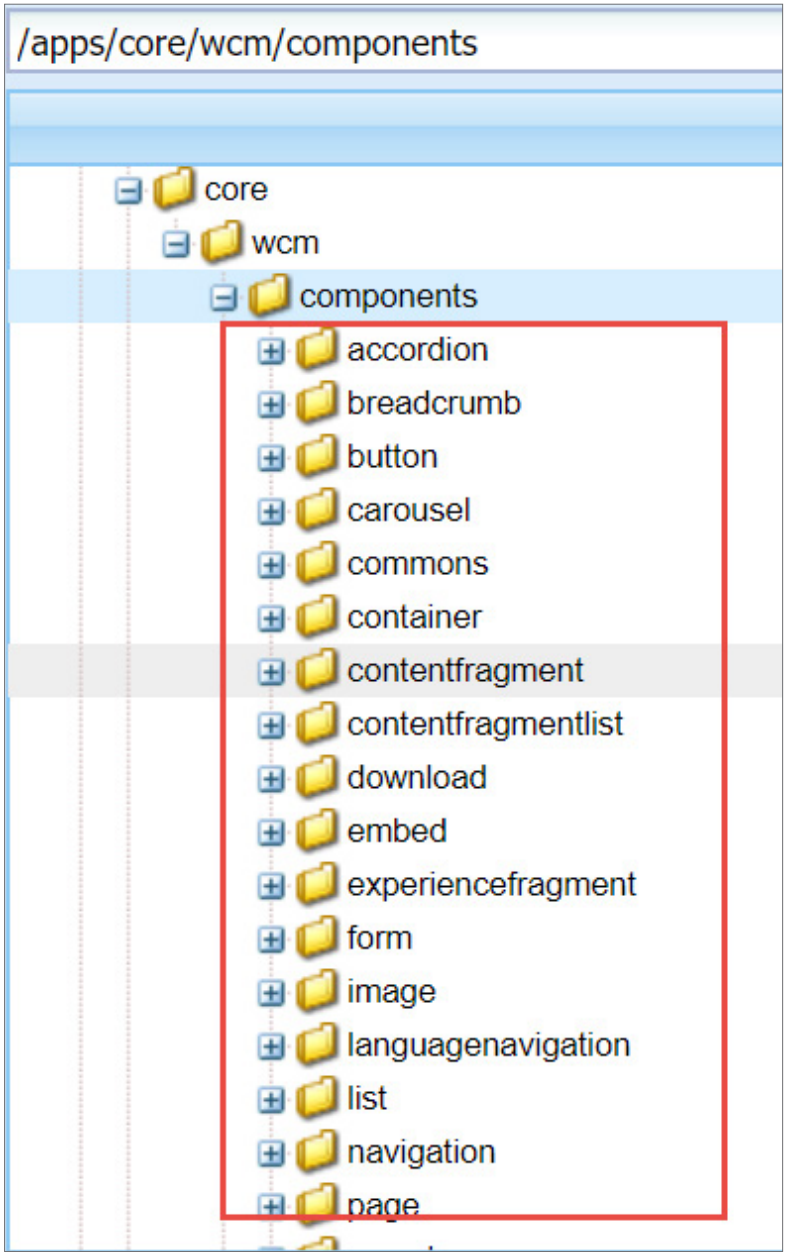
The WKND project you installed earlier included a version of Core components. At any time, you can update your AEM instance to include the latest Core components. Though updating to the latest Core components is optional for your project, you should typically plan to update the Core component version routinely to ensure you have access to the latest functionality. The steps below shows how to manually install the latest Core components. In a real development project, you would update the version number in your maven pom file.

1. Open https://github.com/adobe/aem-core-wcm-components/releases in a browser.
2. Click **Latest release**, as shown, and scroll down to the **Assets** section.



3. Click **core.wcm.components.all-2.8.0.zip**. The package is downloaded.
4. Open CRX Package Manager by opening **http://localhost:4502/crx/packmgr/index.jsp** in a browser.
5. Click **Upload package**. The Upload Package dialog box opens.
6. Browse for the **core.wcm.components.all-2.8.0.zip** file you downloaded earlier and click **OK**. The package is uploaded.

7. Click **Install**. The **Install Package** dialog box opens.
8. Click **Install**. The package is installed.
9. In **CRXDE Lite**, navigate to **/apps/core/wcm/components** and observe the new components.



Exercise 1: Create proxy components

In this exercise, you will create proxy components for the Core components to be used in the training website. After completing this exercise, you can test the AEM site by using different components.

1. In **CRXDE Lite**, navigate to the **/apps/training/components/content** folder. (If the content folder does not exist, create it.)
2. Right-click the **content** folder and click **Create** > **Create Node**. The **Create Node** dialog box opens.
3. Update the following details:
   a. **Name**: **title**
   b. **Type**: **cq:Component**
4. Click **OK**. The **title** component is created.
5. Click **Save All** from the actions bar.
6. Select the **title** node and add the following properties:

Note: When adding the below properties, you must add the property, click **Add** and then click **Save All**. Follow the same procedure for all properties.

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

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

Note: In the next step, you will create the rest of the proxy components by installing a content package. Now that you know how to manually create a proxy component, if you would like to manually create these components, repeat steps 2 to 7 to add the following components:

/content/accordion
/content/contentfragment
/content/contentfragmentlist
/content/experiencefragment
/content/breadcrumb*
/content/carousel^*
/content/image^*
/content/list
/content/separator
/content/sharing
/content/tabs^*
/content/teaser
/content/text*
/form/button
/form/container*
/form/hidden
/form/options
/form/text
/structure/search*
/structure/languagenavigation

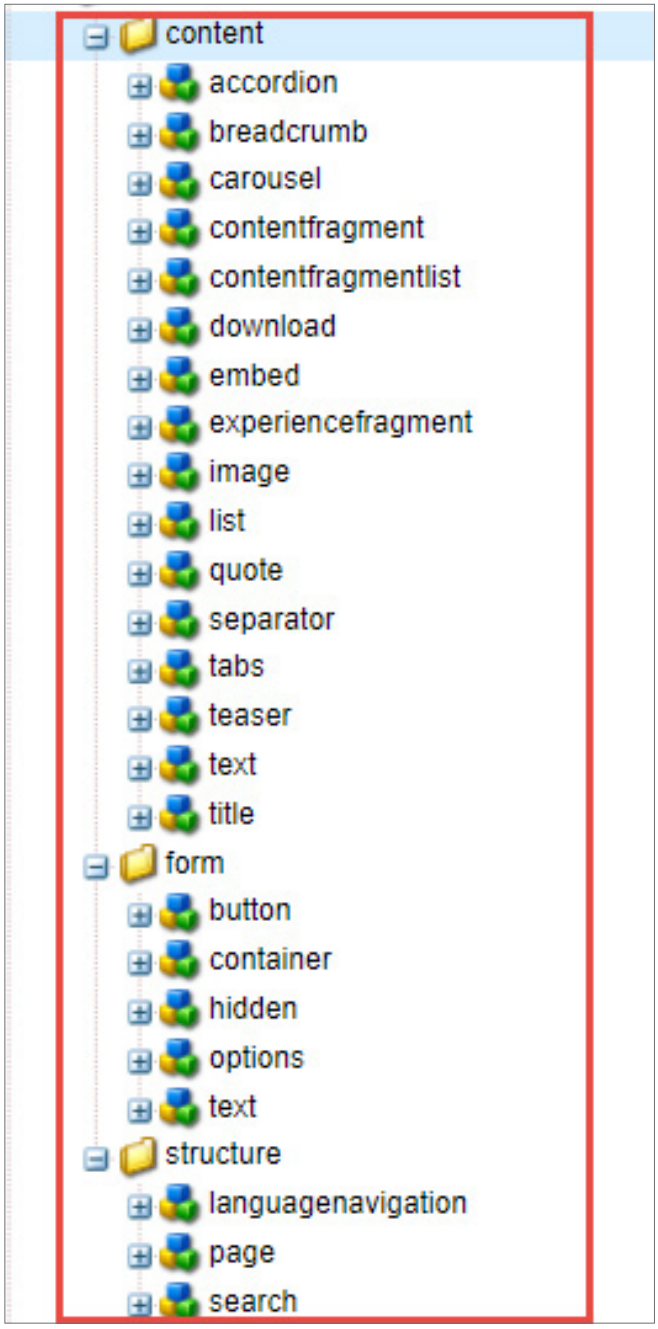
The Caret (^) symbol next to some of the components indicate that the components have additional node structures that are created with the content package.

The Asterisk (*) next to some of the components indicate that the components have an accompanying client library that must be added. You will add these in a later exercise.

8. To install the We.Train proxy components, click http://localhost:4502/crx/packmgr/index.jsp. The **CRXDE Package Manager** opens.
9. Click **Upload Package** from the actions bar. The **Upload Package** dialog box opens.
10. Click **Browse**.
11. In the Exercise Files folder provided to you, navigate to **training-files/core-components** and double-click the **we-train-proxy-components.zip** package.
12. Click **OK** in the **Upload Package** dialog box. The package is uploaded.

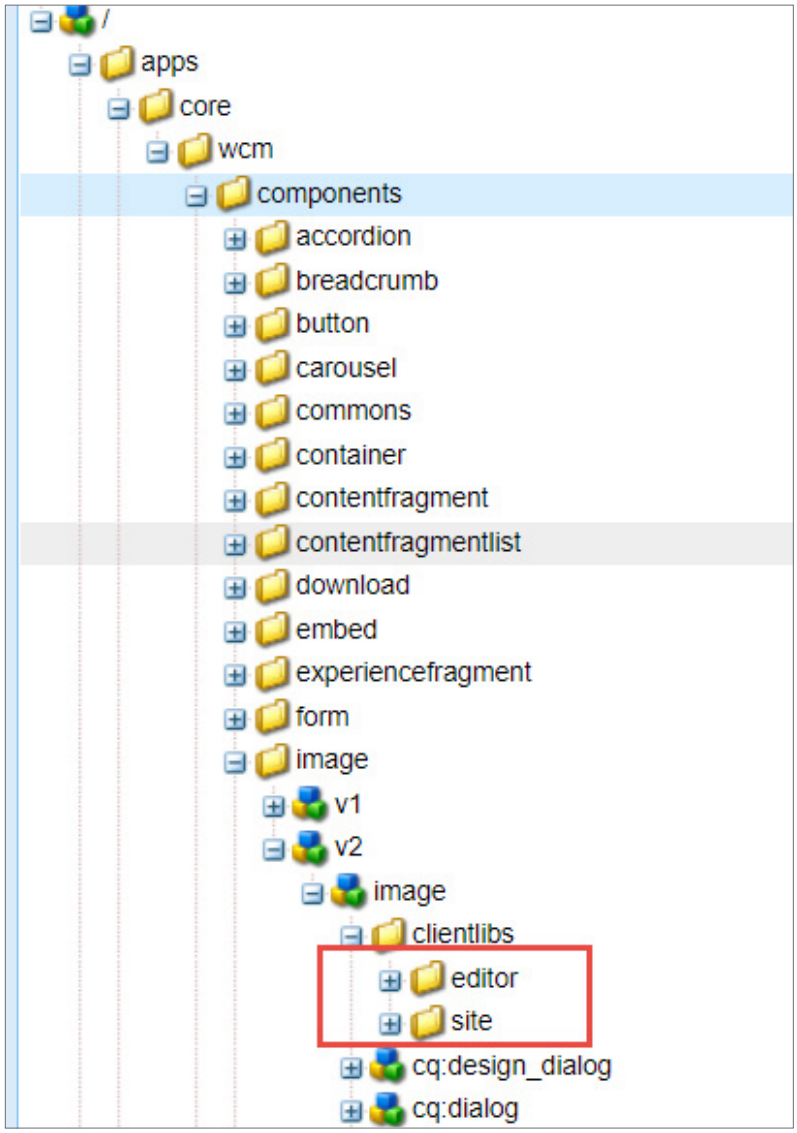
13. In the **we-train-proxy-components.zip** section click **Install**. The **Install Package** dialog box opens.
14. Ignore the **Advanced Settings** and click **Install**. The package is installed on your instance.
15. In **CRXDE Lite**, view the proxy components under:

- /apps/training/components/content
- /apps/training/components/form
- /apps/training/components/structure

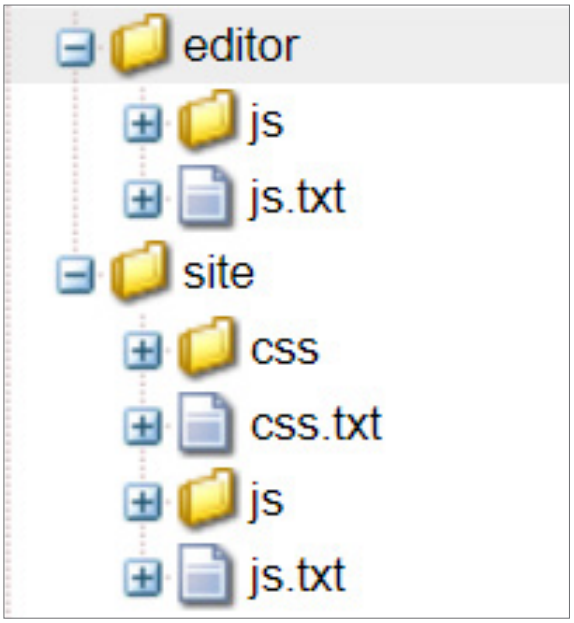


Client Libraries for Core Components

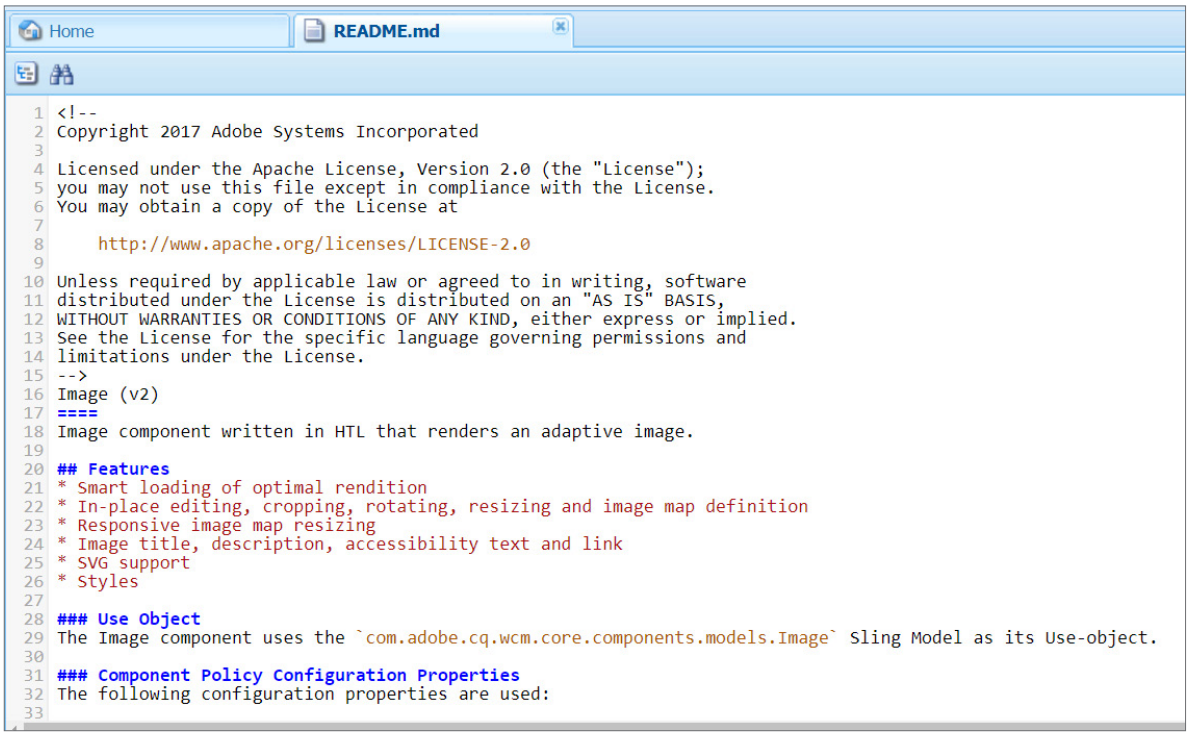
Several Core components have client libraries that must be included on the page for the component to function properly. For example, under `/apps/core/wcm/components/image/v2/image/clientlibs` notice, there are two client libraries—**editor** and **site**, as shown in the below screenshot:



The **editor** clientlib is meant only for the author environment and are typically included by the component dialog. The **site** clientlib must be included in the project's clientlib for the component to work properly.



For each Core component, you can view the `README.md` file to understand what each clientlib does. For example, navigate to `/apps/core/wcm/components/image/v2/image/README.md` to view the details about the Image Core component.

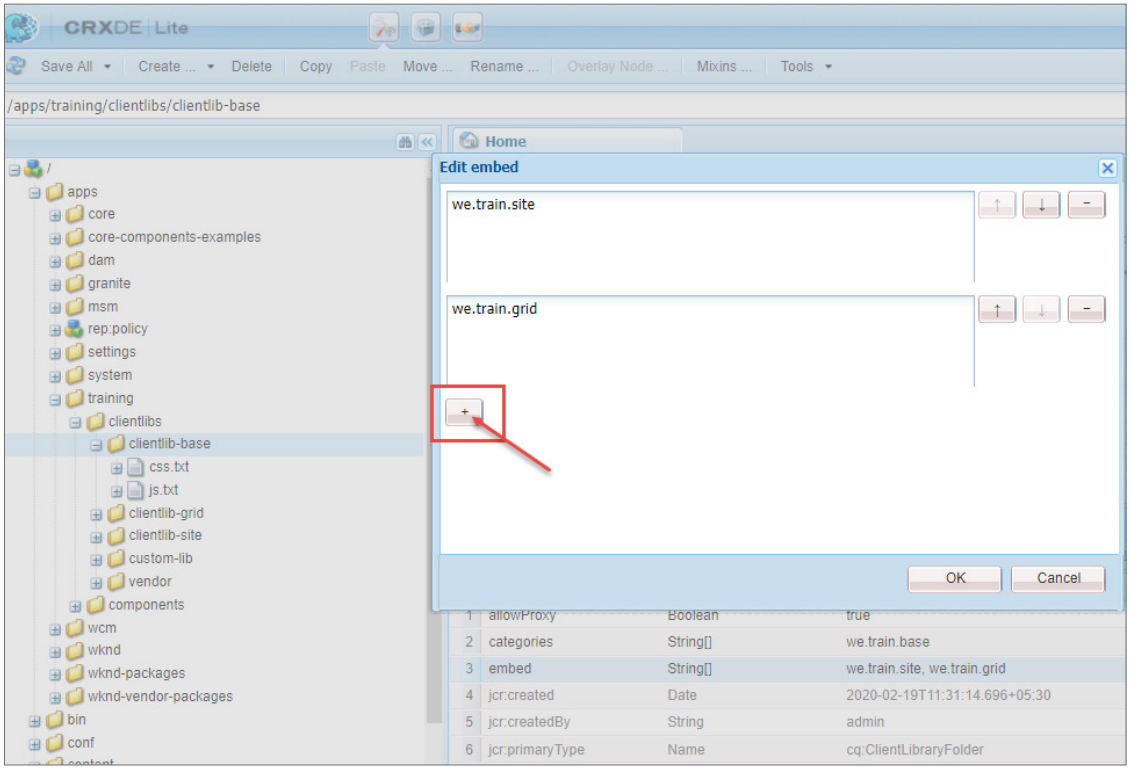


Note: Refer to the following link for more information on client libraries:
Update clientlib-base

Exercise 2: Add Core component client libraries

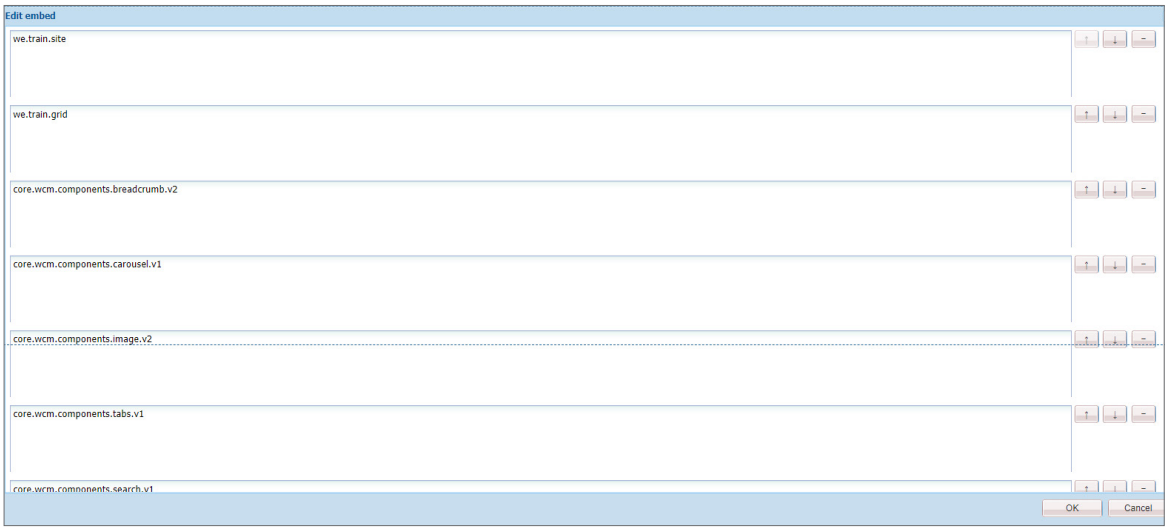
As you are using multiple Core components as proxy components, you must include the required CSS from the Core components. You will add these to the clientlib-base client library with the Core component category names.

1. In **CRXDE Lite**, navigate to **/apps/training/clientlibs/clientlib-base**.
2. Double-click the **embed** property. The **Edit embed** dialog box opens. You should see the clientlibs you added in a previous task.
3. To add more category names, click the plus (+) icon, as shown, and add the **core.wcm.components.breadcrumb.v2** category name.



4. Repeat step 3 to add the following category names as separate entries:

- core.wcm.components.carousel.v1
- core.wcm.components.image.v2
- core.wcm.components.tabs.v1
- core.wcm.components.search.v1
- core.wcm.components.form.text.v2
- core.wcm.components.form.container.v1
- core.wcm.components.accordion.v1



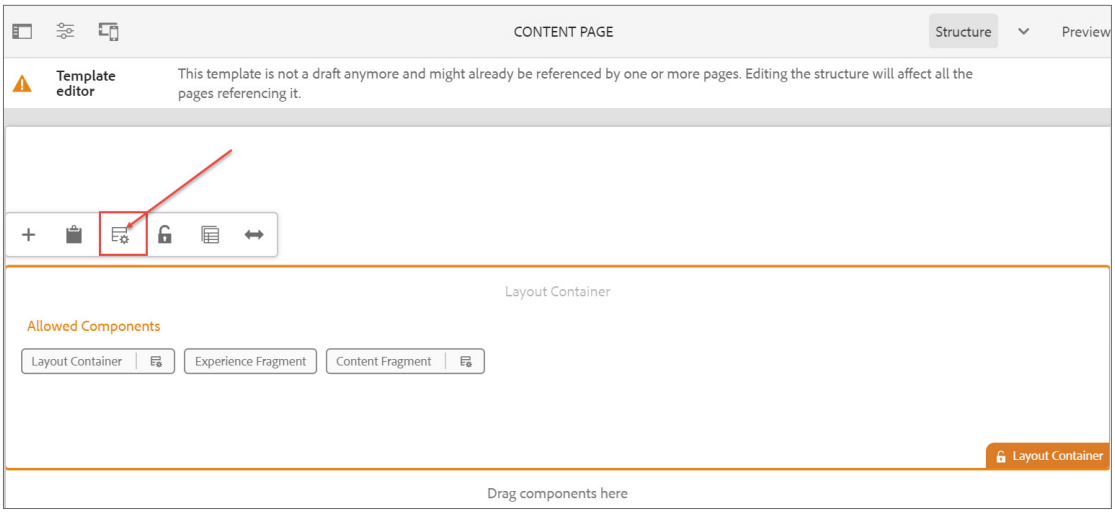
5. Click **OK**.
6. Click **Save All** to save the changes.

Exercise 3: Add proxy components to the template

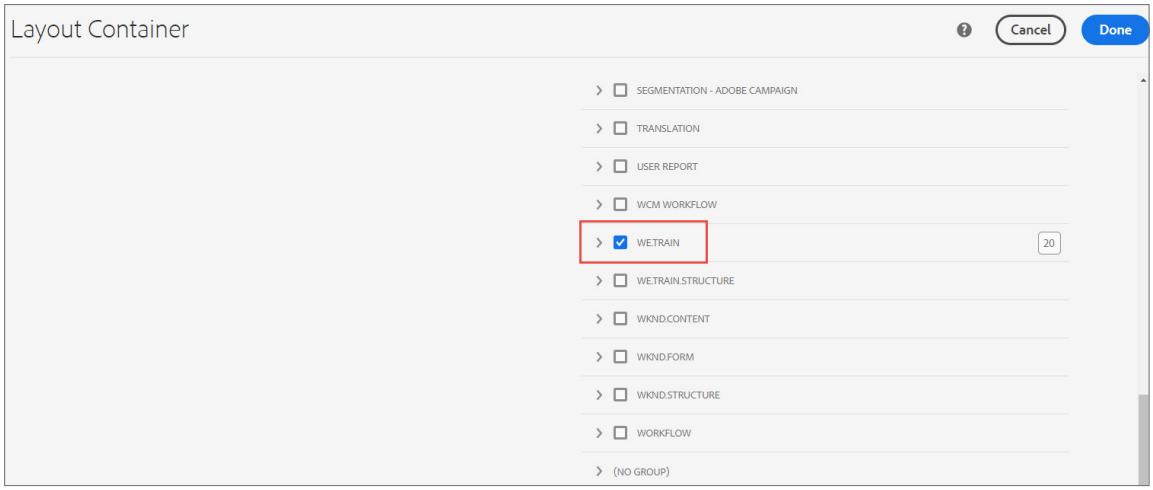
In this exercise, you will add the proxy components to the development template to help authors add components to a page. The page authors cannot edit the Layout Container [root] on the page. You will add a Layout Container to the template and specify the components that an author can use on the page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

To add proxy components to the template:

1. Click http://localhost:4502/aem/start.html.
2. Click **Adobe Experience Manager** > **Tools** > **Templates**. The **Templates** console opens.
3. Click the **We.Train** folder. The folder opens.
4. Select the **Content Page** template by clicking the **Select** icon (check mark). The template is selected.
5. Click **Edit (*e*)** from the actions bar. The template opens on a new browser tab.
6. Select the inner **Layout Container** and click the **Policy** icon from the component toolbar, as shown. The **Layout Container** wizard opens.

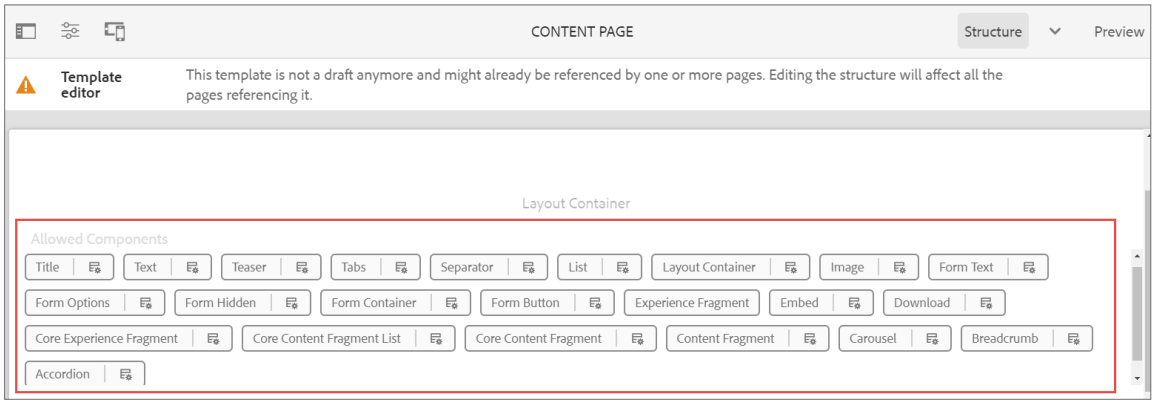


7. In the **Policy** section, ensure **We.Train Content Policy** is selected. You created this policy when you created the template earlier.
8. In the **Properties** section, scroll down and select the **WE.TRAIN** group check box, as shown:



Note: **WE.TRAIN** shows up because this is a filter for all valid components with the property componentGroup=We.Train. If you do not see this category, go back and double check your component properties.

9. Click **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.

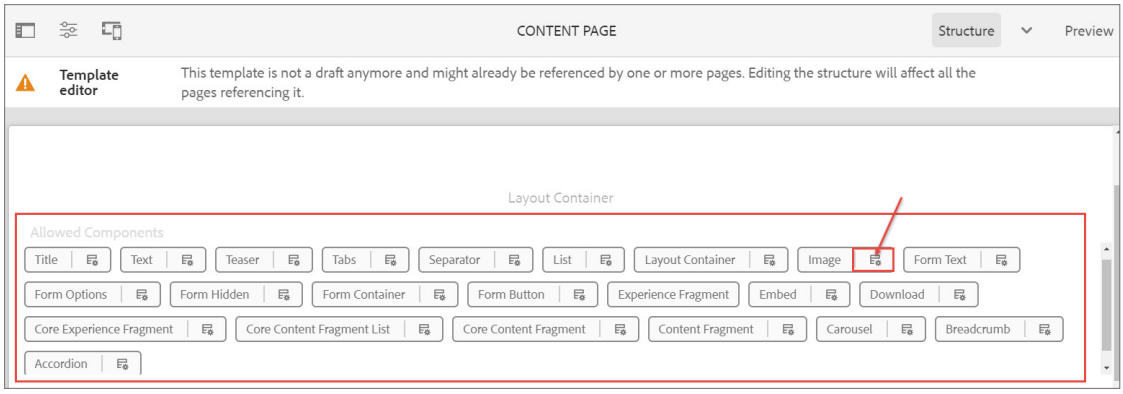


10. Navigate back to the **We.Train** > **English** page.
11. From the **Components** tab on the left, drag one of the components onto the **Drag components here** area of the English page. For example, the **Separator** component.
If you are able to add components to the page, you have successfully completed this exercise.

Exercise 4: Add content policies to proxy components

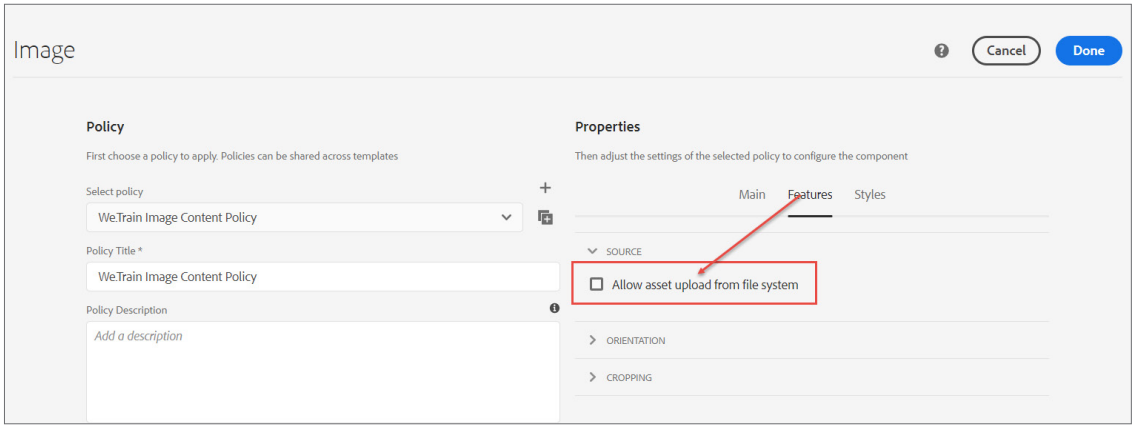
Now that you have proxy components for the Core components, you can make use of the built-in features of the proxy components. In this exercise, you will set up a content policy for the image component that will restrict uploading images from an author's local folder. Authors can only use images from the DAM.

1. Navigate to **Tools** > **Templates** > **We.Train** > **Content Page** in the AEM author service, if you are not already in it.
2. In the inner Layout Container (not the root Layout Container), notice that the proxy image component was automatically added. This is because the componentGroup is We.Train. Notice that there is a policy icon next to the component name. This means that the component has a configurable policy.
3. Click the **Policy** icon on the Image component, as shown:



4. On the **Policy** section, type **We.Train Image Content Policy** in the **Policy Title** box.
5. On the **Properties** section, ensure you are on the **Main** tab.
6. Under **Widths**, click **Add**, and enter **1024**.
7. In the **JPEG Quality** box, enter **100**.

8. On the **Features** tab, in the **Properties** section, ensure the **Allow asset upload from file system** check box is not selected, as shown:



9. Click **Done** to save changes.
10. Navigate back to the **We.Train** > **English** page.
11. From the **Components** tab on the left, drag the **Image** components onto the **Drag components here** area of the English page.
12. With the **Image** component selected, click the Configure icon (wrench icon). The **Image** dialog box opens. Notice that you can only drag an asset from the DAM onto the image component but cannot upload images from your local file system.
13. Click **Cancel**.

Developing Core Components

Core components provide robust and extensible base components, which are used by developers to implement the business requirements.

As a developer, before you start working with Core components:

- View the AEM GEMS session to understand the features of Core components and how to leverage them in AEM
- Check the component library to view the current release of Core components and different examples on how to use the Core components in your project
- Follow Adobe's recommendations on when and how to use Core components in your new or existing projects
- Start developing AEM Sites with Core components through the WKND tutorial that demonstrates how to implement Core components in AEM

Author with Core Components

Core components provide the rich-authoring functionality and several advantages to authors when adding content to pages. Components are available on the **Components** tab of the side panel of the page editor when editing a page.

Components are grouped according to categories called component groups to easily organize and filter the components. The component group name is displayed along with the component in the component browser, and it is also possible to filter components by group to easily find the right component.

Preconfiguring Core Components

With Core components, a template author can configure a number of capabilities through the Template Editor or in the design mode. For example, if an image component should not allow image upload from the file system or if a text component should only allow certain paragraph formatting features, you can with a simple click, enable or disable those features.

As the Core components can be preconfigured by template authors to define what options are allowed as part of a template, and further configured by the page author to define actual page content, each component can have options in two different dialogs:

- Edit dialog: Provides the options that a page author can modify during page editing for the placed components. For example, formatting of content text and rotating an image on a page.
- Design dialog: Provides the options that a template author can modify when configuring a page template. For example, text formatting and image in-place editing options.

The styles of most Core components can be defined by using the AEM style system.

- A template author can define which styles are available for a particular component in the design dialog of that component.
- The content author can then choose which styles to apply when adding the component and creating content.

Note: As a developer, understanding the Core components authoring capabilities and leveraging the Core components is vital for rapid project development. After you know the authoring features of the Core components, you can create a development plan for designing and customizing the components further, if required. To test the authoring capabilities of the Core components, you must create proxy components and add the Core component client libraries to your project.

Exercise 5: Add styles to Core components

The AEM style system is built to provide design flexibility to authors. For example, the core carousel component you installed as a proxy component has padding around the child components. An author may want the ability to optionally have a teaser extend across the entire component for a carousel. This could be accomplished with a component modification (AEM development skills needed) or with the style system and css (basic web development skills needed). Modifying the component requires knowledge on extending the core component safely whereas the style system only requires you to write css in your client library—a much easier task. In this exercise, you will add the style system to carousel and teaser components, giving the authors the ability to change the design if they want to.

Recall that the We.Train site relies on the WKND design. The WKND site is built as a maven project. The project is compiled and then installed into AEM as content package, which is the best practice in AEM. You can see the entire WKND project on github. The WKND project uses a UI.Frontend Module and a webpack development server for dedicated front-end development. As this module is compiled and minified outside AEM, it is included under exercise-files/training-files/clientlibs/ui.frontend.zip for ease of understanding.

1. In the Exercise Files folder provided to you, navigate to training-files/clientlibs folder and unzip the **ui.frontend.zip** file.

Note: When the ui.frontend module is compiled with Maven, all these files are minified and merged into a single file called site-bundle.css. You can find this file in **CRXDE Lite** under **/apps/wknd/clientlibs/clientlib-site/css**

2. Within the ui.frontend folder you unzipped, navigate to src/main/webpack/components/content/carousel/scss/styles

3. Double-click the **_hero.scss** file and notice the new class name, **cmp-carousel--hero**. This class will remove the padding from the child components in the carousel component for a hero look.

```
.cmp-carousel--hero {
    padding-left: 0 !important;
    padding-right: 0 !important;

    .cmp-carousel {
        margin-bottom:4em;
    }

      @media (max-width: $screen-small) {
        .cmp-carousel__indicators {
            float: left;
            padding-left: $gutter-padding;
            padding-top: $gutter-padding;
        }
      }

}
```

Note: This file is compiled into the **site-bundle.css** file because of the @import statements. **_hero.scss** is imported to **carousel.scss**. The **carousel.scss** is imported in **~webpack/site/main.scss** and **main.scss** is then minified to create the **site-bundle.css** within the **clientlibs-site** client library. This method is followed for all components of WKND and a similar process is followed for We.Train specific styles.

4. Navigate to **src/main/webpack/components/content/teaser/scss/styles**.

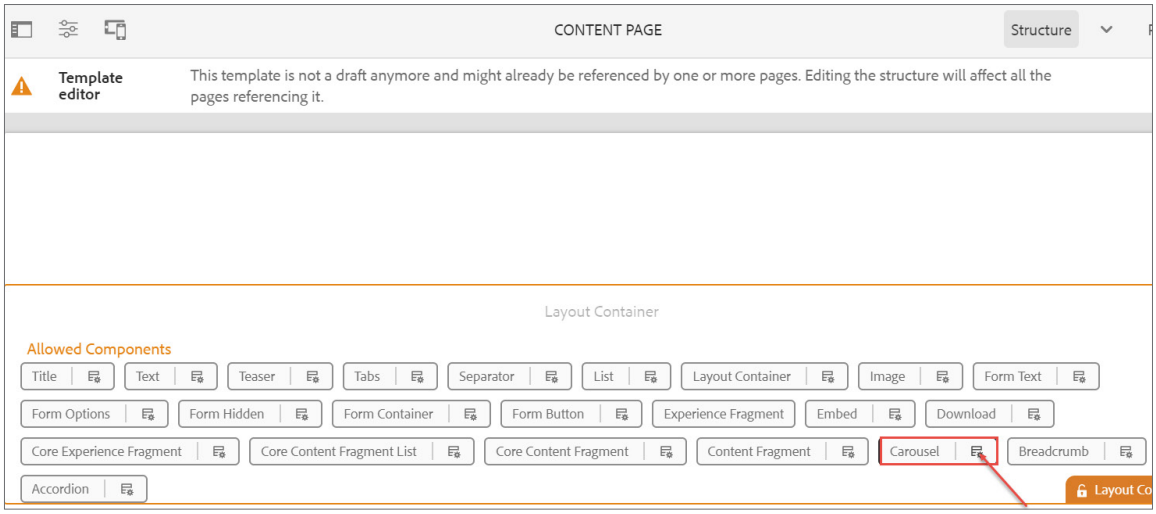
5. Double-click the **_list.scss** file and notice the new class name, **cmp-teaser--list**. This class will resize the image to 200px and change the title and description styles.

```

.cmp-teaser--list {


    .cmp-teaser__image {

        .cmp-image__image {
            object-fit: cover;
            max-height: 200px;
        }
    }

    .cmp-teaser__title {
        font-size: $font-size-medium;
        font-family: $font-family-sans-serif;
        font-weight: $font-weight-bold;
        text-transform: uppercase;
    }

    .cmp-teaser__description {
        font-size: $font-size-small;
        color: $gray;
        text-transform: uppercase;
        p {
            font-size: $font-size-small;
        }
    }
}
```

Note: WKND is already installed in AEM, which means these styles are minified already in /apps/wknd/clientlibs/clientlib-site/css and are ready to be optionally used in the template as styles.

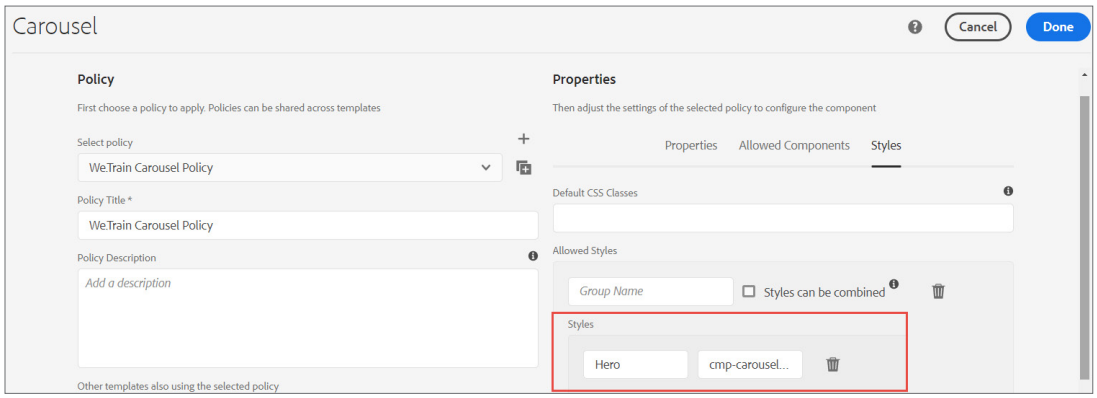
To optionally apply these class names to the component, you need to give the author the ability to select them. To do this, modify the content policies in the template.

6. In your AEM author service, navigate to **Tools** > **General** > **Templates** > **We.Train**.
7. Select the **Content Page** template and click **Edit** (pencil icon). The template opens on a new browser tab.
8. Within the inner Layout Container, click the **Policy** icon on the **Carousel** component, as shown. The **Carousel** wizard opens.



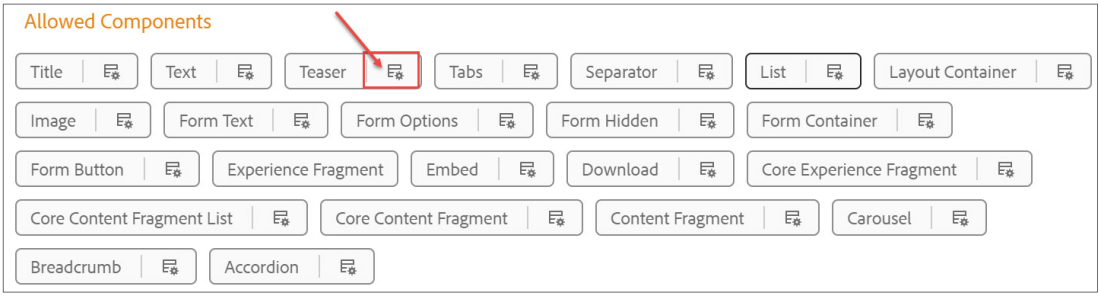
9. On the **Policy** section, enter **We.Train Carousel Policy** in the **Policy Title** box.
10. On the **Properties** section, ensure you are on the **Styles** tab and click **Add** under **Allowed styles**. The **Styles** section appears.
11. Click **Add** under **Styles**. The **Style Name** and **CSS Classes** boxes appear.
12. In the **Style Name** box, type **Hero**.
13. In the **CSS Classes** box, type **cmp-carousel--hero**.

14. Click **Add**. The style is added.



15. Click **Done**.

16. In the **Layout Container**, click the **Policy** icon on the **Teaser** component. The **Teaser** wizard opens.



17. On the **Policy** section, enter **We.Train Teaser Policy** in the **Policy Title** box.

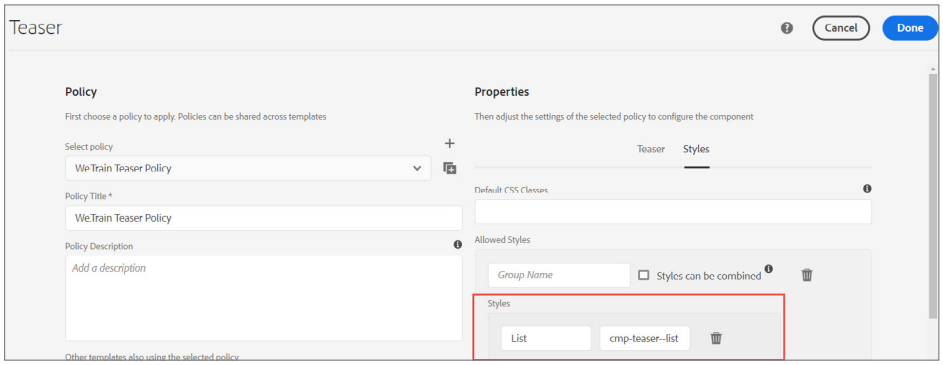
18. On the **Properties** section, ensure you are in the **Styles** tab and click **Add** under **Allowed styles**. Another **Add** button appears under **Styles**.

19. Click **Add** under **Styles**. The **Style Name** and **CSS Classes** boxes are displayed.

20. In the **Style Name** box, type **List**.

21. In the **CSS Classes** box, type **cmp-teaser--list**.

22. Click **Add**. The style is added.



23. Click **Done**. You will test these new styles in the next exercise.

Exercise 6: Author Core components

Core components provide the toolset to quickly start your projects with minimal component development. In this exercise, you will explore the different Core components you added to your project. While working with these Core components, you will notice that they do not have a design yet. By using client libraries, you can easily apply designs to the core components—as they follow the BEM notation. This is a good exercise to come back to after completing this class to fully understand the capabilities of some of the core components.

In this exercise, you will perform the following tasks:

1. Create a Page
2. Add Core components

Task 1: Create a page

1. In your AEM author service, navigate to the **Sites** > **We.Train** > **English** page.
2. Click **Create** > **Page**. The **Create Page Template** wizard opens.
3. Select the **Content Page** template and click **Next**. The **Create Page** wizard opens in the **Properties** tab.
4. In the **Title** box, type **Core Components** and click **Create**. A **Success** dialog box opens.
5. Click **Open** to open the page.

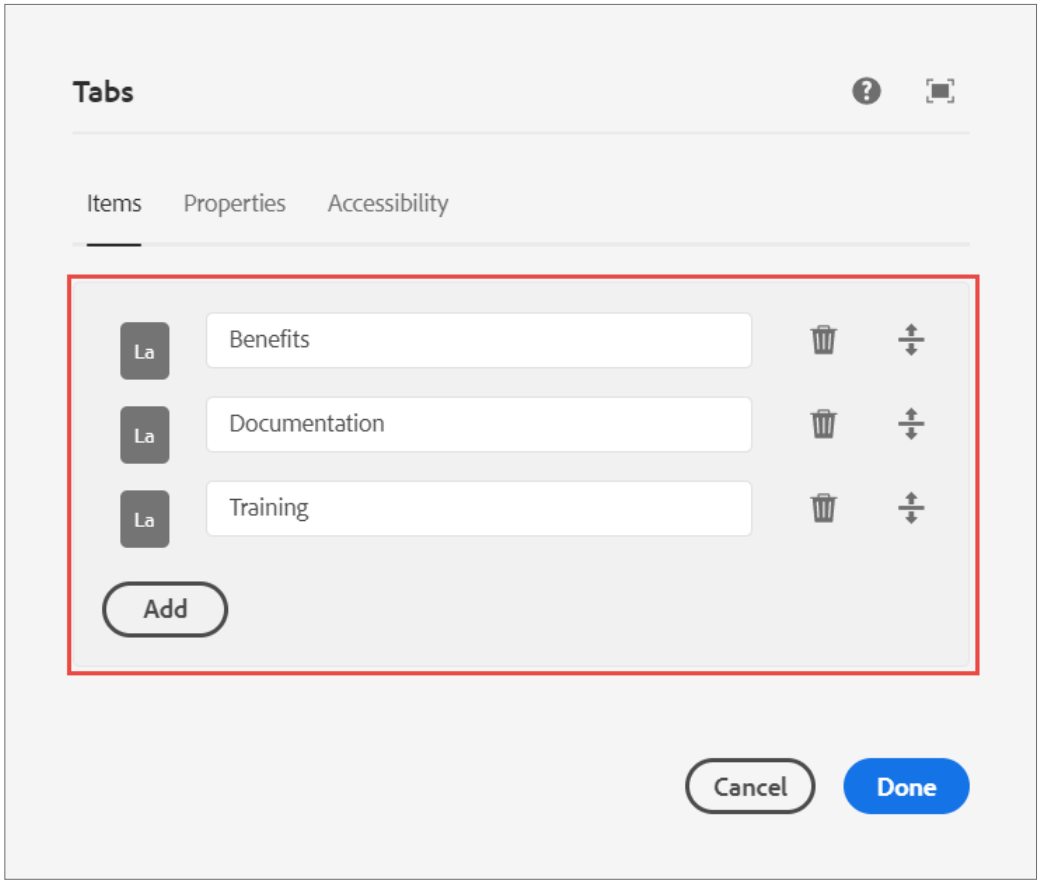
Task 2: Add core components

1. On the page you created in the previous task, ensure you are in the **Edit** mode and click the **Toggle Side Panel**.
2. Click the **Components** icon on the left panel. The list of available components is displayed.
3. Drag the **Title** component onto the **Drag components here** area. The component is added.
4. Double-click the **Title** component you added to edit it. The **Title dialog** box opens.

5. In the **Title** box, type **Learn about Core Components** and click the **Done** at the lower right. The title is updated to **Learn about Core Components**.
6. From the **Components** tab in the left panel, drag the **Tabs** component onto the **Drag components here** area. Two tabs are added—**TAB 1** and **TAB 2**, as shown:

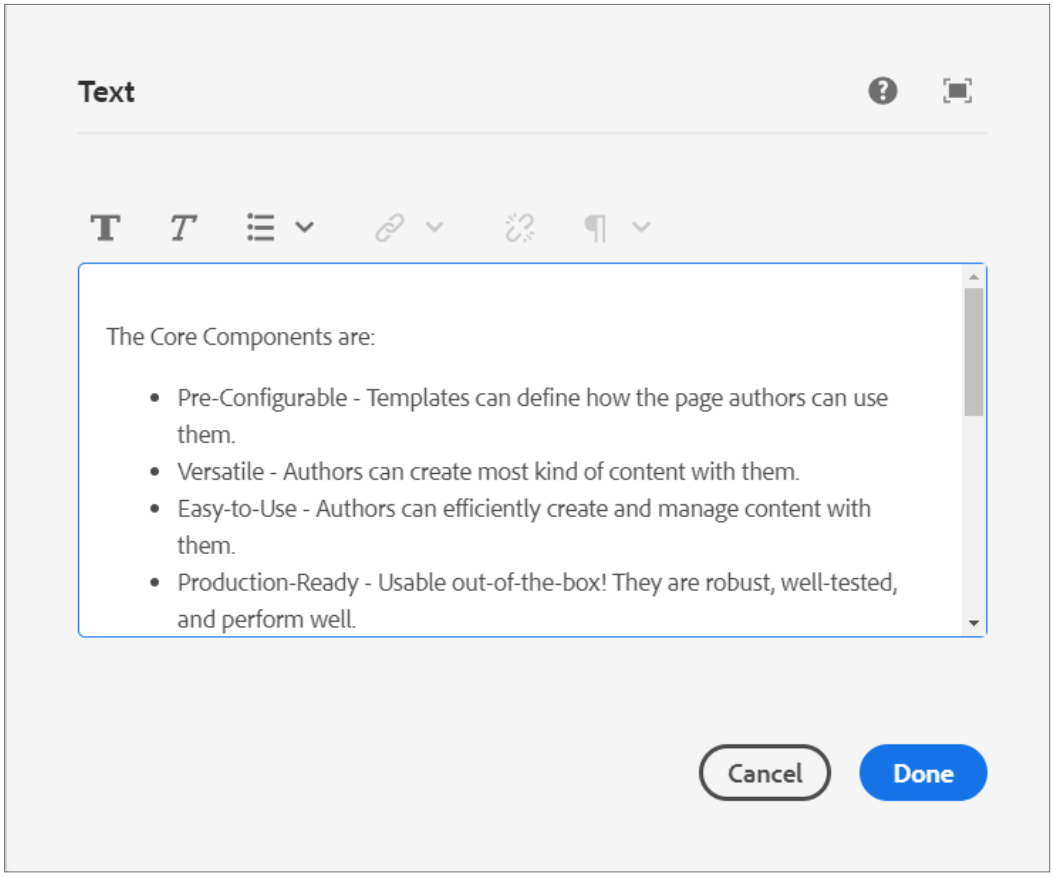
7. Double-click the tabs component. The **Tabs** dialog box opens.
8. On the default **Items** tab, rename **Tab 1** as **Benefits** and **Tab 2** as **Documentation**.
9. Click **Add**. The **Insert New Component** dialog box opens.
10. Select **Layout Container**. A new tab is added.

11. Name the new tab as **Training**. The **Tabs** dialog box should look as shown:



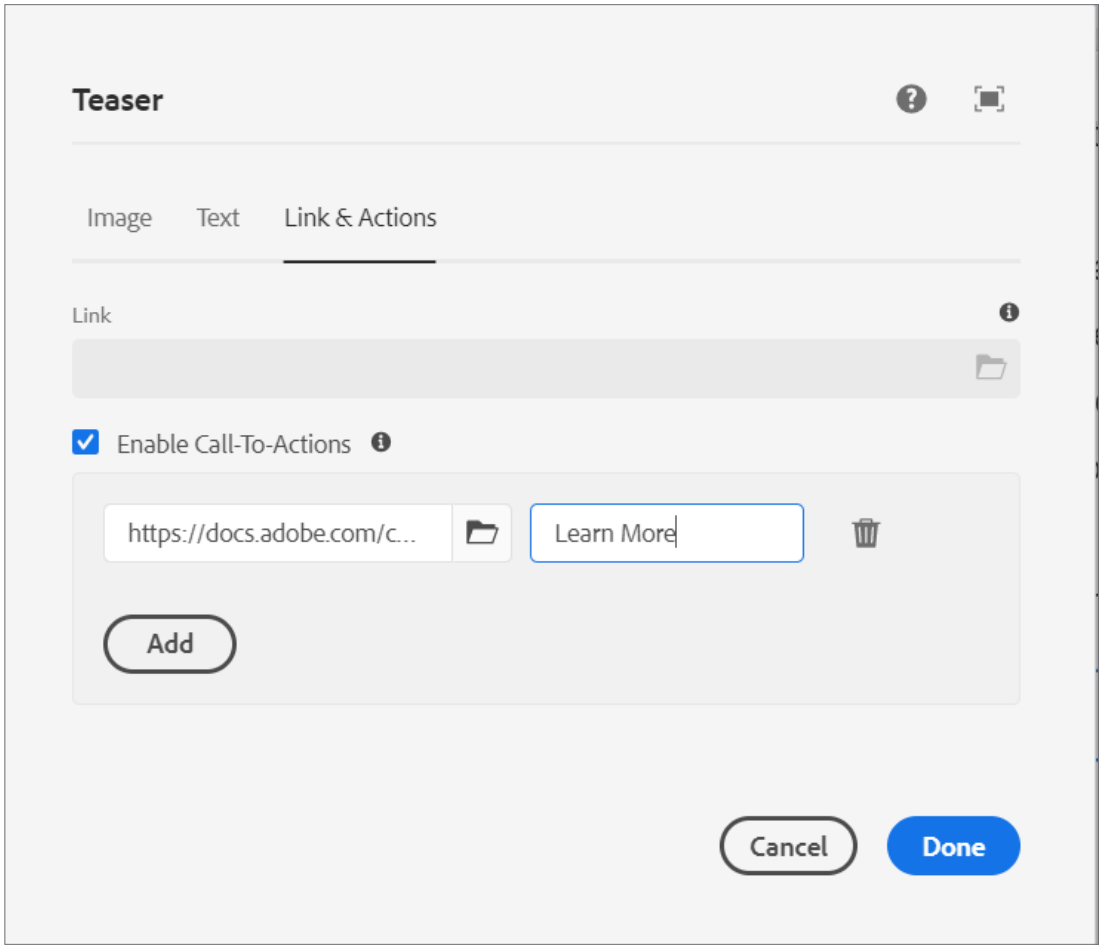
12. Click **Done** to save the changes. The three tabs are added to the page.
13. On the **BENEFITS** tab, click the **Drag components here** area and click the **Insert component** icon (+ icon) that appears. The **Insert New Component** dialog box opens.
14. Add a **Text** component from the I**nsert New Component** dialog box. The component is added.
15. In the Exercise Files folder provided to you, navigate to training-files/core-components folder and copy the content from **Benefits.txt**.
16. On the **CORE COMPONENTS** page, double-click the **Text** component. The **Text** dialog box opens.

17. Paste the content and apply the bullet list format to the text, as shown:

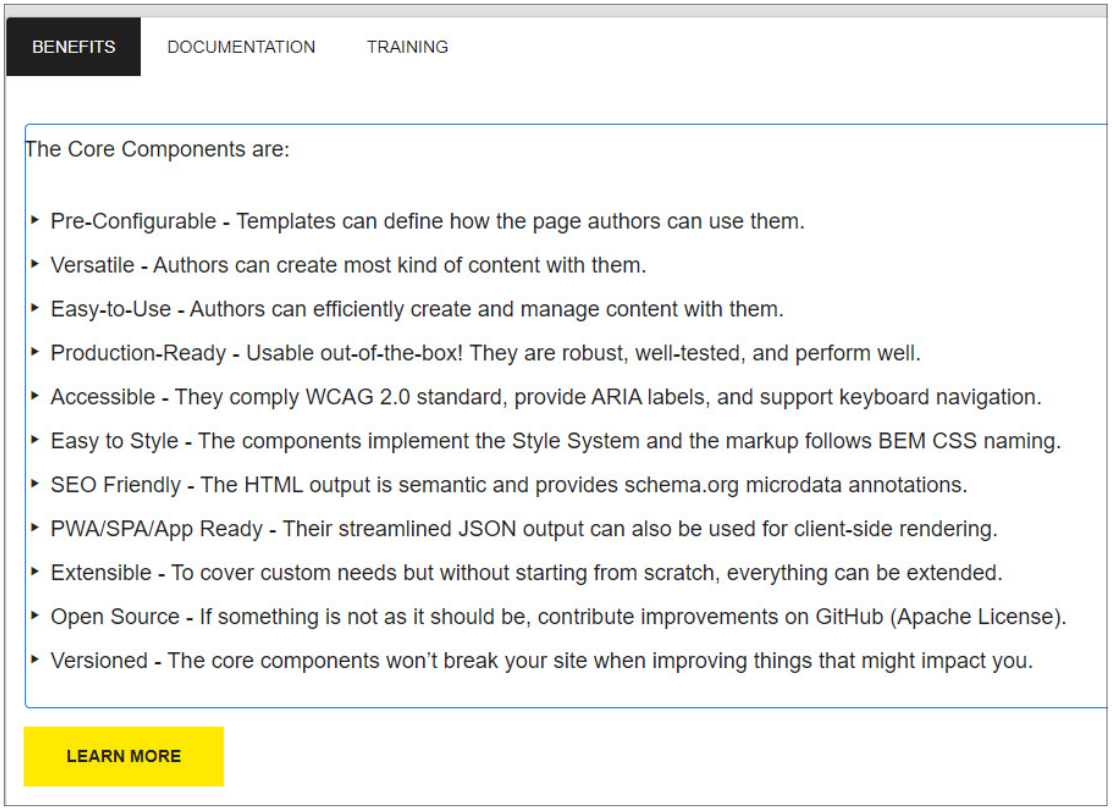


18. Click **Done** to save the changes. The text is added to the **Benefits** tab.
19. Click the **Drag components here** area below the text you added in the previous step and click the **Insert component** icon (the + icon) that appears.
20. Add a **Teaser** component. The component is added.
21. Click the **Teaser** component and click the **Configure** icon (the wrench icon). The **Teaser** dialog box opens.
22. On the **Link & Actions** tab, select the **Enable Call-to-Actions** check box. Two boxes, **Link** and **Text**, appear.
23. In the **Link** box, type **https://docs.adobe.com/content/help/en/experience-manager-core-components/using/introduction.html**

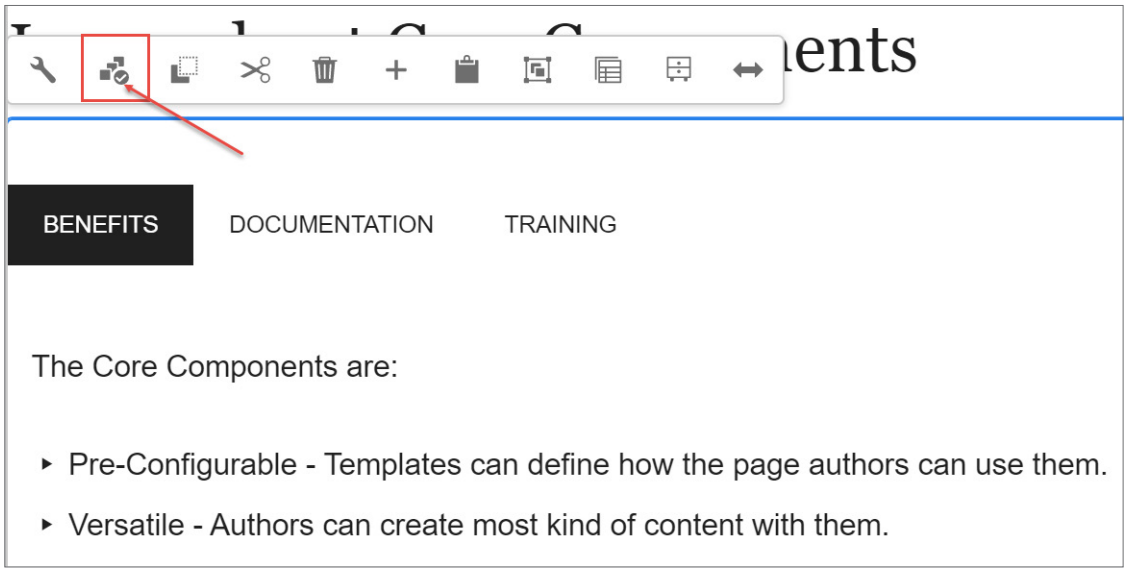
24. In the **Text** box, type **Learn More**. The **Teaser** dialog box should look as shown:



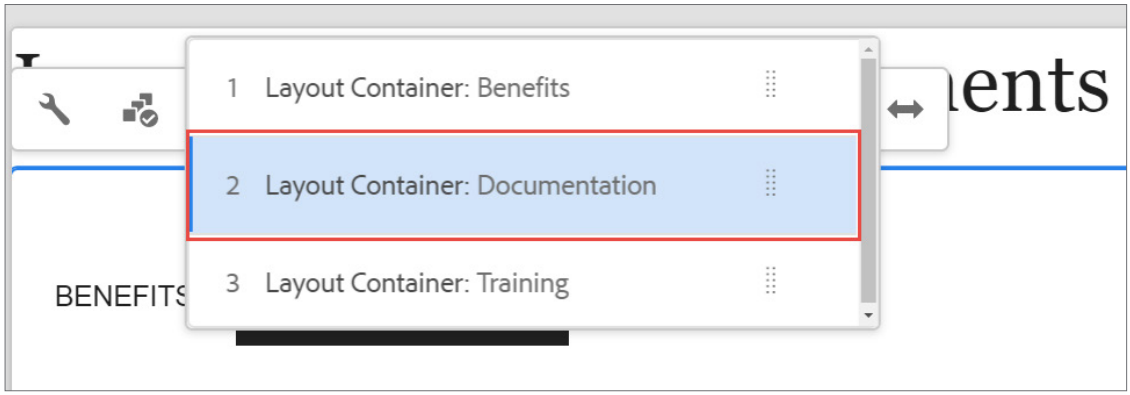
25. Click **Done** to save the changes. The **LEARN MORE** button is added to the page. The **BENEFITS** tab should look as shown:



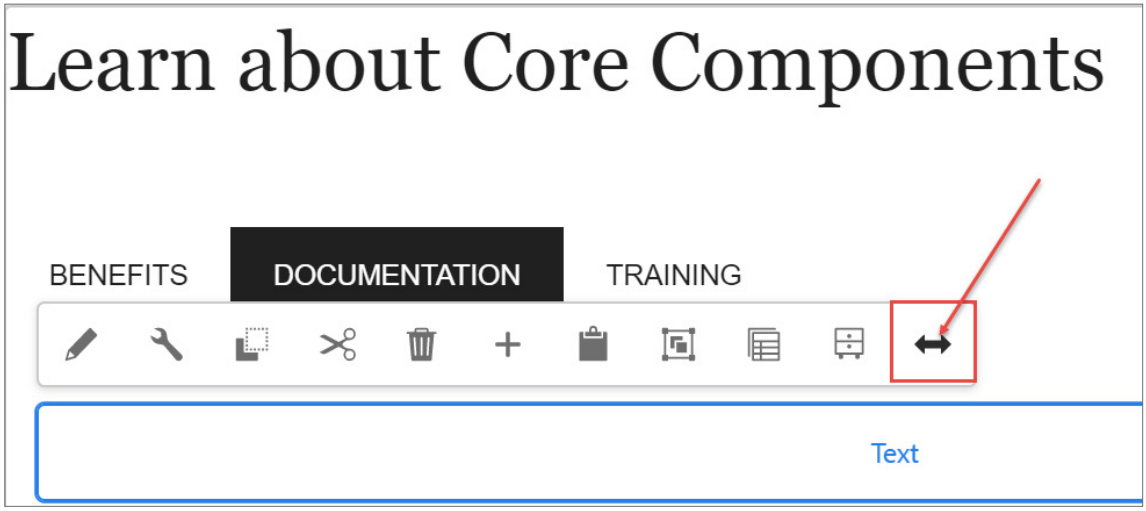
26. To add components and content on the **DOCUMENTATION** tab, click the **Tabs** component and click **Select panel**, as shown:



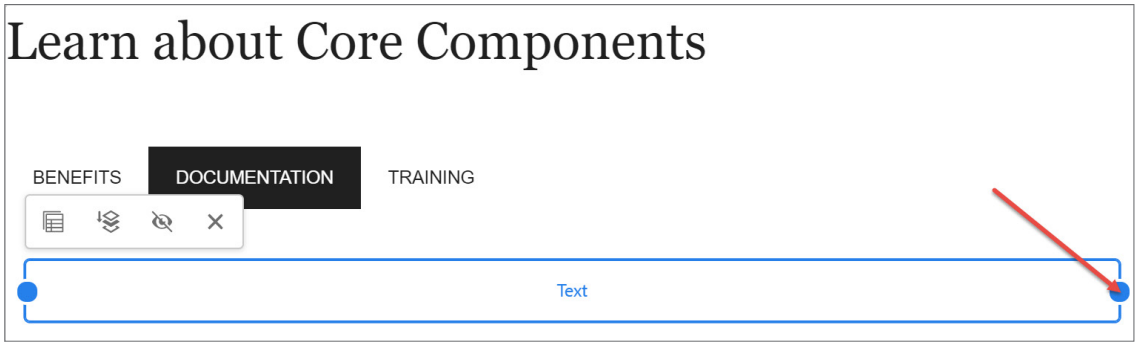
27. Click **Layout Container: Documentation**, as shown. The **DOCUMENTATION** tab is highlighted in black. The components and content you insert now will be added under the **DOCUMENTATION** tab.



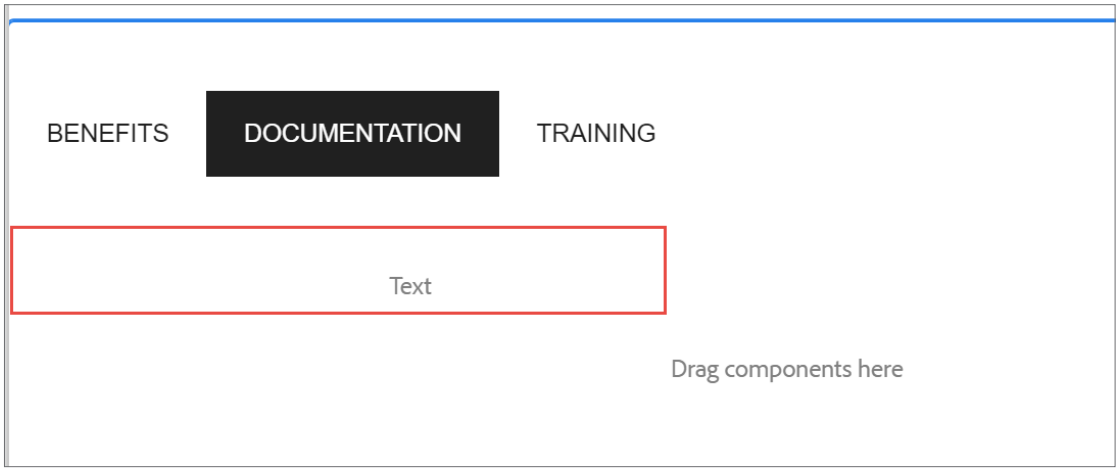
28. On the **DOCUMENTATION** tab, click the **Drag components here** area.
29. Click the **Insert component** icon (the + icon) and add a **Text** component.
30. Click the **Text** component and click **Layout**, as shown. The layout is now editable.



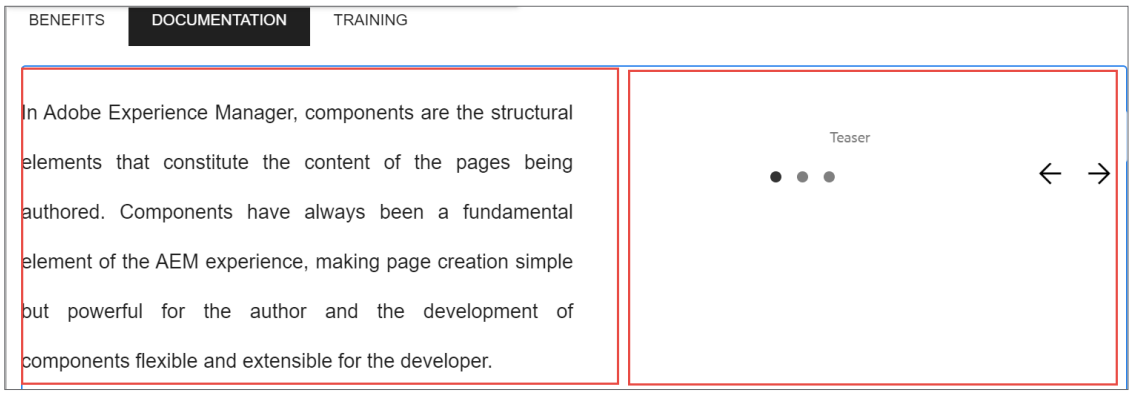
31. Drag the blue circle at the right side, as shown, and adjust the layout to 6/12 columns.



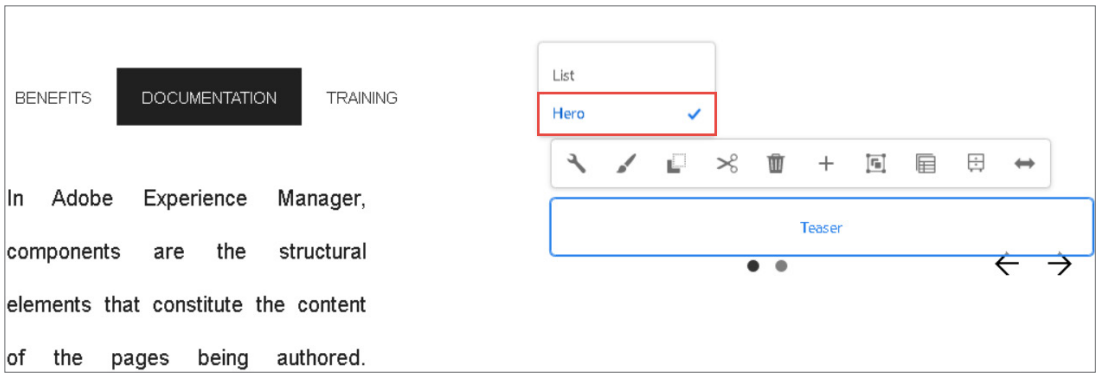
32. Click **Close** (X). The component should look, as shown:



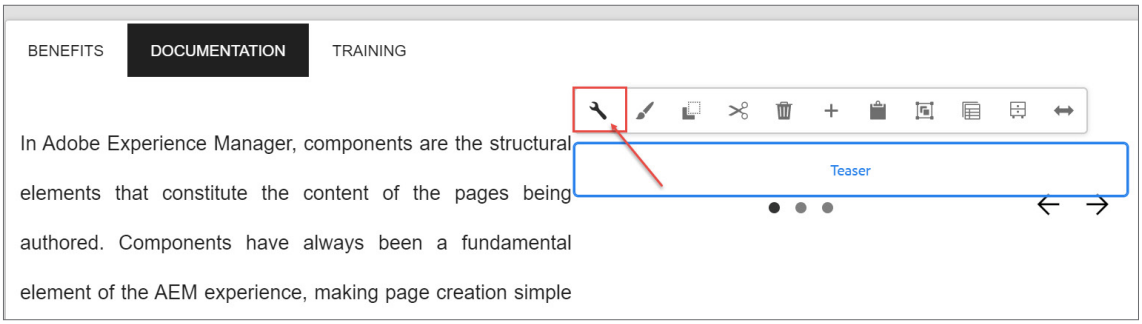
33. In the Exercise Files folder provided to you, navigate to training-files/core-components folder and copy the content from **Documentation.txt**.
34. Click the **Text** component in AEM and click the **Edit icon** (pencil icon) to add content.
35. Paste the content you copied in step 33 and click **Save** (check mark). The text is added under the **DOCUMENTATION** tab.
36. Add a **Carousel** component under the Text component in the **DOCUMENTATION** tab.
37. Click the **Carousel** component and click the **Layout** icon. The layout is now editable.
38. Drag the right blue circle and adjust the layout to 6/12 columns. The two components are now placed adjacent to each other, as shown:



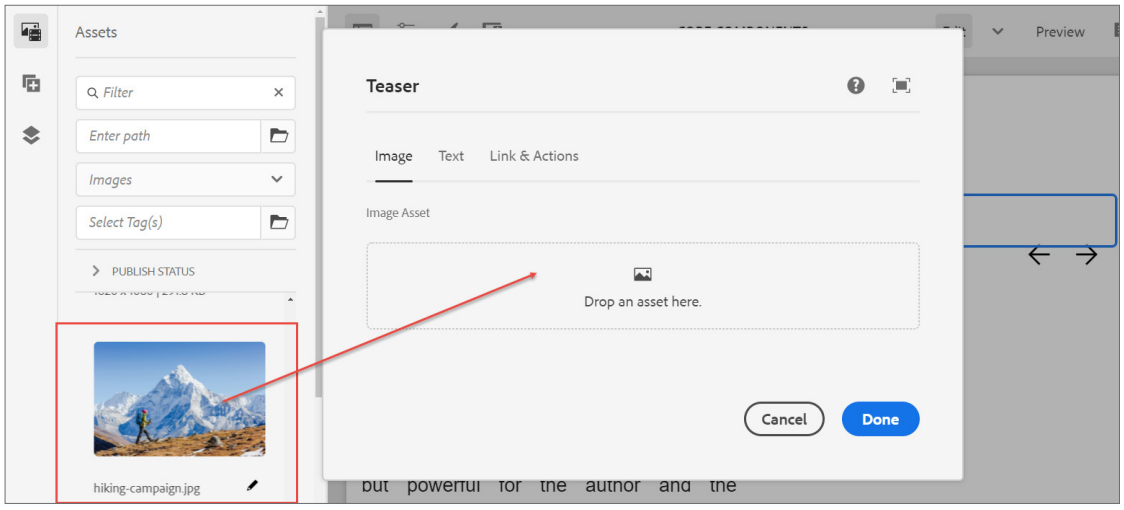
39. Select the **Teaser** component, click the **Styles** icon (paint brush icon) and click **Hero**, as shown. You should notice a design change.



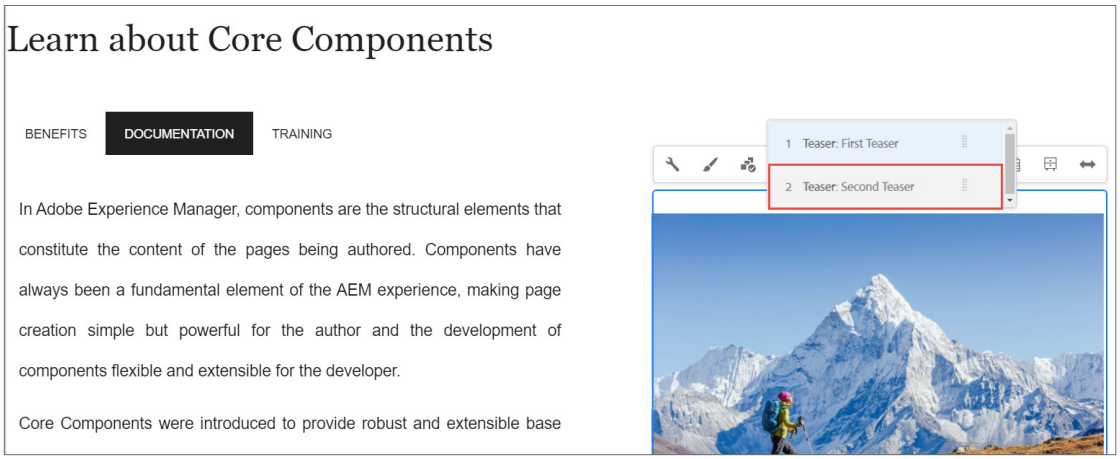
40. Select the **Teaser** dialog box and click **Configure**, as shown. The **Teaser** dialog box opens.



41. On the **Image** Tab, drag an image from the **Assets** tab onto the **Drop an asset here** area. The image is added.



42. On the **Text** Tab, select the **Get title from linked page** check box.
43. On the **Links & Actions** tab, select the **Enable Call-to-Actions** check box. Two boxes, **Link** and **Text**, appear
44. In the **Link** box, enter **https://docs.adobe.com/content/help/en/experience-manager-core-components/using/introduction.html**
45. In the **Text** box, enter **Author Core Components**.
46. Click **Done**.
47. Click the **Teaser** component, click the **Style** icon (paintbrush icon) and click **List**. You should notice a design change.
48. Select the **Carousel** component, click the **Select Panel** icon and click **Teaser: Second Teaser**, as shown, to add content under the second teaser.

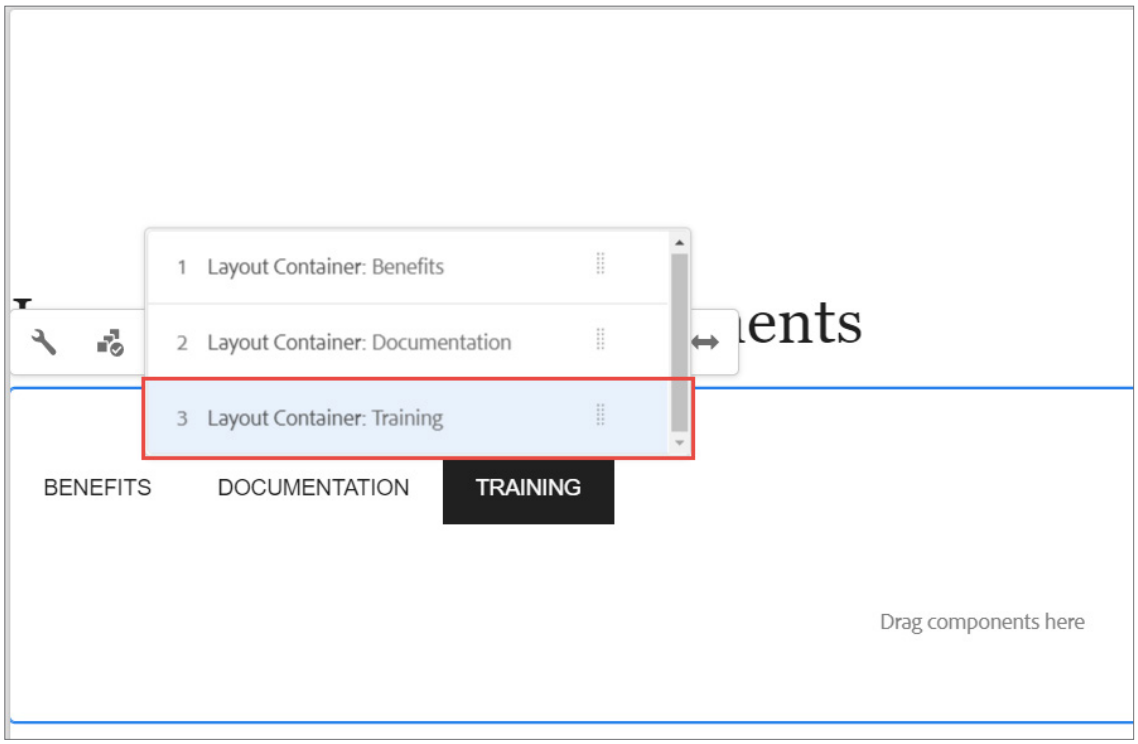


49. Click the **Toggle Side Panel** icon to access the options in the left panel.
50. Select **Teaser** component and click the **Configure** icon. The **Teaser** dialog box opens.
51. From the **Assets** tab, drag an image onto the **Drop an asset here** area.
52. On the **Text** tab, select the **Get title from linked page** check box. Two boxes, **Link** and **Text**, appear.
53. On the **Links & Actions** tab, select the **Enable Call-to-Actions** check box. Two boxes, **Link** and **Text**, appear.
54. In the **Link** box, enter **https://docs.adobe.com/content/help/en/experience-manager-core-components/using/introduction.html**
55. In the **Text** box, type **Develop Core Components**.

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

57. Click the **Teaser** component, click the **Style** icon (paintbrush icon) and click **List**. You should notice a design change.

58. Select the **Tabs** component, click the **Select Panel** icon and click **Layout Container: Training** to add components and content under the **TRAINING** tab.



59. Click the **Drag components here** area and add the **Title** component.

60. Double-click the **Title** component you added now to edit it. The **Title** dialog box opens.

61. In the **Title** box, type **Sign up Today!**

62. From the **Type/Size** drop-down menu, select **H3**.

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

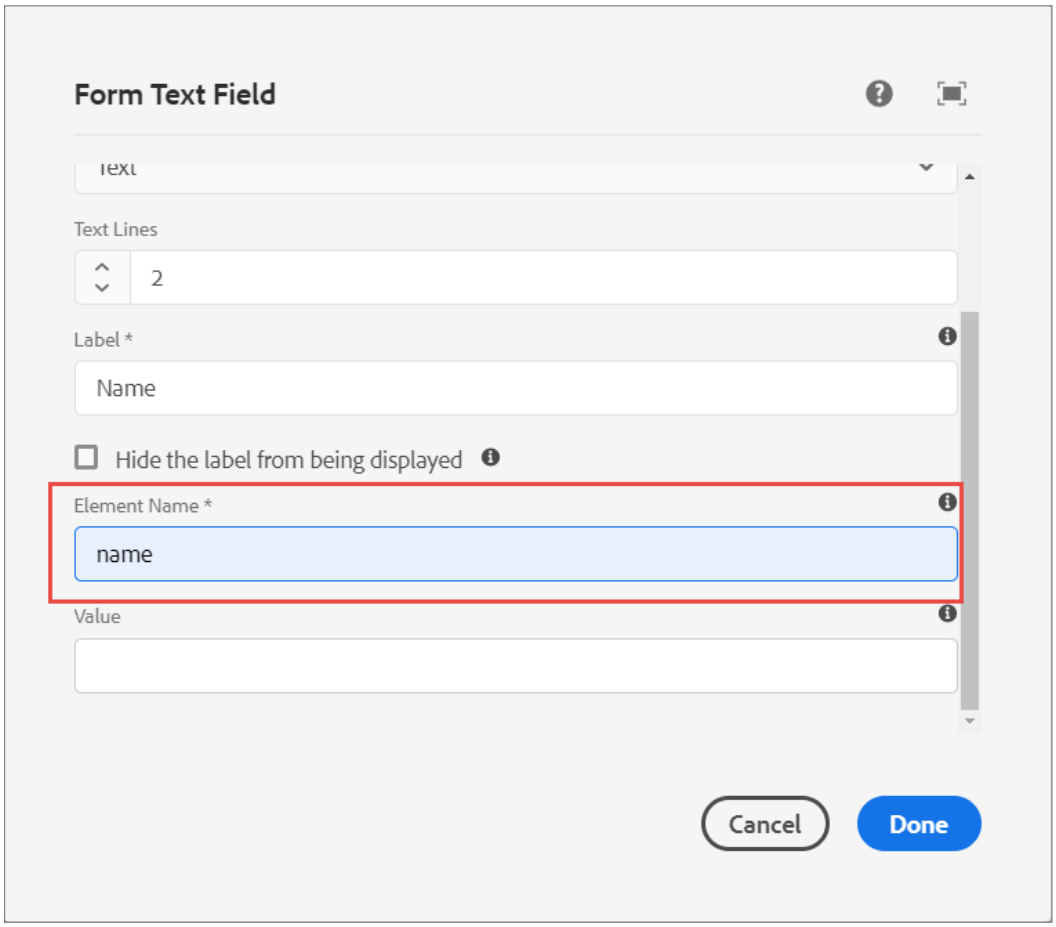
64. Add a **Form Container** component, select it and click **Configure**. The **Form Container** dialog box opens. Notice under **Content Path** that the submitted form is saving to /content/usergenerated/*. This could be an external database as well.

65. Click **Cancel**.

66. Add a **Form Text** component, select it and click **Configure**. The **Form Text Field** dialog box opens.

67. In the **Label** box, type **Name**.

68. In the **Element Name** box, type **name**, as shown:



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

70. Add another **Form Text** component to the page, select it and click **Configure**. The **Form Text Field** dialog box opens.

71. In the **Label** box, enter **Email**.

72. In the **Element Name** box, enter **email**.

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

74. Add a **Form Options** component to the page, select it and click **Configure**. The **Form Options Field** dialog box opens.

75. In the **Title** box, enter **Audience**.

76. In the **Name** box, enter **audience**.

77. Click **Add** under **Option**. The **Text** box appears.

78. Enter **Author** in the **Text** box.

79. Click **Add.** Another **Text** box appears.
80. Type **Developer** in the **Text** box.
81. Click **Done** to save the changes.
82. Add a **Separator** component to the page. A line that separates the two components are added.
83. Add a **Form Button** component below the Separator component, select it, and click **Configure**. The **Form Button** dialog box opens.
84. In the **Title** box, enter **Sign Up!**
85. Click **Done** to save the changes.

To view your completed work, click **Preview** at the upper-right corner of the page. Click all tabs and links to check if all the elements are working correctly.

Note: You can see the completed version of this exercise by uploading and installing the **core-component-authoring-in-xf.zip** content package from the exercise files. This content package has an experience fragment called **Core Components**. You can view this fragment by adding an Experience Fragment component on the page and dragging the Core Components fragment onto the page.

References

- Core components introduction:
  https://docs.adobe.com/content/help/en/experience-manager-core-components/using/introduction.html
- Component library:
  http://opensource.adobe.com/aem-core-wcm-components/library.html
- Core components versions:
  https://docs.adobe.com/content/help/en/experience-manager-core-components/using/versions.html
- Author with Core components:
  https://docs.adobe.com/content/help/en/experience-manager-core-components/using/get-started/authoring.html
- Using Core components:
  https://docs.adobe.com/content/help/en/experience-manager-core-components/using/get-started/using.html
- BEM:
  http://getbem.com/introduction/

Working with Components

Introduction

In addition to the wide range of default components available in Adobe Experience Manager (AEM), you can develop new components that offer flexible, feature-rich authoring functionality. To develop components in AEM, you must understand the structure of the components and its configuration, component hierarchy and inheritance, component edit behavior, HTML Template Language (HTL) business logics, and design configuration through content policies.

Objectives

After completing this module, you will be able to:

- Describe the structure of a component
- Describe structure components and content components
- Explain the HTL business logic
- Create a header component
- Add JavaScript business logic to a header component
- Add Sling Model business logic to a header component
- Create an edit dialog for the component
- Add an editconfig node to the component
- Explain AEM design dialogs
- Create a design dialog for a component

Structure of a Component

The component structure includes:

- 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: It is the component title. For example, this property is used as a label when the component is listed in the components browser.
- jcr:description: It is the description for the component. You can use this property as a mouse-over hint in the components browser.
- cq:icon: It 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 edit the design of a component

Structure Components and Content Components

The content of a page is distributed into layouts 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.

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 the structure components of the page.
- Authors can add and remove content components from the page.

HTL Business Logic

HTL is HTML5 that helps develop and design business logic in HTL completely. HTL helps you add complex business logic to the code through the following options:

- Sling Models
- Server-Side JavaScript

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 Models support 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 is convenient to use 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"
    };
});
```

Exercise 1: Add a header component

In this exercise, you will create a component from the start and use mock HTL to add the component to a template for testing and designing.

In this exercise, you will perform the following tasks:

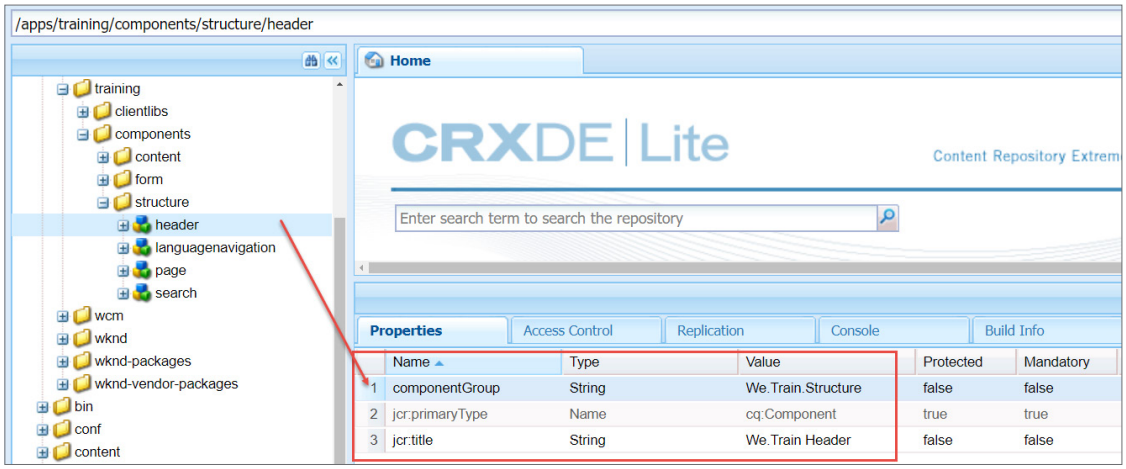
1. Create a header component
2. Add the header to the template
3. Add styling to the header component

Task 1: Create a header component

1. Open the **CRXDE Lite** page (http://localhost:4502/crx/de) if it is not open.
2. Navigate to the **/apps/training/components/structure** folder.
3. Right-click the **structure** folder and click **Create** > **Create Node**. The **Create Node** dialog box opens.
4. Enter the following details in the dialog box:
   a. **Name**: **header**
   b. **Type: cq:Component**
5. Click **OK**. The **header** node is created.
6. Click **Save All.**
7. Ensure the **header** node is selected and add the following properties:

Name	Type	Value
jcr:title	String	We.Train Header
componentGroup	String	We.Train.structure

8. Click **Save All** from the actions bar. The node should look as shown:



9. Right-click the **header** node and click **Create** > **Create File**. The **Create File** dialog box opens.
10. In the **Name** box, type **header.html**.
11. Click **OK**. The **header.html** file is created. Notice that the **header.html** editor opens at the right.
12. Click **Save All** from the actions bar.
13. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/structure/header** folder.
14. Open the **basic_header.html** file in Notepad++ (or any editor of your choice), copy the content of the file and paste it in the **header.html** editor in **CRXDE Lite**.
15. Click **Save All** to save changes.
16. Review the code you added. Notice how the code snippet of **header.html** has a simple navigation that lists the child pages of the current page. If there are no child pages, Page One, Page Two, Page Three, and Page Four are displayed. The **header.html** also has the Language Navigation and Search proxy components coded in. You can configure these.

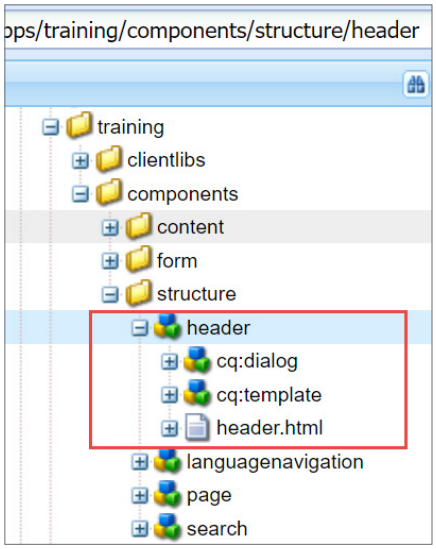
```
<div class="cmp-experiencefragment--header">
    we<strong>TRAIN</strong>

    <!--/* Adds the language navigation core component */-->
    <div class=""
        data-sly-resource="${'languagenavigation' @ resourceType='training/components/structure/languagenavigation'}">
    </div>

    <!--/* Adds the search core component */-->
    <div class="" data-sly-resource="${'search' @ resourceType='training/components/structure/search'}"></div>

    <!--/*********************************************************************
         * Using HTL we can quickly create a list of links for the header *
         *********************************************************************/-->
    <ul class="">
        <li class="" data-sly-repeat="${['Page One','Page Two', 'Page Three','Page Four']}">
            <a class="cmp-navigation__item-link" href="${'#' @ extension = 'html'}">${item}</a>
        </li>
    </ul>
</div>
```

Before you add the header component to the template, you must add a few node structures so that the template can function properly. To begin, create a dialog placeholder node for the header component, so that you can add the placeholder to a template or a page.

17. Right-click the **header** component and click **Create** > **Create Node**. The **Create Node** dialog box opens.
18. Provide the following details:
    a. **Name**: **cq:dialog**
    b. **Type**: **nt:unstructured**
19. Click **OK**. The **cq:dialog** node is created.
20. Click **Save All**.

Next, create a cq:template node structure. A few proxy components installed earlier had cq:template nodes too. cq:template nodes enables you to predefine sub components under a component when the component is added to a template or page. In this scenario, the header component will utilize the core languagenavigation and search components, and you need to create cq:template nodes for those components.

21. Right-click the **header** component and click **Create** > **Create Node**. The **Create Node** dialog box opens.
22. Provide the following details:
    a. **Name**: **cq:template**
    b. **Type**: **nt:unstructured**
23. Click **OK**. The **cq:template** node is created.
24. Click **Save All**. The header component should look as shown:



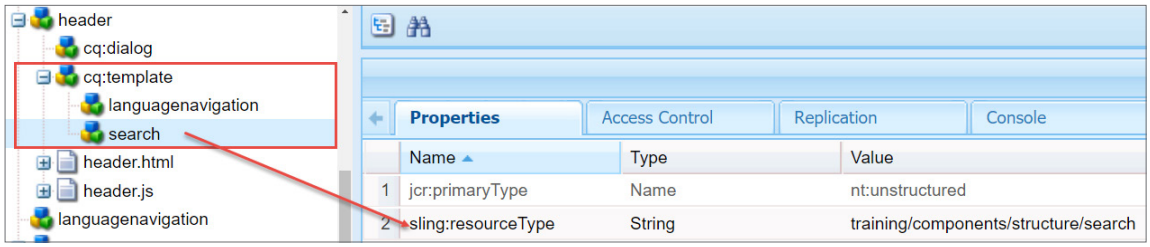
25. Right-click the **cq:template** node and click **Create** > **Create Node**. The **Create Node** dialog box opens.
26. Provide the following details:
    a. **Name**: **languagenavigation**
    b. **Type**: **nt:unstructured**
27. Click **OK**. The **languagenavigation** node is created.
28. Click **Save All**.
29. Ensure the **languagenavigation** node is selected and add the following property:

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

30. Click **Save All**.
31. Right-click the **cq:template** node and click **Create** > **Create Node**. The **Create Node** dialog box opens.
32. Provide the following details:
    a. **Name**: **search**
    b. **Type**: **nt:unstructured**
33. Click **OK**. The **search** node is created.
34. Click **Save All**.
35. Ensure the **search** node is selected and add the following property:

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

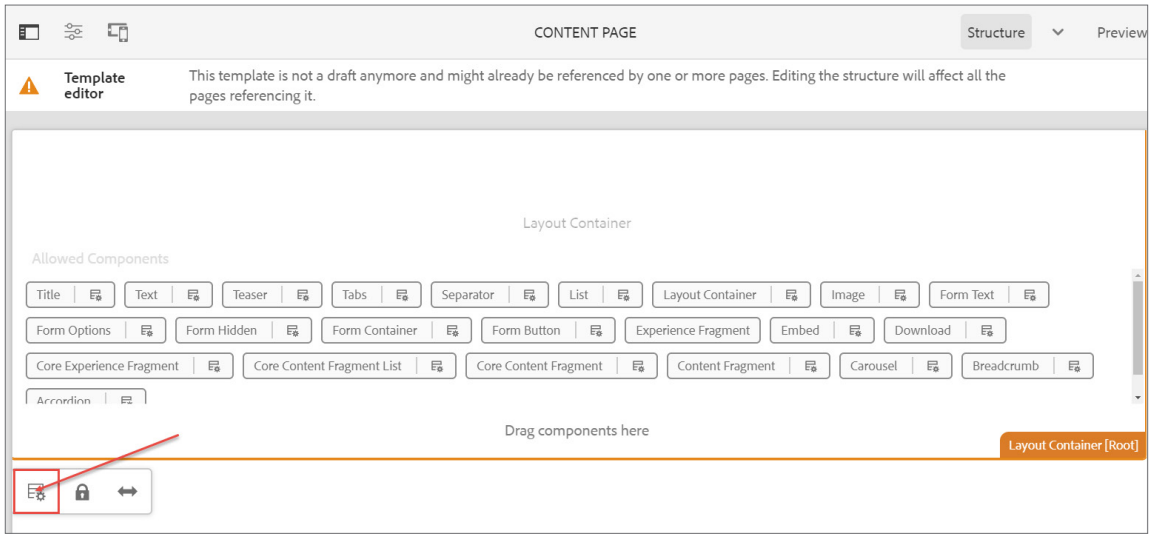
36. Click **Save All**. The **cq:template** node should look as shown:



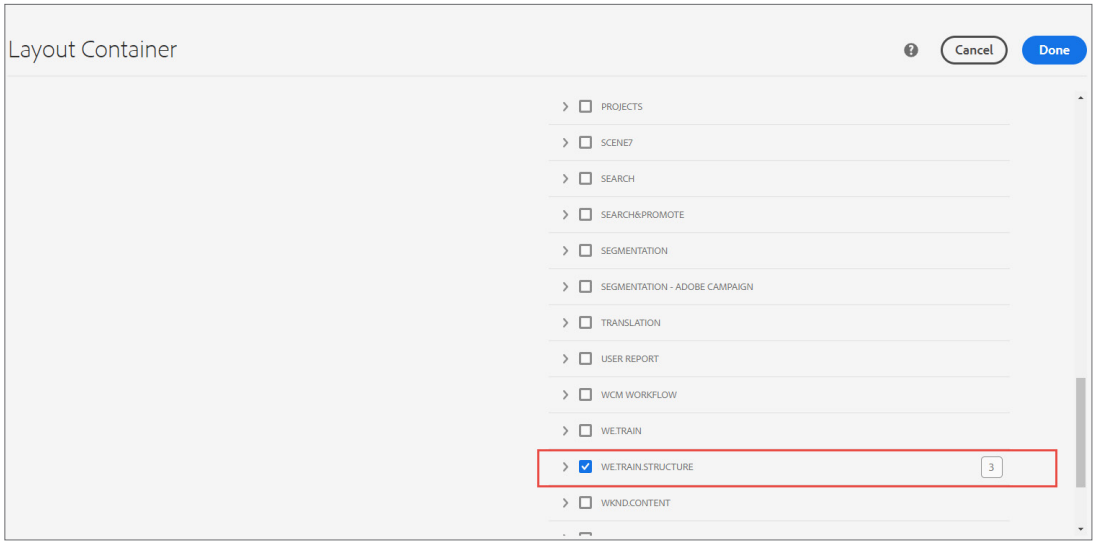
Task 2: Add the header to the template

To add the header component to the template:

1. In your AEM author service, click **Tools** > **Templates**. The **Templates** console opens.
2. Click the **We.Train** folder. The We.Train page opens.
3. Select the **Content Page** template by clicking the **Select** icon (check mark icon).
4. Click **Edit (*e*)** from the actions bar. The **Template editor** opens.
5. Select **Layout Container [Root]** and click the **Policy** icon from the component toolbar, as shown. The **Layout Container** policy wizard opens.



6. On the **Properties** section, on the **Allowed Components** tab, scroll down the components list, and select the **WE.TRAIN.STRUCTURE** check box, as shown. This group contains the header component.



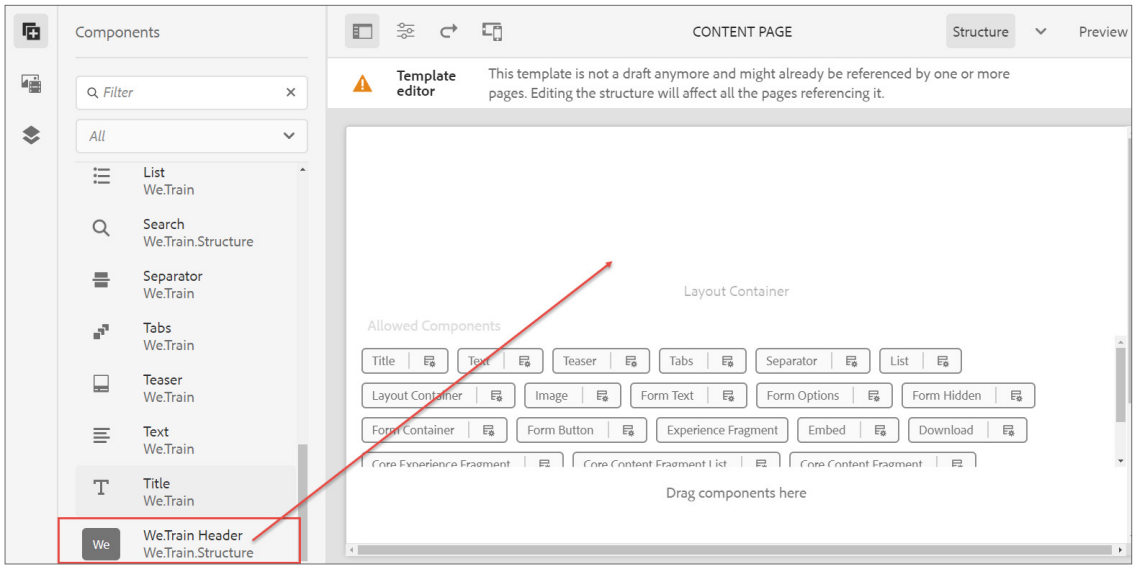
7. Click **Done** to save the changes.

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.

---

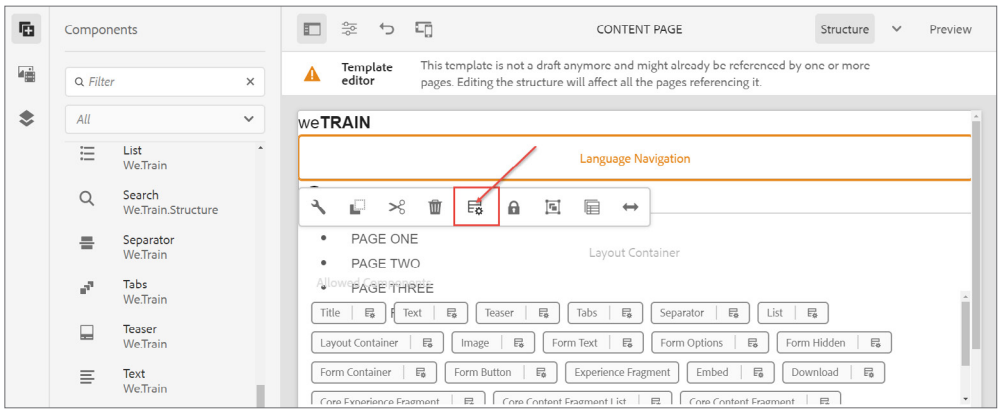
8. Refresh the **Content** page.
9. Click the **Toggle Side Panel** icon, if the left panel is not visible, and click the **Components** icon. The **Components** panel opens.

10. Drag the **We.Train Header** component from the panel and add it above the **Layout Container**, as shown. The **Header Component** is added to the template.



After adding the header component, you must set policies for the two proxy components.

11. Select the **Language Navigation** component and click the **Policy** icon from the component toolbar as shown. The **Language Navigation** wizard opens.



12. In the **Policy** section, in the **Policy Title** box, type **We.Train LangNav Content Policy**.
13. In the **Properties** section, type **/content/we-train** in the **Navigation Root** box.
14. Click **Done** to save the changes.
15. Select the **Search** component and click the **Policy** icon. The **Quick Search** page opens
16. In the **Policy** section, type **We.Train Search Content Policy** in the **Policy Title** box.
17. In the **Properties** section, type **/content/we-train** in the **Search Root** box.
18. Click **Done** to save the changes.

Note: You could have configured the language navigation and search components by using a cq:template node too—similar to the carousel and tab proxy components. For example, review /apps/training/components/content/tab in **CRXDE Lite**.

19. Click http://localhost:4502/editor.html/content/we-train/en.html. The **English** page of the We.Train site with the header component and its child pages opens, as shown:

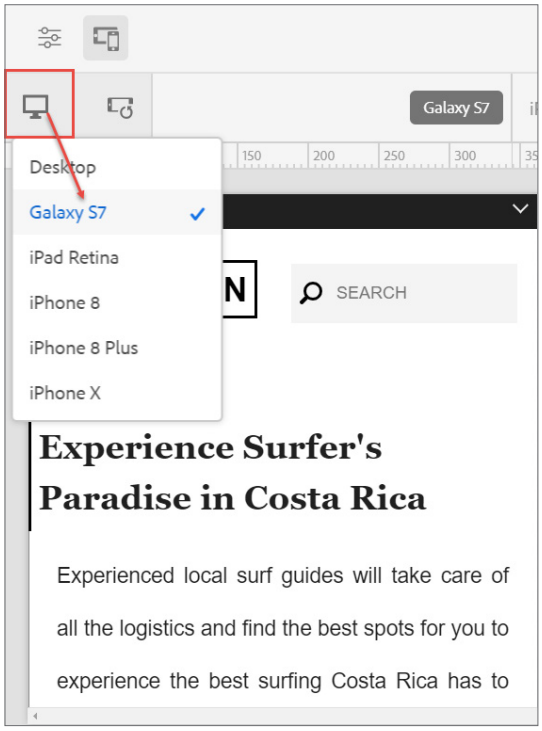
Task 3: Add styling to the header component

You 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.

1. Click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components/structure/header** node.
3. Double-click **header.html** to open the script for editing.
4. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/structure/header** folder.
5. Open the **basic_full_header.html** file in Notepad++ (or any editor of your choice), copy the content of the file, and replace the existing content in the **header.html** editor in **CRXDE Lite**. with the new content.
6. Click **Save All**.
7. Review the code you added. Notice in the 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. Also, notice all the class names are incorporated. These are added to correlate with the WKND design.
8. Navigate to the **English** page of the We.Train site and refresh the page.

9. Click **Preview**. The **Preview** mode opens
10. Click the **Emulator** icon, as shown:

11. Click various devices such as Galaxy S7, iPhone 8 Plus, iPad Retina, and iPhone X, and notice that the header is responsive.



You have now created a new structure component, added it to the editable template design, and extended it with a responsive design.

Exercise 2: Add JavaScript business logic to the header component

In this exercise, you will use JavaScript to add business logic to the header component. Using JavaScript is a quick way to add simple business logic to a component that is not needed for Content Services. For heavier business logic in a component, Sling Models should be used. The next exercise covers Sling Models.

1. Click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components/structure/header** node.
3. Right-click the **header** node and click **Create** > **Create File**. The **Create File** dialog box opens.
4. Enter **header.js** in the **Name** box and click **OK**. The **header.js** file is created. Notice that the **header.js** file is open for editing at the right.

To add code to the header.js file:

5. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/structure/header** folder.
6. Open the **header.js** file in Notepad++ (or any editor of your choice), copy the content of the file, and paste it in the **header.js** editor in **CRXDE Lite**.
7. Click **Save All**.

8. Review the code you added now. In the following code snippet:
   a. Line 4 gets the language root of the website. That is, We.Train (1) > English (2) >... and so on.
   b. Line 12 displays a log message in the error.log file.
   c. Line 14 gets all children under the language root that have not been filtered by the PageFilter.

```
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.     return {
21.         items: items,
22.         rootPage: root
23.     }
24. });
```

To update the code in the header.html page:

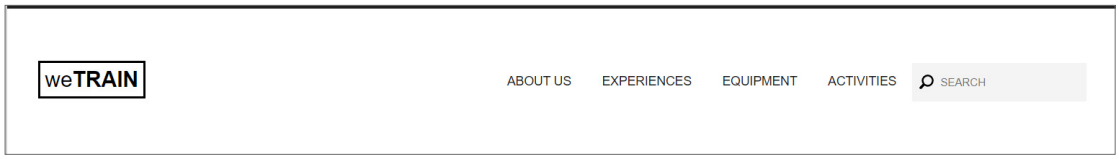
9. Double-click the **header.html** to open the script for editing. The editor opens at the right.
10. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/structure/header** folder.
11. Open the **js_header.html** file in Notepad++ (or any editor of your choice), copy the content of the file and replace the existing content in the **header.html** editor in **CRXDE Lite** with the new content.
12. Click **Save All**.
13. Open **header.html** to view the code you added. Notice the following:
    a. Line 2 defines the header object returned from header.js.
    b. Line 18 uses the header object to get the rootPage.path.

c. Lines 24–34 have been updated to iterate over the children pages in the header.items page array.

To view the changes on the English page:

14. Navigate to the **English** page of the We.Train site and refresh the page. If the **English** page is not open, click http://localhost:4502/editor.html/content/we-train/en.html. .

15. Click **Preview** from the page toolbar and notice that the header component has the same functionality as in the previous exercise.

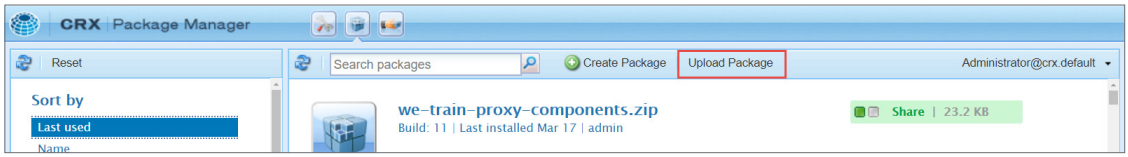


Exercise 3: Add a Sling Model business logic to the header component

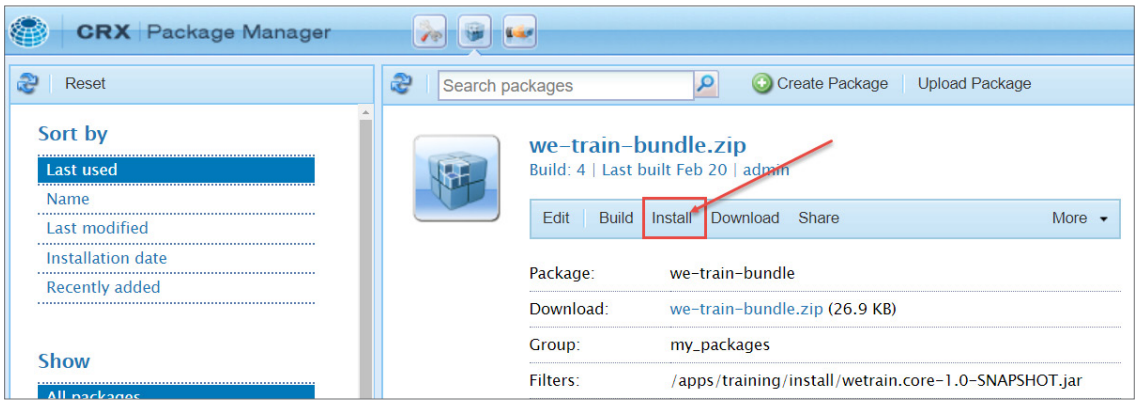
In this exercise, you will use a Sling Model for the business logic of the component. Using Sling Models enable heavier computation as well as extensibility for headless implementations. Sling Models are java classes compiled into an OSGi bundle.

For convenience, you will install a content package that contains the we.train.core-0.0.1-SNAPSHOT.jar file (OSGi Bundle) under an /install folder in the JCR. OSGi listens to these types of folders and automatically installs bundles in this location.

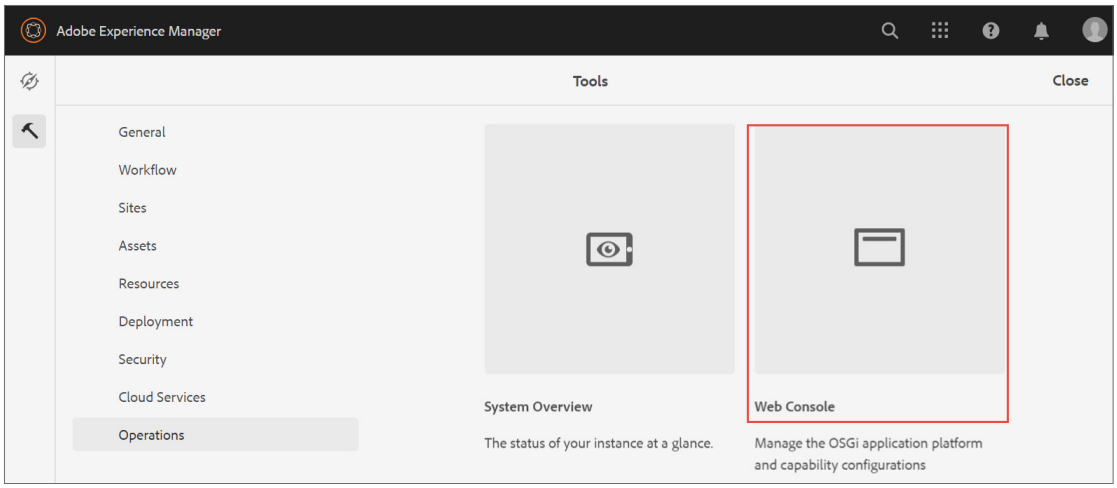
1. Click http://localhost:4502/crx/packmgr/index.jsp. The **Package Manager** opens.
2. Click **Upload Package**, as shown. The **Upload Package** dialog box opens.



3. Click **Browse** and navigate to the Exercise Files folder provided to you.
4. From the training-files/component-basics folder, select we-train-bundle.zip content package.
5. Click **OK**. The package is uploaded.
6. Click **Install,** as shown. The package is installed. Notice how the bundle was installed into the JCR at /apps/training/install. As a result, OSGi automatically installs it into the OSGi container.

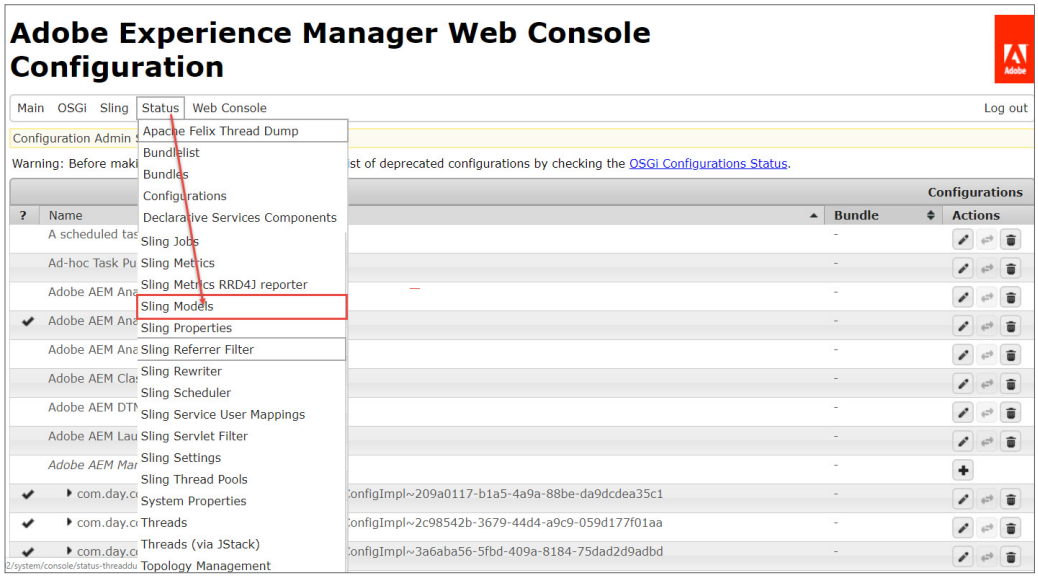


7. To verify if the bundle is installed successfully on your instance, in your AEM author service, navigate to **Tools** > **Operations** > **Web Console**, as shown. The **Adobe Experience Manager Web Console Configuration** page opens.

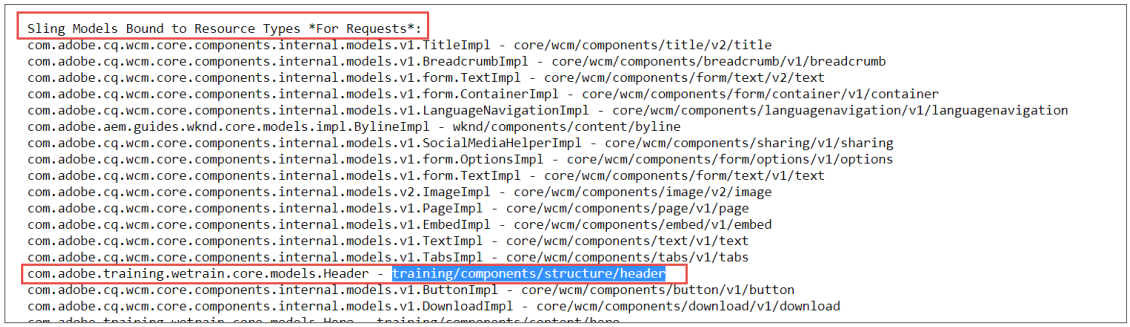


8. Click **OSGi** > **Bundles**. Notice that the We.Train - Core bundle is now active in AEM.

9. Click **Status** on the header bar and select **Sling Models** from the drop-down menu, as shown. The **Adobe Experience Manager Web Console Sling Models** page opens.



10. Press Ctrl+F (Windows) or Command+F (Mac) on your keyboard and type **training/components/structure/header** in the search field. Notice that the **com.adobe.training.wetrain.core.models.Header - training/components/structure/header** is under **Sling Models Bound to Resource Types *For Requests*** (if you cannot find this, press **Ctrl+F** or **Command+F**). This confirms that the bundle is installed successfully.



Now that the sling model is installed, you can use it in the header HTL code:

11. Click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.
12. Navigate to the **/apps/training/components/structure/header** node.
13. Double-click the **header.html** to open the script for editing. The editor opens at the right.
14. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/structure/header** folder.
15. Copy the code from **model_header.html** file and replace the existing content in the **header.html** editor in **CRXDE Lite** with the new content.
16. Click **Save All**.
17. Review the code you added in **header.html**. Notice that the model_header.html script is the same as js_header.html, except you are requesting the Sling Model rather than the Javacript.

```
<div class="cmp-experiencefragment--header"
    data-sly-use.header="com.adobe.training.wetrain.core.models.Header">
    <div class="cmp-layoutcontainer--utility aem-GridColumn aem-GridColumn--default--12">
        <div class="aem-Grid aem-Grid--12">

            <!--/* Adds the language navigation core component */-->
            <div class="languagenavigation cmp-languagenavigation--default cmp-languagenavigation--dark cmp-languagenavigation--header
                        aem-GridColumn aem-GridColumn--offset--default--10 aem-GridColumn--offset--phone--8"
                data-sly-resource="${'languagenavigation' @ resourceType='training/components/structure/languagenavigation'}">
            </div>
        </div>
    </div>
    <div class="cmp-layoutcontainer--header">
        <div class="aem-Grid aem-Grid--12">
            <!--/* Adds the site logo */-->
            <div
                class="wknd-header--logo aem-GridColumn aem-GridColumn--offset--tablet--1 aem-GridColumn--offset--phone--1 aem-GridColumn--tablet--
-6 aem-GridColumn--default--2">
                <a href="${header.rootPage.path @ extension='html'}">we<strong>TRAIN</strong></a>
            </div>

            <!--/* Adds the site navigation */-->
            <div class="cmp-navigation--header aem-GridColumn aem-GridColumn--default--8 aem-GridColumn--tablet--3">
                <nav class="cmp-navigation">
                    <ul class="cmp-navigation__group" data-sly-list="${header.items}">
                        <!--/**************************************************************
                             * Using HTL we can quickly create a list of links for the header *
                             * Notice how the sling model defined above as 'header' is used   *
                             * to get the list of pages that should be displayed.             *
                             **************************************************************/-->
                        <li class="cmp-navigation__item cmp-navigation__item--level-1 ${item.title==currentPage.title ? ' cmp-navigation__item--
active' : ''}">
                            <a class="cmp-navigation__item-link"
                                href="${item.path @ extension = 'html'}">${item.title}</a>
```

To view the changes on the English page:

18. Navigate to the **English** page of the We.Train site and refresh the page. (If the **English** page is not open, click http://localhost:4502/editor.html/content/we-train/en.html).
19. Click **Preview** from the page toolbar, and ensure you can see the subpages of the Activities page.

Configuring the Edit Dialog

Dialogs are a key element of a component as they provide an interface for authors to configure and provide input to the component.

Dialogs are a part of the Granite UI Foundation that uses Coral 3 UI components and styles. As the dialogs you will create are an extension of the Granite UI foundation, you can use many of the properties, concepts, and design patterns defined in the Granite UI documentation.

Due to the dialog node structure complexity, you should reuse dialog node structures from other components as much as possible. Inheriting a dialog box from a core component or copying one are both viable options. If your component is extending a core component, it can inherit the dialogs' functionality and potentially extend it. Otherwise, if you are creating a completely unique component, study the dialogs of the core components to identify potential dialog fields you could use, copy the node structure from the core component dialog, and customize as needed.

Exercise 4: Create an edit dialog for a component

In this exercise, you will create a new component that will include a custom dialog for authoring input. In this exercise, you will perform the following tasks:

1. Create a hero component
2. Create the hero edit dialog

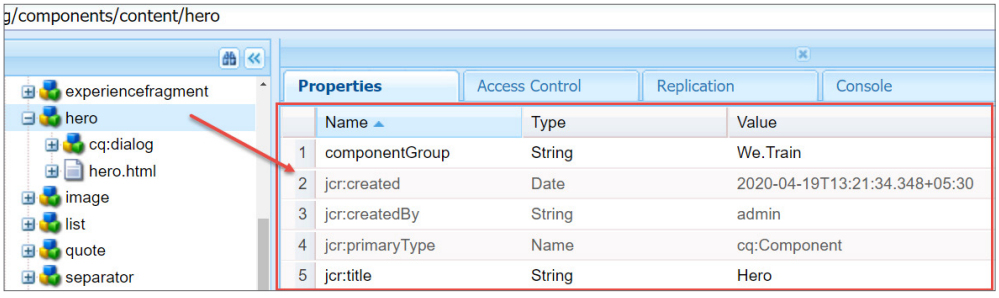
Task 1: Create a hero component

In this task, you will create another component from the start. The first iteration of this component will use the init_hero.html script. This will result in only a placeholder showing up on the page because there is no dialog for input.

1. In the **CRXDE Lite** page (http://localhost:4502/crx/de.), navigate to the **/apps/training/components/content** folder.
2. Right-click the **content** folder and click **Create** > **Create Node.** The **Create Node** dialog box opens.
3. Enter the following values in the dialog box:
   a. **Name**: **hero**
   b. **Type: cq:Component**
4. Click **OK**. The **hero** node is created.
5. Ensure the **hero** node is selected and add the following properties:

Name	Type	Value
jcr:title	String	Hero
componentGroup	String	We.Train

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



7. Right-click the **hero** component and click **Create** > **Create File.** The **Create File** dialog box opens.
8. In the **Name** box, enter **hero.html** and click **OK**. Notice that the file is created and the **hero.html** editor opens at the right.
9. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/content/hero** folder.
10. Open **init_hero.html** file in Notepad ++ (or any editor of your choice), copy its content, and paste it in the **hero.html** file in **CRXDE Lite**.
11. Review the code you just included in **hero.html**.

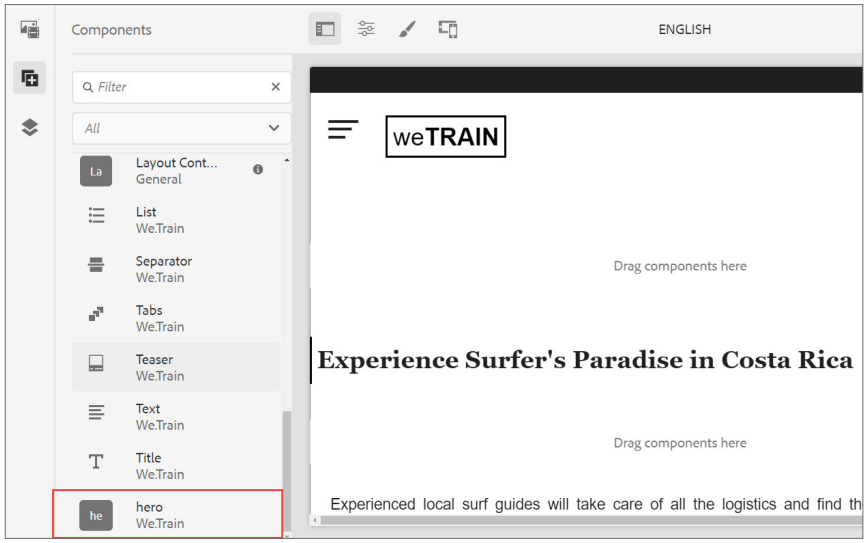
```
01. <div data-sly-use.template="core/wcm/components/commons/v1/templates.html">
02.
03. </div>
04.
05. <!--/* component placeholder */-->
06. <sly data-sly-call="${template.placeholder @ isEmpty=true}"></sly>
```

12. Click **Save All**.

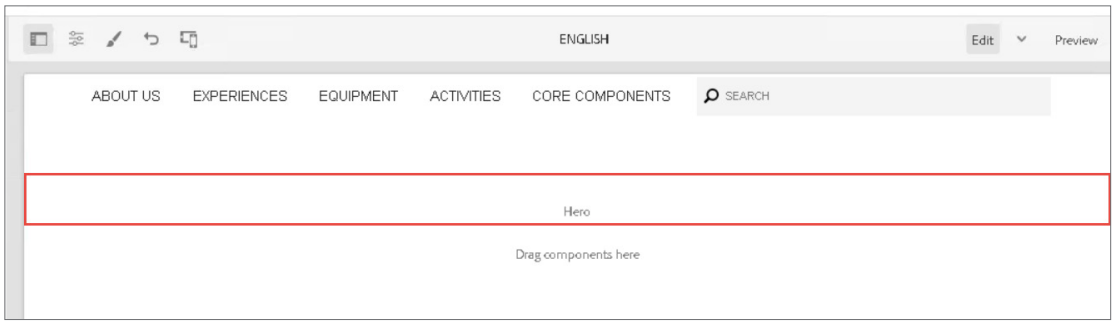
Next, you need to add a cq:dialog node to the hero component. This cq:dialog node will be used as a placeholder initially

13. Select the **hero** component and click **Create** > **Create Node**. The **Create Node** dialog box opens.
14. Update the following values:
    a. **Name**: **cq:dialog**
    b. **Type**: **nt:unstructured**
15. Click **OK**.
16. Click **Save All**.
17. Click **CRXDE Lite** from the header bar to navigate back to AEM.

18. Navigate to **Sites** > **We.Train** > **English** page.
19. Select the **English** page and click **Edit**. The English page opens in the edit mode.
20. Click the **Toggle Side Panel** icon on the top-left, if you do not see the left panel.
21. Click the **Components** tab on the left. You should now see the **Hero** component, as shown. The **Hero** component was automatically added to the list because the We.Train componentGroup is allowed on the layout container for the page.



22. Drag the **Hero** component onto the page and observe the placeholder.



Task 2: Create the hero edit dialog

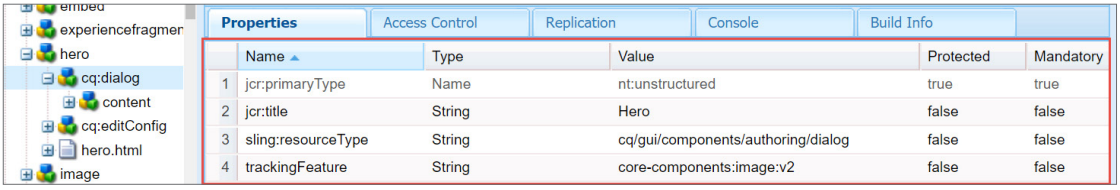
In the previous task, you created a simple hero component. In this task, you will ensure authors have the ability to add content to the component. You will copy an existing dialog structure and customize it to the hero component.

1. Click http://localhost:4502/crx/de. 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 needs.

2. Navigate to the **/apps/training/components/content/hero** node.
3. Delete the **cq:dialog** node below the **hero** node.

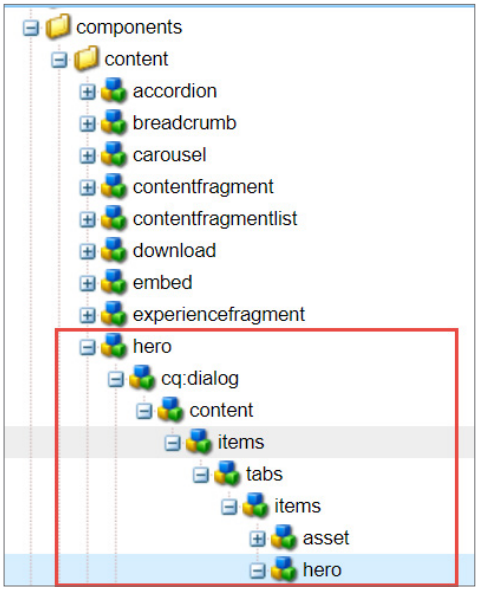
4. Click **Save All**.
5. Navigate to the **/apps/core/wcm/components/image/v2/image** node.
6. Right-click the **cq:dialog** node and click **Copy**. The node is copied.
7. Navigate to the **/apps/training/components/content** folder.
8. Right-click the **hero** component and click **Paste**. The cq:dialog node of Core Image is added to the hero component.
9. Click **Save All**.
10. Select the **cq:dialog** node, and on the **Properties** tab, double-click **jcr:title** and update the value as **Hero**.
11. Right-click the **extraClientlibs** property and click **Delete** to delete it.
12. Similarly, delete the **helpPath** property.
13. Click **Save All**. The **Properties** tab should look as shown:



	Name	Type	Value	Protected	Mandatory
1	jcr:primaryType	Name	nt:unstructured	true	true
2	jcr:title	String	Hero	false	false
3	sling:resourceType	String	cq/gui/components/authoring/dialog	false	false
4	trackingFeature	String	core-components:image:v2	false	false

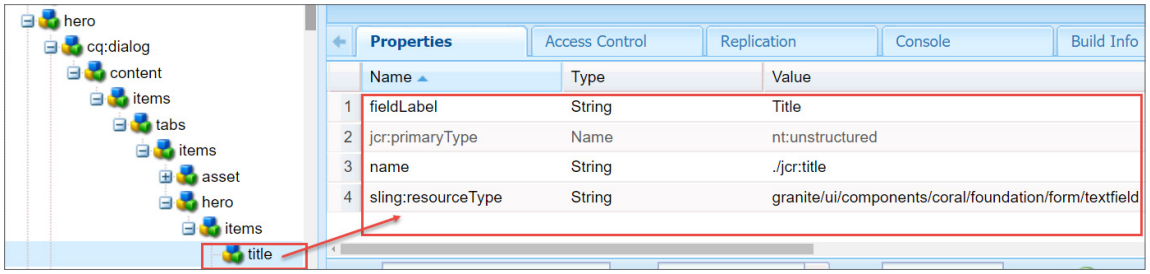
14. Under the hero component, navigate to the **cq:dialog/content/items/tabs/items/metadata/items/columns** node.
15. Right-click the **columns** node and click **Delete** to delete it.
16. Click **Save All**.
17. Navigate to the **cq:dialog/content/items/tabs/items/metadata** node.
18. Right-click the metadata node, click **Rename** and rename it as **hero**.

19. Click **Save All**. The node structure should look, as shown:



20. Select the **hero** node you renamed now, and in the **Properties** tab, double-click **jcr:title**.
21. In the **Value** box, enter **Hero**.
22. Click **Save All**.
23. Right-click the **items** node that is below the **hero** node and click **Create** > **Create Node**. The **Create Node** dialog box opens.
24. Update the following details:
    a. **Name**: **title**
    b. **Type**: **nt:unstructured**
25. Click **OK** > **Save All** to save the changes.
26. 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/coral/foundation/form/textfield



27. 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 and paste the **title** node and update the required values accordingly.

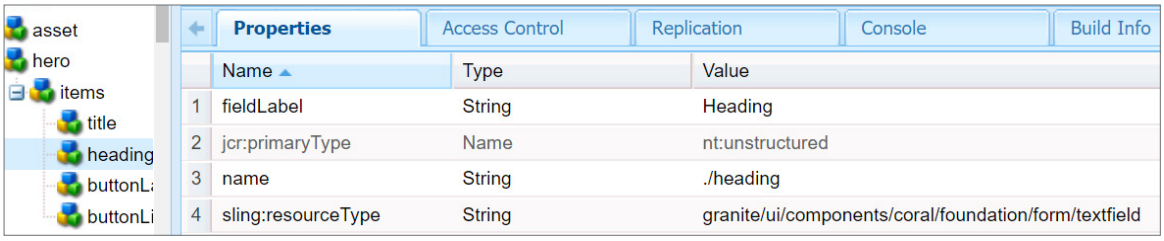
28. Right-click the **title** node and click **Copy** to copy it.

29. Select the **/hero/items** node, right-click **items** and click **Paste**. The **Copy of title** node is created.

30. Rename the **Copy of title** node to **heading**.

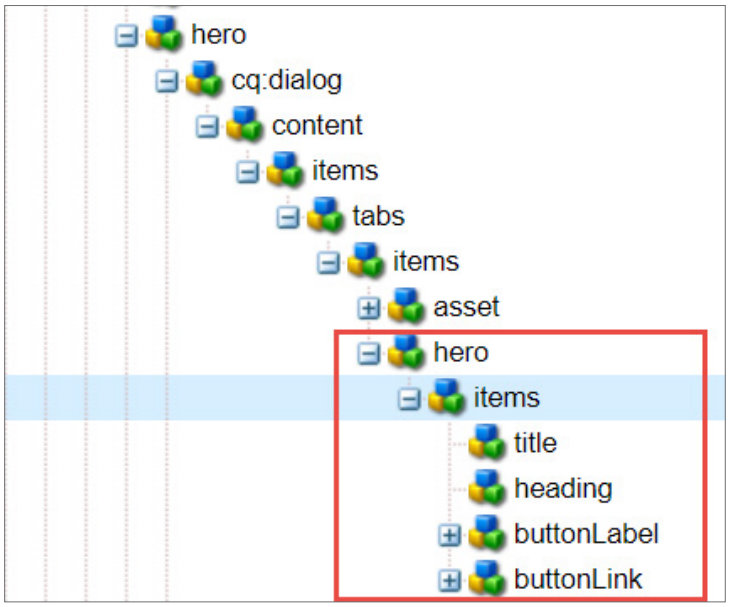
31. Select the **heading** node and update the values of the following properties:

Name	Type	Value
fieldLabel	String	Heading
name	String	./heading
sling:resourceType	String	granite/ui/components/coral/foundation/form/textfield



32. Click **Save All**.

33. Perform steps 29 to 31, and create **buttonLabel** and **buttonLink** nodes below the **/hero/items** node. The field structure should look as shown:



34. 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/coral/foundation/form/textfield

35. Click **Save All**.

36. 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/coral/foundation/form/pathbrowser

37. Click **Save All**.

To test the dialog box, it needs to be opened on a page:

38. In **CRXDE Lite**, double-click the **hero.html** file to open it for editing.

39. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/content/hero** folder.

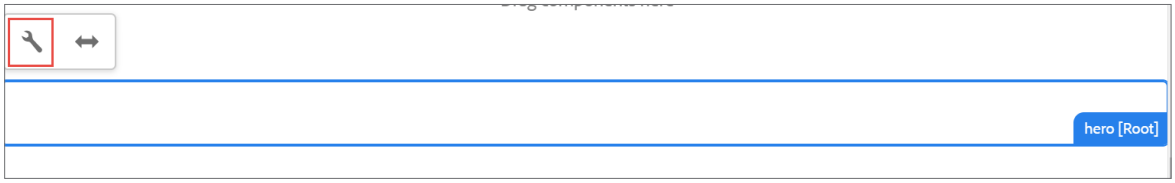
40. Open **model_hero.html** file in Notepad ++ (or any editor of your choice), copy its content, and replace the existing content in the **hero.html** file in **CRXDE Lite** with the new content.

41. Click **Save All**.

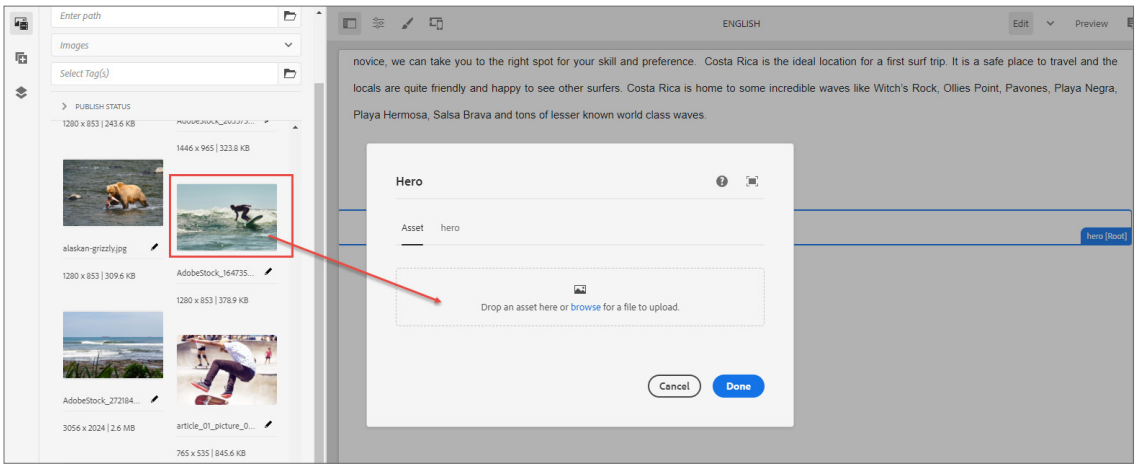
42. Navigate back to **We.Train** > **English** page. (http://localhost:4502/editor.html/content/we-train/en.html).

43. If you have not already dragged a Hero component onto the page, drag the Hero component onto the Layout Container.

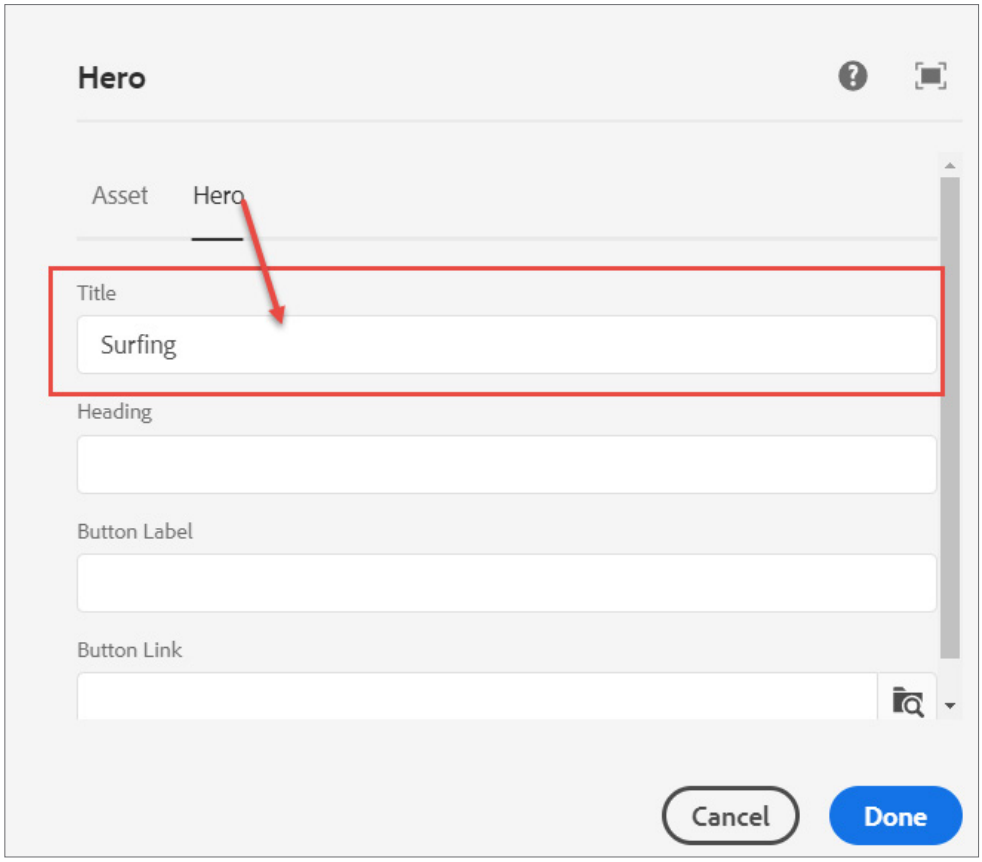
44. Select the **hero[Root]** component and click the **Configure** icon (wrench icon), as shown. The **Hero** dialog box opens.



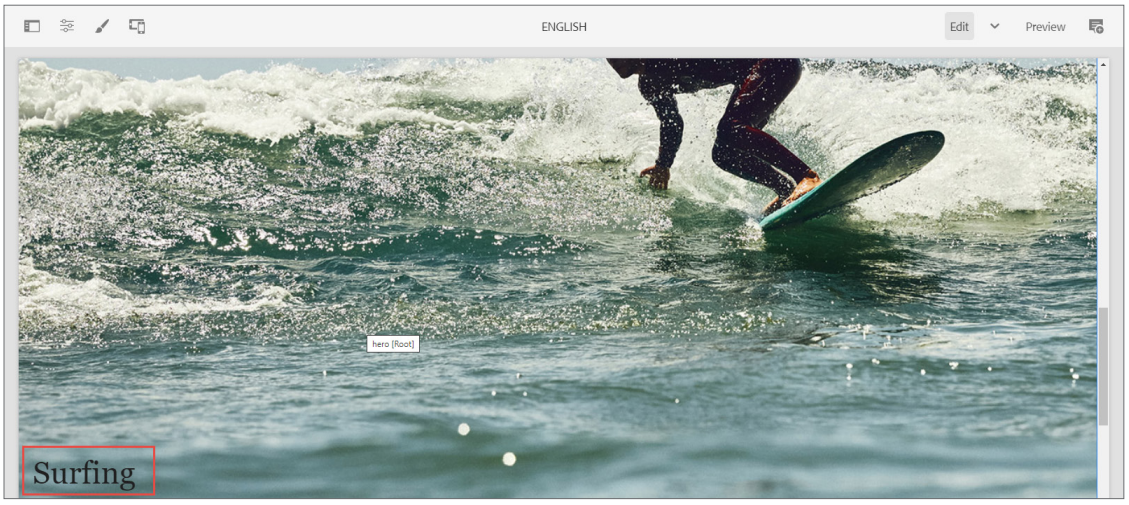
45. Drag an image from the Assets tab on the left onto the Hero dialog box, as shown:



46. Click the **Hero** tab, and add a title in the **Title** box. For example, Surfing.



47. Click **Done** to save the changes. If you see an image with a title, as shown, you have successfully created a dialog box with your component.



Note: In this exercise, you used a Sling model as the business logic. This allows the component to render as JSON for a headless implementation. If JSON output is not needed, the same business logic for this component can be accomplished with pure HTL. You can see this in the script ui.apps/.../training/components/content/hero/purehtl_hero.html

Exercise 5: Add an editconfig node to the component

In an editable UI, authors want the convenience of fewer clicks. An editConfig node enables a streamlined authoring experience. In this exercise, you will create an editConfig node that enables users to drag an asset from the Assets rail onto the hero component without opening the dialog box.

To add an editConfig node:

1. Click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/core/wcm/components/image/v2/image** node.
3. Right-click the **cq:editConfig** node and click **Copy**. The node is copied.
4. Navigate to the **/apps/training/components/content** folder.
5. Right-click the **hero** component and click **Paste** to paste the node you copied in step 3. The cq:editConfig node is added.
6. Expand the **cq:editConfig** node by clicking the + icon, right-click the **cq:inplaceEditing** node, and click **Delete** to delete the cq:inplaceEditing node.
7. Click **Save All**.
8. Under the hero component, navigate to the **cq:editConfig/cq:dropTargets/image** node, select the **parameters** node, and add the following property:

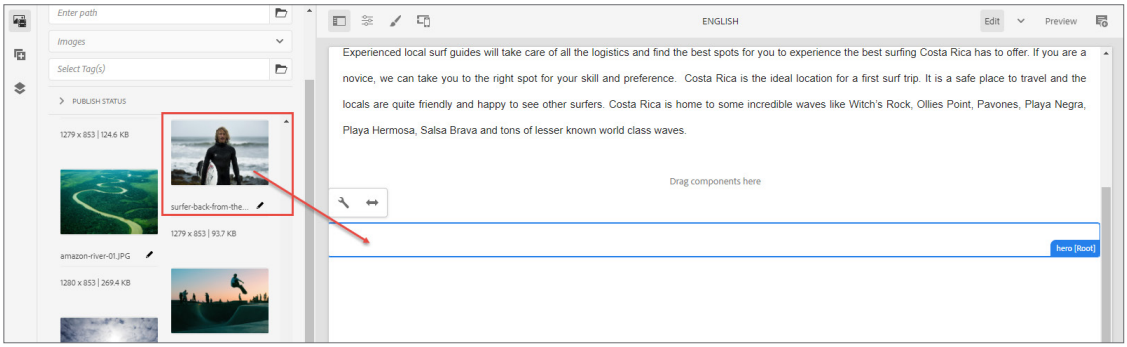
Name	Type	Value
sling:resourceType	String	training/components/content/hero

9. Click **Save All**.

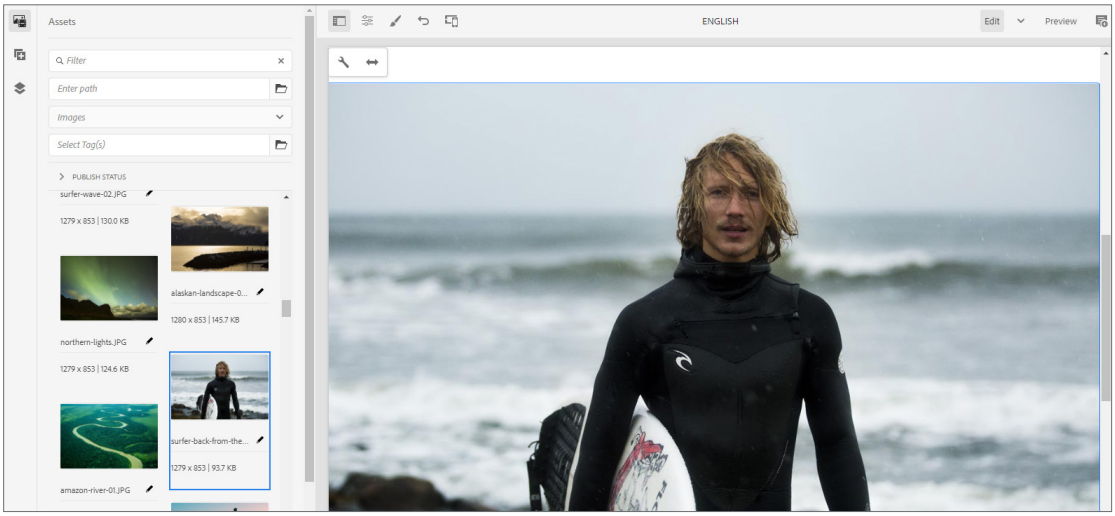
To test if the cq:editConfig is added to the hero component:

10. Navigate to the **English** page of the We.Train site. (http://localhost:4502/editor.html/content/we-train/en.html)
11. Click the **Toggle Side Panel** icon from the page toolbar if you do not see the left panel and click the **Assets** icon. The **Assets** panel opens.

12. Drag an image from the **Assets** panel onto the **hero** component, as shown:



The image is added to the component, as shown. This indicates the success of this exercise. You used cq:editConfig to add the ability of your hero component to accept content through the drag-and-drop feature.



Design Dialogs

Design dialogs are similar to the dialogs used to edit and configure content, but they provide the interface for authors to configure and provide the 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 map asset to 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 the following location by default:
/conf/project_name/settings/wcm/policies/component_name/policy_randomNumber

You can also find policies in the /apps or /libs folder. In this case, the resource will be resolved in the following order: /conf, /apps, and /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: Create a design dialog for a component

In this exercise, you will create a new component with a design dialog, and then create a supporting policy that will be applied to all pages created from the template.

In this exercise, you will perform the following tasks:

1. Create a footer component
2. Create a design dialog for a content policy

Task 1: Create a footer component

1. Ensure the **CRXDE Lite** is open, or click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components** folder.
3. Right-click the **structure** folder, and click **Create** > **Create Node**. The **Create Node** dialog box opens.
4. Provide the following details:
   a. **Name: footer**
   b. **Type: cq:Component**
5. Click **Save All**.
6. Ensure the footer node is selected and add the following properties:

Name	Type	Value
jcr:title	String	We.Train Footer
componentGroup	String	We.Train.structure

7. Click **Save All**.
8. Right-click the **footer** component and click **Create** > **Create File**. The **Create File** dialog box opens.
9. In the **Name** box, enter **footer.html** and click **OK**. Notice that the **footer.html** editor opens at the right.
10. Click **Save All**.

To add code to the footer.html:

11. In the Exercise Files folder provided to you, navigate to the **ui.apps/src/main/content/jcr_root/apps/training/components/structure/footer** folder.
12. Open the **model_footer.html** file in Notepad++ (or any editor of your choice), copy the content of the file, and update the **footer.html** editor in **CRXDE Lite**.
    The footer component uses a Sling Model called **Footer.java**. The bundle, we.train.core-0.0.1-SNAPSHOT.jar you installed previously already has this model built. If the bundle is not installed, you can install it through the Web Console using the **we.train.core-0.0.1-SNAPSHOT.jar** provided to you.
13. Click **Save All** to save changes.

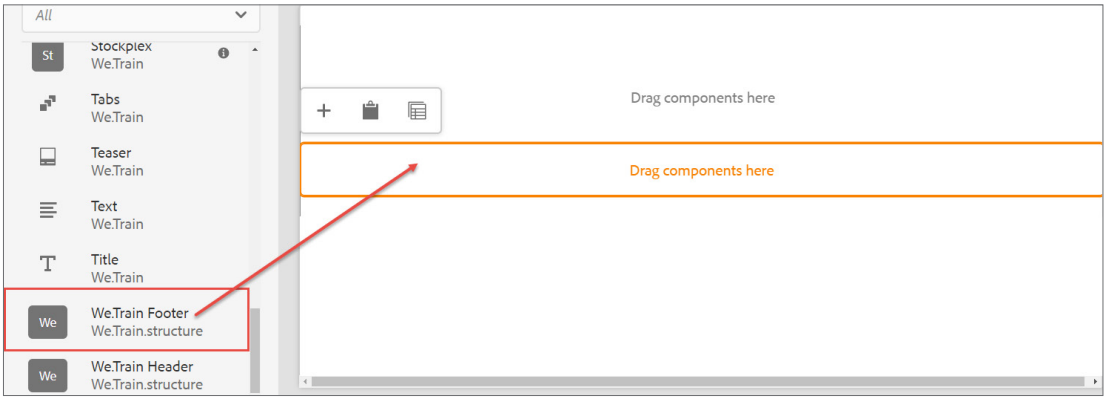
Next, you need to add a cq:dialog node to add the footer component to the template:

14. Select the **footer** component and click **Create** > **Create Node**. The **Create Node** dialog box opens.
15. Provide the following details:
    a. **Name**: **cq:dialog**
    b. **Type**: **nt:unstructured**
16. Click **OK** and click **Save All**.

To add the footer component to the template:

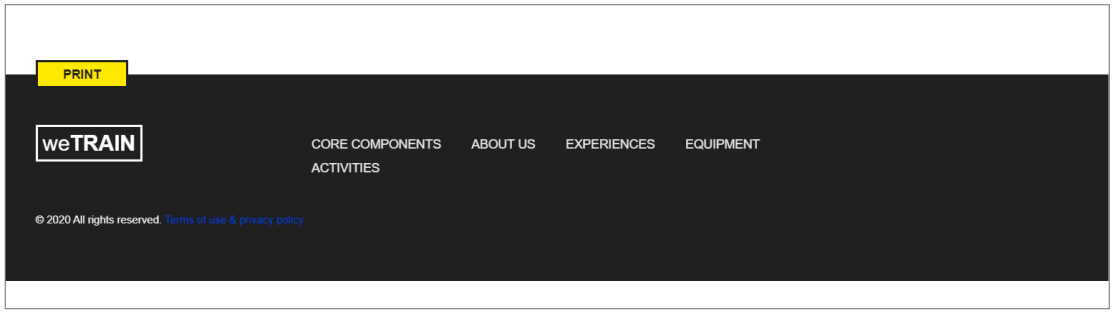
17. In your AEM author service, click **Adobe Experience Manager** from the header bar, click the **Tools** icon and click **Templates**. The **Templates console** opens.
18. Click the **We.Train** folder, select the **Content Page** template and click **Edit (*e*)** from the actions bar.
19. Click the **Toggle Side Panel** icon from the page toolbar, and click the **Components** icon. The **Components** panel opens. Notice that **We.Train Footer** is available.

20. Drag the **We.Train Footer** component from the **Components** panel onto the **Drag components here** placeholder, as shown. The component is added.



Notice that the footer component is added to the template.

21. As the footer component was added to the template, it was automatically added to all pages created from it. Refresh the English page and notice how the footer is rendering.

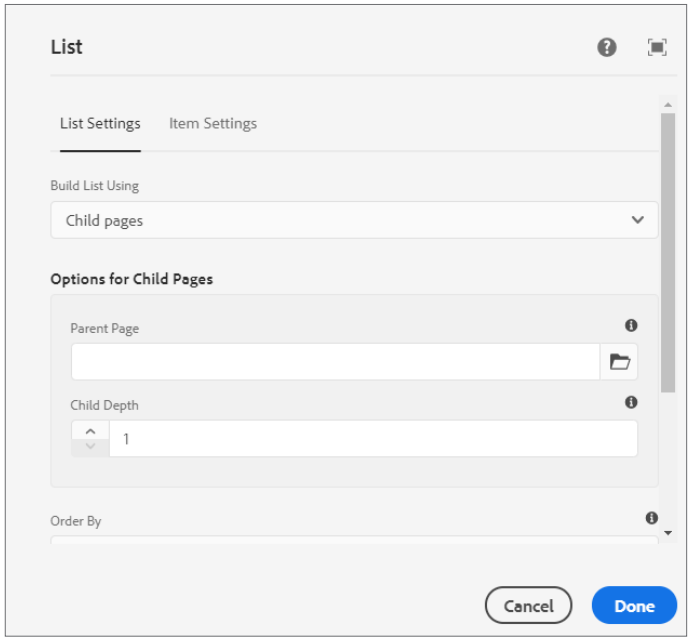


Task 2: Create a design dialog for a content policy

Sometimes, organizations may want to update and change the links on the 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 the beginning, you will build it based on the list component dialog. To see this dialog:

1. Ensure the **English** page is open.
2. Select the **Drag components here** area, click the **Insert component** icon (+ icon). The **Insert New Component** dialog box opens.
3. Add a **List** component.
4. Select the **List** component and click the **Configure** icon (wrench icon) from the component toolbar. The **List** dialog box opens. Observe the dialog box and the options it provides.



5. Click **Cancel** to close the dialog box.

To create a design dialog:

6. In **CRXDE Lite**, navigate to the **/apps/training/components/structure** folder.

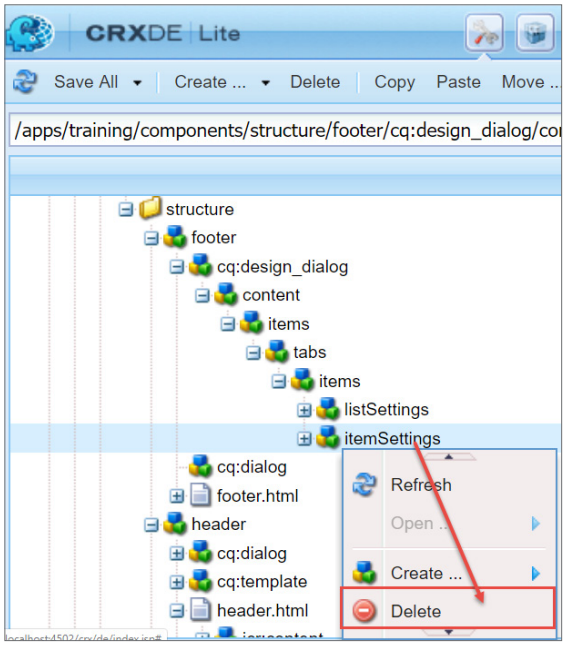
7. Right-click the **footer** node and click **Create** > **Create Node** The **Create Node** dialog box opens.
8. Provide the following details:
   a. **Name**: **cq:design_dialog**
   b. **Type: nt:unstructured**.
9. Click **OK**. The node is created.
10. Click **Save All**.
11. Select the **cq:design_dialog** node and add the following properties:

Name	Type	Value
sling:resourceType	String	cq/gui/components/authoring/dialog
jcr:title	String	Footer Policy

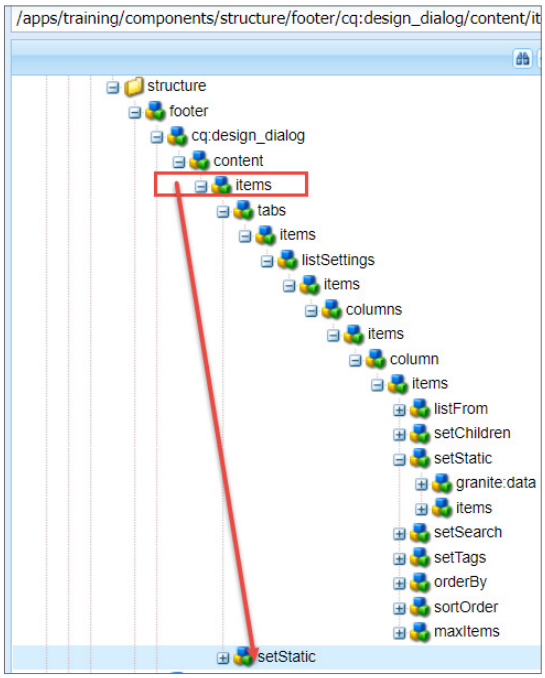
12. Click **Save All**.
13. Navigate to the **/apps/core/wcm/components/list/v2/list/cq:dialog** node, select the **content** node and click **Copy** from the actions bar.
14. Navigate to the **/apps/training/components/structure/footer** node, select the **cq:design_dialog** node and click **Paste**.
15. Click **Save All**.

Next, you will remove all the extra fields from the dialog and customize it to your footer component.

16. Under the footer node, navigate to the **cq:design_dialog/content/items/tabs/items** node, right-click the **itemSettings** node and click **Delete**, as shown. The node is deleted.

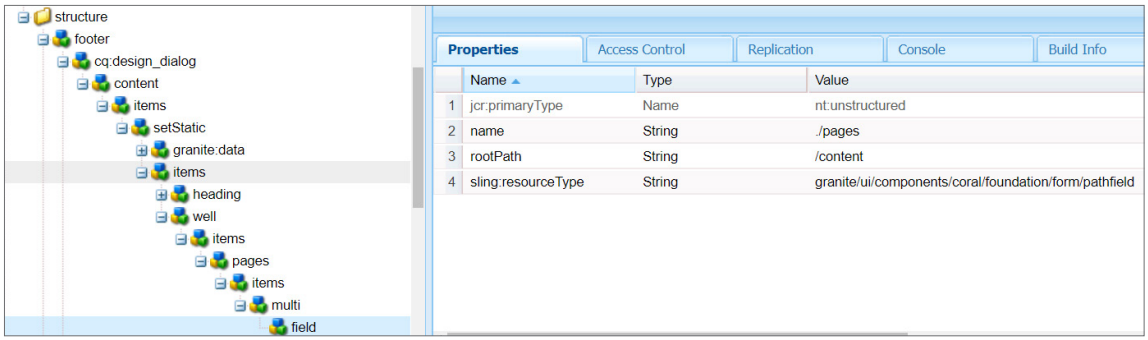


17. Click **Save All**.
18. Under the footer node, navigate to the **cq:design_dialog/content/items/tabs/items/listSettings/items/columns/items/column/items** node.
19. Right-click the **setStatic** node and click **Copy**.
20. Navigate to the **cq:design_dialog/content/items** node and paste the node you copied in the previous step.



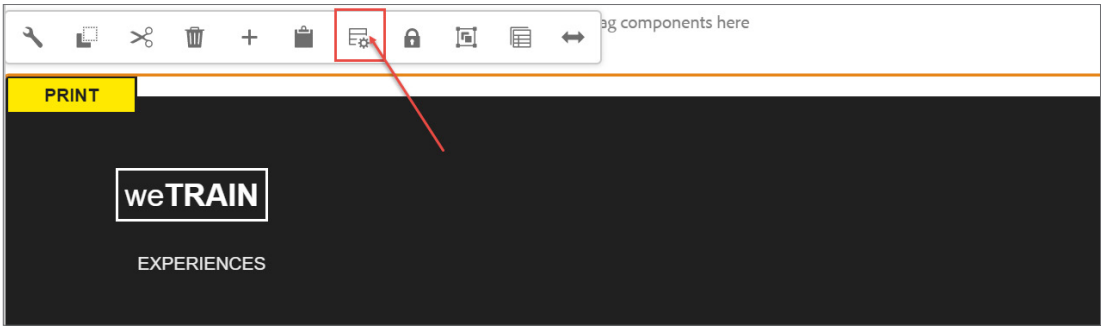
21. Click **Save All**.
22. Navigate to the **cq:design_dialog/content/items/setStatic** node.
23. On the **Properties** tab, right-click the **granite:class** property and click **Delete**. The property is deleted from the node.
24. Click **Save All**.
25. Navigate to the **cq:design_dialog/content/items** node.
26. Right-click the **tabs** node, and click **Delete**.

27. Navigate to the **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 be as shown:



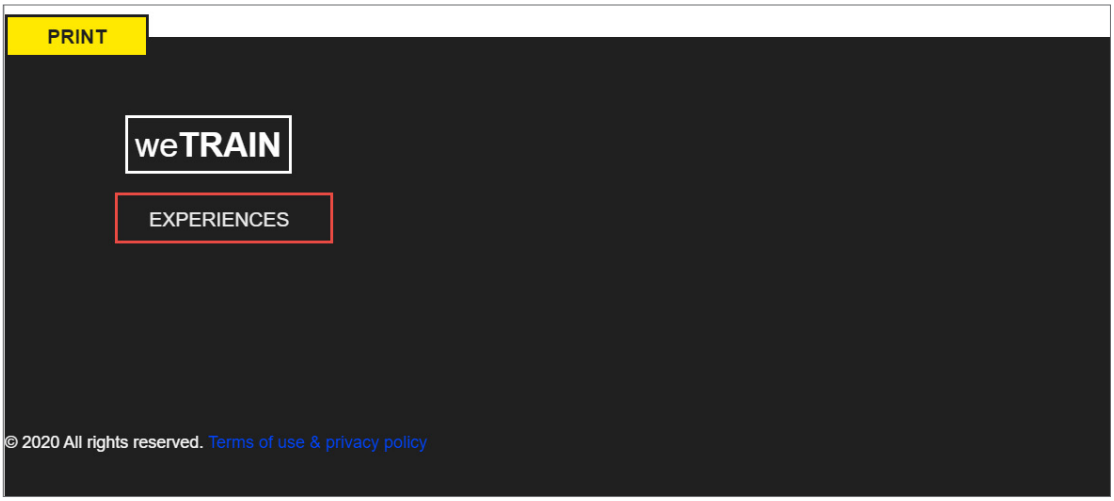
Now that the design dialog is built, you can create a content policy in the template editor.

28. Open the **Content Page** template, select the **Footer** component, and click the **Policy** icon from the component toolbar, as shown. The **Footer Policy** dialog box opens.



29. In the **Policy** section, in the **Policy Title** box, type **We.Train Footer Policy**.
30. In the **Properties** section, on the **List Settings** tab, select **Fixed List** from the **Build List Using** drop-down menu.
31. Under **Options for Fixed List**, click **Add**, and type **/content/we-train/us/en/experience**.
32. Click **Done**.

33. Navigate to the **English** page of the We.Train site. Notice that the footer is updated with a link to the **Experience** page that you added in the previous step.



34. Click **Preview** from the page toolbar, and click **Experiences** in the footer to navigate to the **Experiences** page of the We.Retail site.

35. Open other pages of the We.Train site and observe the changes in the content policy.

You successfully created a component with a design dialog to allow new options to be set in the component's content policy. You also saw that the options set in a component's content policy, through the design dialog, affect all pages in the site using that policy.

References

- Customizing Components
- Customizing Dialogs

Creating Custom Components

Introduction

Adobe Experience Manager (AEM) components are used to hold, format, and render the content 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 the beginning.

Objectives

After completing this module, you will be able to:

- Explain the important features of components
- Create a custom component
- Add a dialog field validation to the custom component
- Add a base style to the custom component
- Add a style system to the custom component
- Add a Sling model as the business logic
- Create localization information

Features of Components

Some common component features are:

- **Component Placeholder**: Helps validate if the component is on the page and is available only on the page Edit mode. The placeholder is not available on the page Preview mode or on the publish service. If the content is 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 with 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 provide alternative visual variations of a component. 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 the script to 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 slash character, or the end of the request URL) comprises the selectors. If the resource path spans the complete request URL, no selectors exist. Also, 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 both 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 the console. You can either create a new dictionary or add a dictionary to an existing project.

Exercise 1: Create a custom component

You need to create a custom component called Stockplex component to display stock information based on the user's specification. The user should be able to change the design as well as export the content as JSON.

To create the Stockplex component:

1. Navigate to **Adobe Experience Manager** > **Tools** > **CRXDE Lite**. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components** folder.
3. Right-click the **content** folder and click **Create** > **Create Node**. The **Create Node** dialog box opens.
4. Enter the following details:
   a. **Name: stockplex**
   b. **Type: cq:Component**
5. Click **OK**. The **stockplex** node is created.
6. Click **Save All** to save the changes.
7. Ensure the **stockplex** node is selected and add the following properties:

Name	Type	Value
jcr:title	String	Stockplex
componentGroup	String	We.Train
jcr:description	String	We.Train complex stock component

8. Click **Save All**.

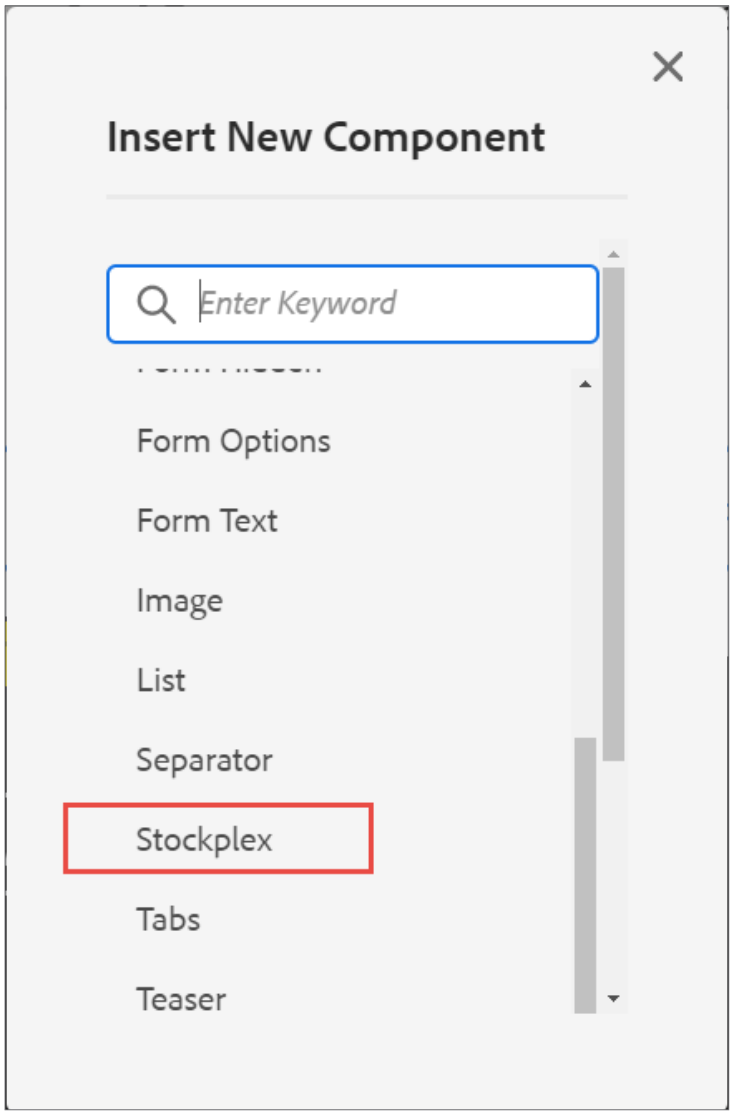
9. Right-click the **stockplex** component and click **Create** > **Create File**. The **Create File** dialog box opens.
10. In the **Name** box, enter **stockplex.html** and click **OK**. The file is created. Notice that the editor opens on the right.
11. Click **Save All**.
12. In the Exercise Files folder provided to you, navigate to ui.apps/src/main/content/jcr_root/apps/training/components/content/stockplex.
13. Open the **placeholder_stockplex.html** file in Notepad ++ (or any editor of your choice), copy its content and paste it in the **stockplex.html** editor in **CRXDE Lite**.
14. Click **Save All**.

You must create a cq:dialog node to enable the component in the authoring UI.

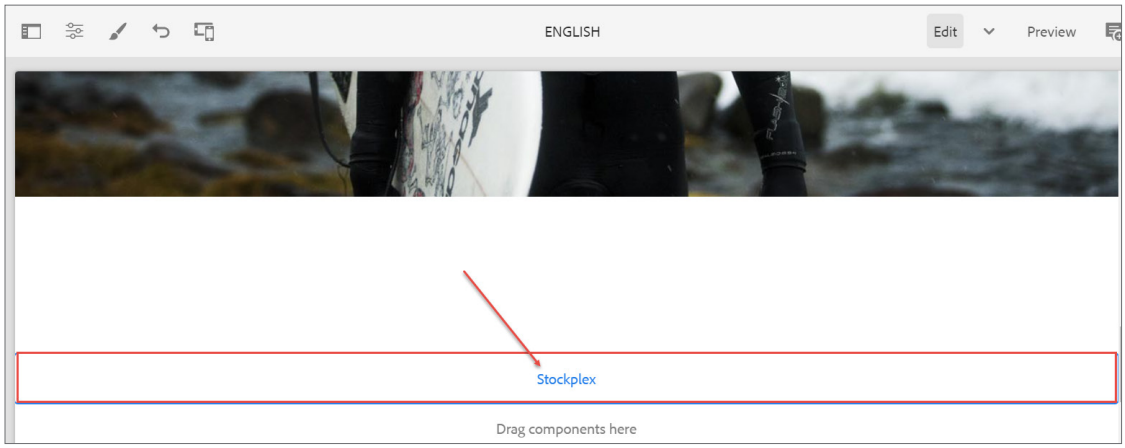
15. Right-click the **stockplex** component and click **Create** > **Create Node**. The **Create Node** dialog box opens.
16. Update the following details:
    a. **Name**: **cq:dialog**
    b. **Type**: **nt:unstructured**
17. Click **OK**. The **cq:dialog** node is created.
18. Click **Save All**.
19. Open the **English** page of the We.Train site in the Edit mode. (http://localhost:4502/editor.html/content/we-train/en.html)
20. On the page, click the **Drag components here** placeholder and click the **Insert component** icon (+ icon) from the component toolbar. The **Insert New Component** dialog box opens.

21. The Stockplex component is now available on the page, as shown, as the component was added to the We.Train group.

Note: If the Stockplex component does not appear on the page, you must enable the component in the Layout Container of the Content Page template.



22. Select the **Stockplex** component from the list. The stockplex component is added to the page, as shown:



In this exercise, you created a custom component and added the necessary elements for an author to be able to add it to the page.

Exercise 2: Add a dialog field validation

In this exercise, you will build a dialog box with a validation formfield. The dialog will be based on a core component dialog, which ensures minimal customization is required. The customization includes creating a validation form field that will ensure the content entered by the author is in the correct format for the component.

In this exercise, you will perform the following tasks:

1. Create the dialog
2. Add Symbol Validation

Task 1: Create the dialog

1. Ensure you are in the **CRXDE Lite** page, or click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components/content** folder.
3. Select the **stockplex** node, and double-click **stockplex.html** to open the script for editing.
4. In the Exercise Files folder provided to you, navigate to ui.apps/src/main/content/jcr_root/apps/training/components/content/stockplex.
5. Open the **dialog_stockplex.html** file in Notepad ++ (or any editor of your choice), copy its content and replace the existing content in the **stockplex.html** editor in **CRXDE Lite** with the new content.

6. Review the code you added. Notice that the data for the component is expected to be in the component properties as symbol, summary, and showStockDetails.

```
<div class="cmp-stockplex" data-sly-use.template="core/wcm/components/commons/v1/templates.html" data-sly-test.symbol="${properties.symbol}">

    <div class="cmp-stockplex__symbol">${properties.symbol}</div>
    <div class="cmp-stockplex__currentPrice">Current Value: 300</div>

    <div data-sly-test.summary="${properties.summary}">
        <h3>Summary: ${properties.summary}</h3>
    </div>

    <div class="cmp-stockplex__button">
        <a href="#">
            <button>Placeholder</button>
        </a>
    </div>

    <div class="cmp-stockplex__details" data-sly-test.summary="${properties.showStockDetails}">
        <ul class="column1">
            <li>Request Date: November 13, 2018</li>
            <li>Open Price: 300</li>
            <li>Range High: 303</li>
        </ul>
        <ul class="column2">
            <li>Range Low: 299</li>
            <li>Close: 303</li>
            <li>Volume: 30000</li>
        </ul>
    </div>
</div>

<!-- If there is no stock symbol added to the dialog, create a component placeholder -->
<sly data-sly-call="${template.placeholder @ isEmpty=!symbol, classAppend='cmp-stockplex'}"></sly>
```

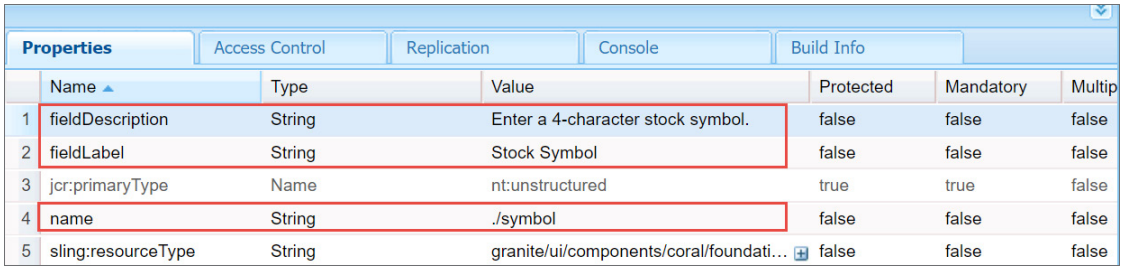
7. Click **Save All**.

Next, you will create a dialog based on an existing dialog to allow author input:

8. Select the **cq:dialog** node that is below the **stockplex** component.
9. Right-click the **cq:dialog** node and click **Delete**. The node is deleted.
10. Click **Save All** from the actions bar.
11. Navigate to the **/apps/core/wcm/components/title/v2/title/cq:dialog** node, right-click the **cq:dialog** node and click **Copy**. The node is copied.
12. Navigate to the **/apps/training/components/content** folder, right-click the **stockplex** node and click **Paste**.
13. On the **cq:dialog** node, update the following properties in the **Properties** section:
    a. Change the value of **jcr:title** to **Stockplex**.
    b. Right-click the **helpPath** property and click **Delete** to delete it.
14. Click **Save All**.
15. Under the **stockplex** node, navigate to the **cq:dialog/content/items/tabs/items/properties/items/columns/items/column/items** node.
16. Right-click the **types** node and click **Delete** to delete it.
17. Right-click the **defaulttypes** node and click **Delete** to delete it.
18. Click **Save All** from the actions bar.
19. Navigate to the **cq:dialog/content/items/tabs/items/properties/items/columns/items/column/items/title** node.
20. Right-click the **title** node, click **Rename** and rename it as **symbol**.

21. Update the following properties for the **symbol** node:

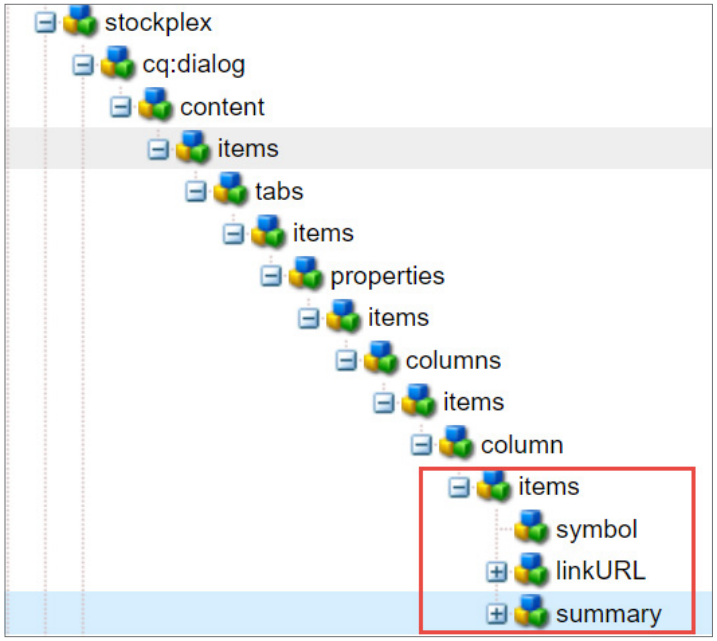
Name	Type	Value
name	String	./symbol
fieldLabel	String	Stock Symbol
fieldDescription	String	Enter a 4-character stock symbol



22. Right-click the **symbol** node and click **Copy**.

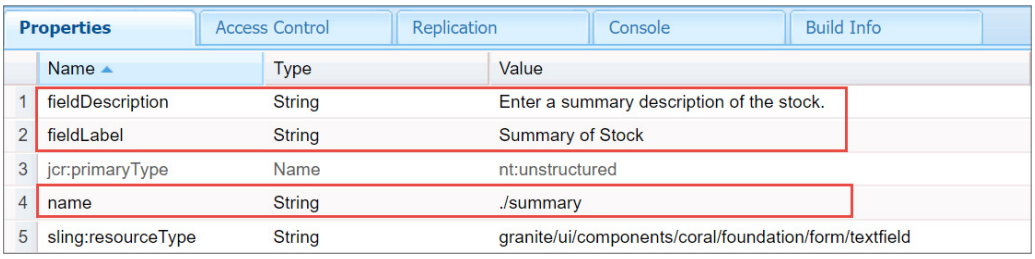
23. Right-click the **items** node that is above the **symbol** node and click **Paste**.

24. Rename the **Copy of symbol** node to **summary**. The summary and symbol nodes are siblings as shown:



25. 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



26. Right-click the **symbol** node and click **Copy** to copy it.

27. Right-click the **items** node that is above the **symbol** node and click **Paste** to paste the node.

28. Rename the **Copy of symbol** node to **stockdetails**.

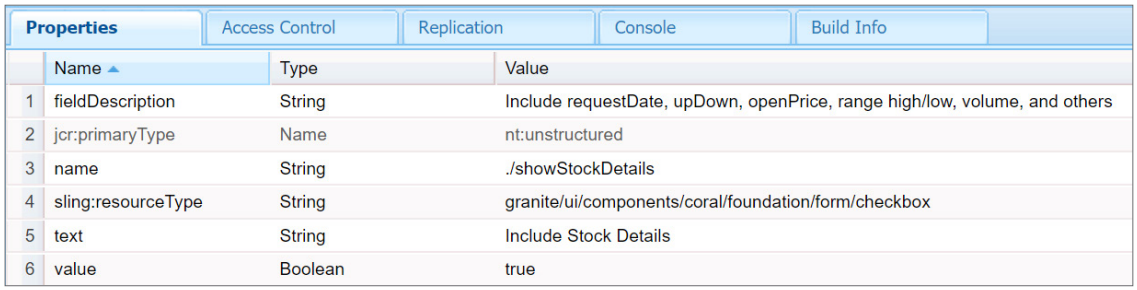
29. Update the following properties of the **stockdetails** node:

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

30. Add the following new properties to the **stockdetails** node:

Name	Type	Value
text	String	Include Stock Details
value	Boolean	true

31. Select the **fieldLabel** property, right-click it and click **Delete** from the list. The **stockdetails** node properties should look as shown:



Properties | Access Control | Replication | Console | Build Info

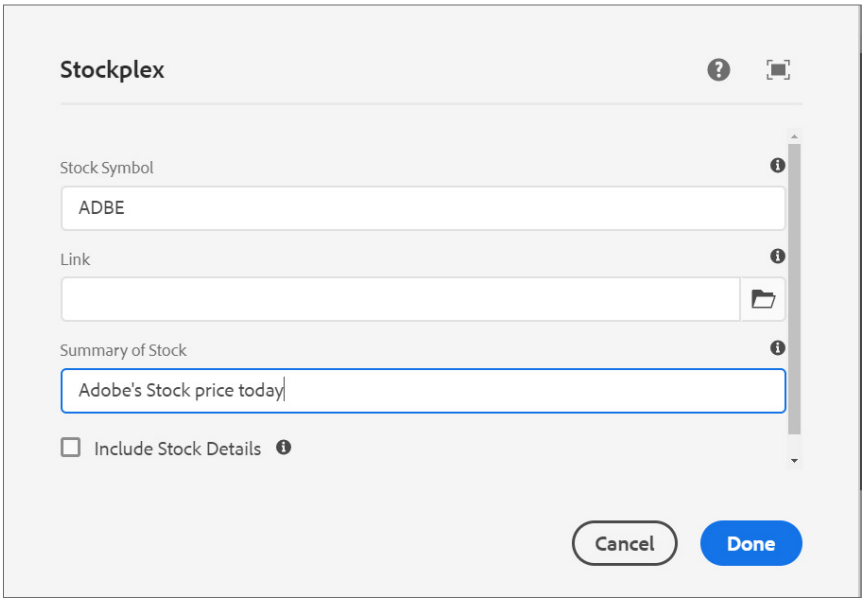
	Name	Type	Value
1	fieldDescription	String	Include requestDate, upDown, openPrice, range high/low, volume, and others
2	jcr:primaryType	Name	nt:unstructured
3	name	String	./showStockDetails
4	sling:resourceType	String	granite/ui/components/coral/foundation/form/checkbox
5	text	String	Include Stock Details
6	value	Boolean	true

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

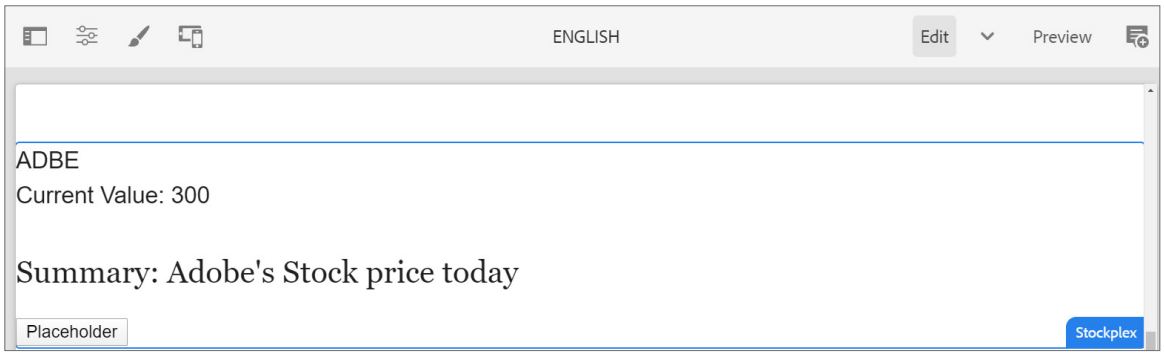
33. Navigate to the tab where the English page is open or click http://localhost:4502/editor.html/content/we-train/en.html

34. Select the **stockplex** component you added in a previous task and click the **Configure** (wrench) icon. The **Stockplex** dialog box opens.

35. Provide the following values, as shown:
    a. **Stock Symbol**: **ADBE**
    b. **Summary of Stock**: **Adobe's Stock price today**



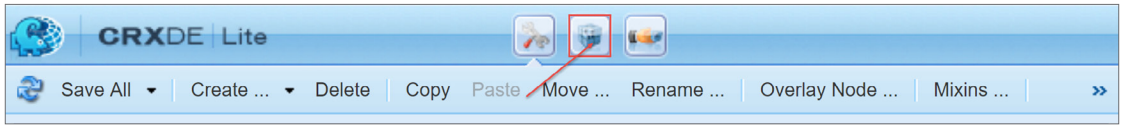
36. Click **Done**. You can view the content rendered on the stockplex component in the English page.



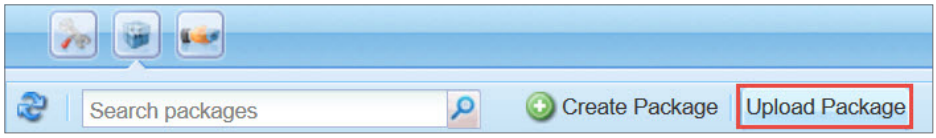
Task 2: Add symbol validation

Now that you have a basic dialog with three form fields, you need to create validation of the stock symbol. As the symbol description suggests, the user must enter a four-letter symbol. You can accomplish this by including client-side JavaScript with a client library. For convenience, you will upload the client library with package manager.

1. Navigate back to **CRXDE Lite** page or click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.
2. Click the **Package** icon from the header bar, as shown. The **CRX Package Manage**r page opens.

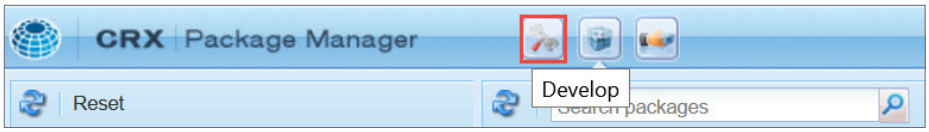


3. Click **Upload Package** from the actions bar, as shown. The **Upload Package** dialog box opens.



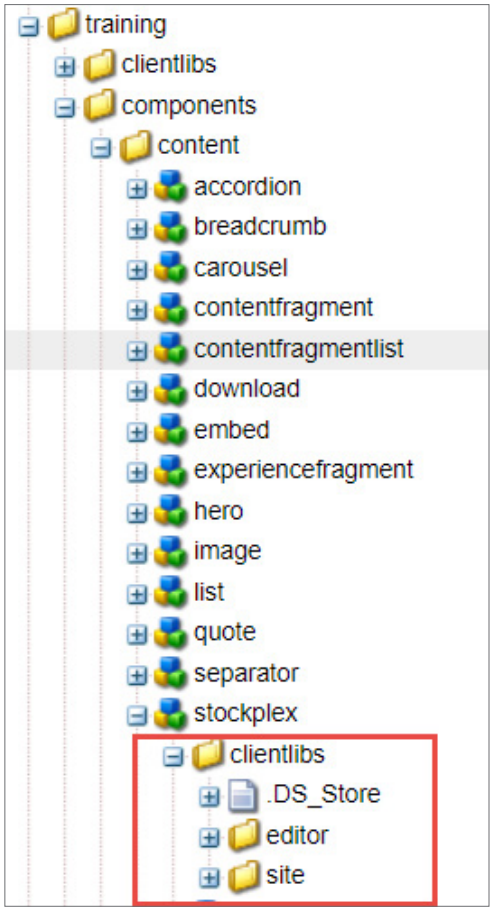
4. Click **Browse** in the dialog box. The **Open** dialog box opens.
5. In the Exercise Files folder provided to you, navigate to the **training-files/additional-component-features** folder and select the **stockplex-clientlibs-all.zip** package.
6. Click **OK** in the **Upload Package** dialog box. The package is uploaded.
7. Click **Install** from the actions bar in the **stockplex-clientlibs-all.zip** package section. The **Install Package** dialog box opens.
8. Click **Install**. The package is installed.

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



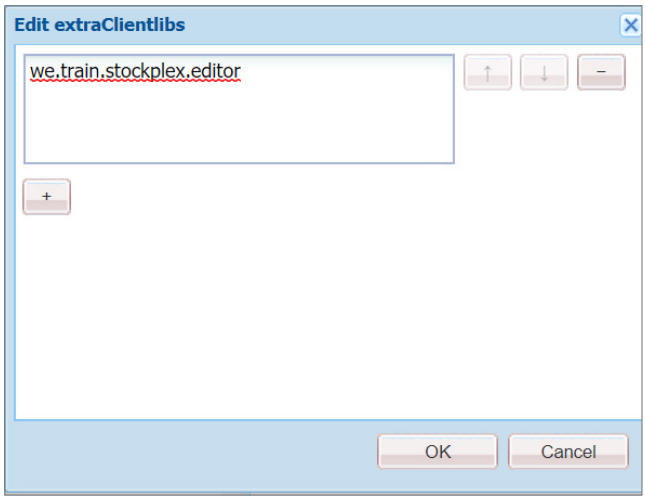
10. Navigate to the **/apps/training/components/content/stockplex/clientlibs** folder. Notice that the content package contains two client libraries for the stockplex component, as shown:

- editor: A client library containing a validation script for a dialog.
- site: A css/js file for the component design.



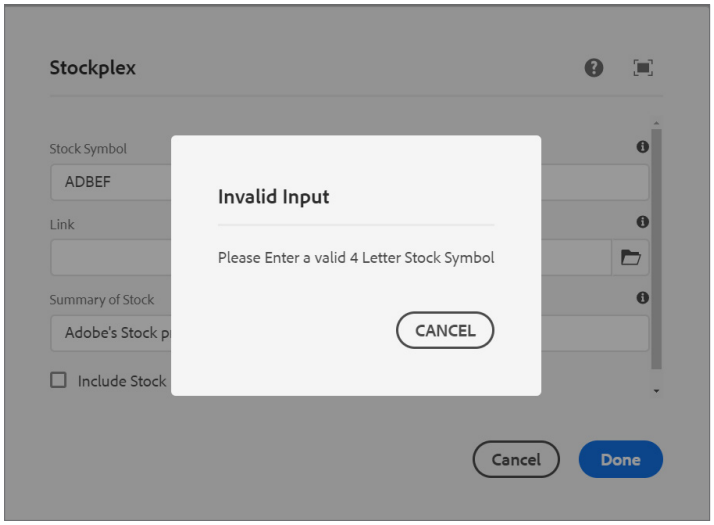
To use the stockplex editor client library, you must add it to the dialog box. To have your client library loaded solely for your component dialog, you must set the cq:dialog node property, extraClientLibs to the category name of the clientlib you created. This is recommended if your client library is quite large and/or your field is specific to that dialog and will not be needed in other dialogs.

11. Navigate to the **/apps/training/components/content/stockplex/cq:dialog** node.
12. In the **Properties** section, double-click the **extraClientlibs**. The **Edit extraClientlibs** dialog box opens.
13. Replace the existing value to **we.train.stockplex.editor**, as shown, and click **OK**. The existing value is replaced with the new value.



14. Click **Save All**.
15. Navigate to the tab where the **English** page is open.
16. Select the **stockplex** component and click the **Configure** (wrench) icon. The **Stockplex** dialog box opens.

17. In the **Stock Symbol** box, type more than four characters, for example, ADBEF, and click **Done**. The **Invalid Input** dialog box opens with a message, as shown. This indicates that you created an author dialog box for a custom component and added JavaScript in a client library to perform validation on one of the fields.



18. Click **CANCEL** to close the dialog box.
19. Click **Cancel** in the Stockplex dialog box to discard your changes.

Exercise 3: Add a base style to the custom component

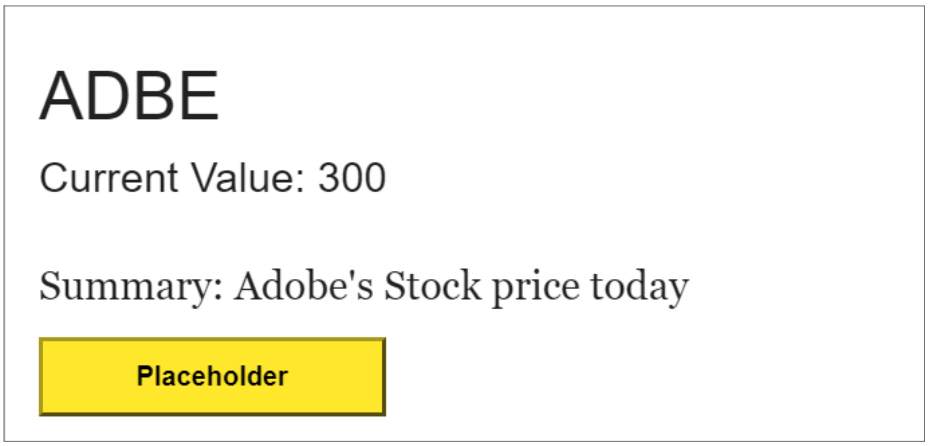
In the previous exercise, you installed the client libraries for the stockplex component. In this exercise, you will add a design to the stockplex component.

1. Ensure you are in **CRXDE Lite** or click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/components/content/stockplex/clientlibs/site** node. Notice that the clientlibs for the design of the stockplex component are available in this folder.

To use the design in stockplex component, you must add the design to the HTL.

3. Navigate to the **/apps/training/components/content** folder.
4. Expand the **stockplex** node and double-click the **stockplex.html** to open the script for editing.
5. Delete the existing code.
6. In the Exercise Files folder provided to you, navigate to **ui.apps/src/main/content/jcr_root/apps/training/components/content/stockplex**.
7. Open the clientlibs_stockplex.html file in Notepad ++ (or any editor of your choice), copy its content and replace the existing content in the stockplex.html editor in **CRXDE Lite** with the new content you copied.
8. Click **Save All**. Now that the HTL script is updated with class names, you can review the design you uploaded in the previous exercise.
9. In **CRXDE Lite**, navigate to **/apps/training/components/content/stockplex/clientlibs**. The clientlibs/site is reserved for clientlibs that are a part of the component design. This enables you to easily separate authoring clientlibs (clientlibs/editor) from design clientlibs.

10. Expand **clientlibs/site/stockplex.less**. Notice how this file is used to manage different styles for the stockplex component. If a new style is needed, you can create a new file under the styles folder and add it to stockplex.less.
11. Expand **clientlibs/site/styles/default.less**. As the name suggests, this is the default design of the stockplex component. You will see this design applied in the next step.
12. Navigate to the tab where the English page is open, and refresh the page. Notice that the stockplex component has a new design, as shown. You modified your component's script to use the CSS styles from the component's client library. After modifying, you can see the change in the appearance of the component.



Exercise 4: Add a style system to the custom 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 choose different designs they want to use on the page.

1. Ensure you are in **CRXDE Lite** or click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.

To view different designs that you want the authors to access:

2. Navigate to **clientlibs/site/styles** and notice the different .less files. Each .less file is a different style that can be optionally injected to the HTML.
3. Open the bold.less file. Notice how the class name cmp-stockplex--bold is never directly called in the stockplex.html file. Conversely, the classes that are used in the default.less file are hard coded into stockplex.html.
4. Open clientlibs/site/stockplex.less file. Notice how all .less files are being included—regardless of if they are directly used in the stockplex.html file or not. This allows for the styles to be loaded and then optionally injected to the HTML with the style system.

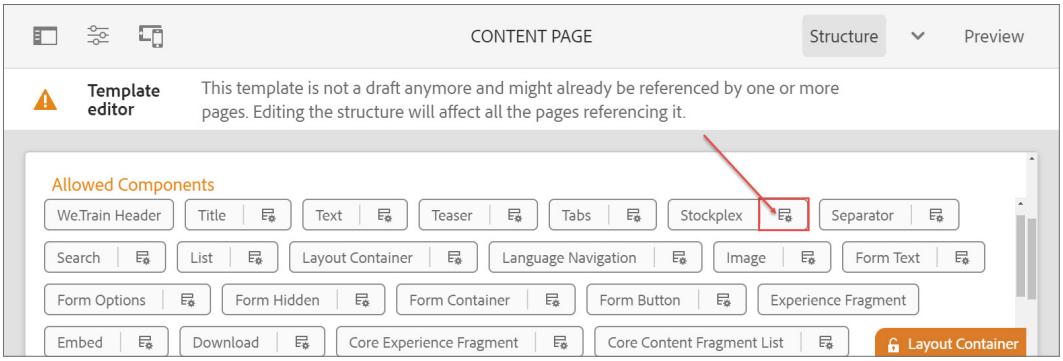
To help a template author define styles in a content policy, you must add the style system as a cq:design_dialog. Just like cq:dialogs, you will copy a cq:design_dialog from a core component and customize it.

5. Navigate to the **/apps/core/wcm/components/text/v2/text** node, right-click the **cq:design_dialog** node, and click **Copy** to copy the node.
6. Navigate to the **/apps/training/components/content**, right-click the **stockplex** node, and click **Paste** to paste the node. The node is added to the stockplex component.
7. Click **Save All**.
8. Select the **cq:design_dialog** node, and on the **Properties** tab, update the **jcr:title** property to **Stockplex**.
9. Click **Save All** from the actions bar.

10. Navigate to the **/apps/training/components/content/stockplex/cq:design_dialog/content/items/tabs/items** node, right-click the **plugins** node, and click **Delete**. The node is deleted.
11. Click **Save All**.

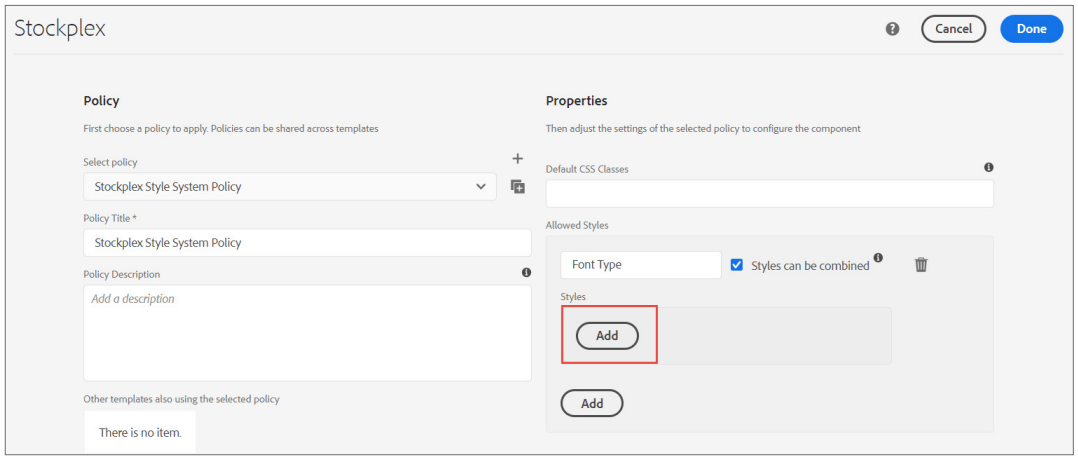
Now, as a template author, you will add the style system to the stockplex content policy.

12. Click localhost:4502/aem/start.html. The **Navigation** page opens.
13. Click the **Tools** (hammer) icon. The **Tools** console opens.
14. Ensure you are in the **General** section and click the **Templates** tile. The **Templates** console opens.
15. Click the **We.Train** folder, select the **Content Page** template, and click **Edit (*e*)** from the actions bar. The **Content Page** template editor opens on a new tab.
16. In the **Layout Container** component, click the **Policy** icon next to the **Stockplex** component, as shown. The **Stockplex policy** wizard opens.



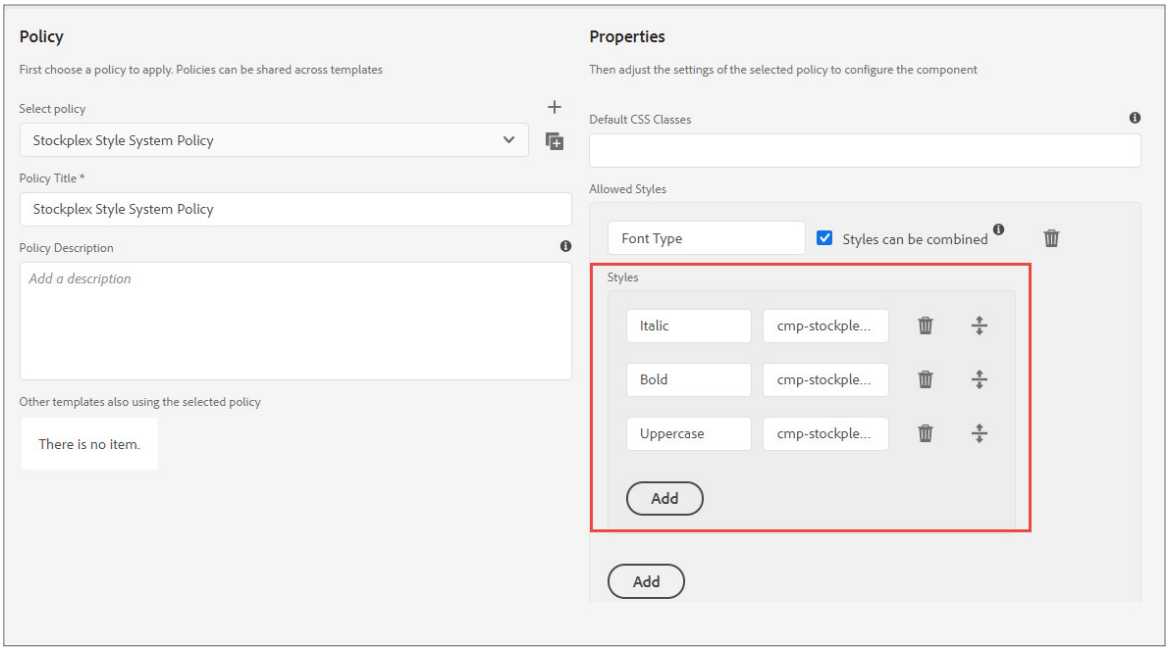
17. On the **Policy** section, type **Stockplex Style System Policy** in the **Policy Title** box.
18. On the **Properties** section, under the **Styles** tab, click the **Add** button below the **Allowed Styles** field.
19. In the **Group Name** box, type **Font Types**, and select the **Styles can be combined** check box.

20. Click the **Add** button that is below the **Styles** field, as shown. A new style field is created.



21. Enter the **Style Name** as **Italic** and **cmp-stockplex--italic** as the **CSS Classes**.
22. Click the **Add** button below the **Italic** style you just added. A new style field is created.
23. Enter the **Style Name** as **Bold** and **cmp-stockplex--bold** as the **CSS Classes**.
24. Click the **Add** button again, and enter the **Style Name** as **Uppercase** and **cmp-stockplex--uppercase** as the **CSS Classes**.

Your styles should look similar to the one shown in the following screenshot:

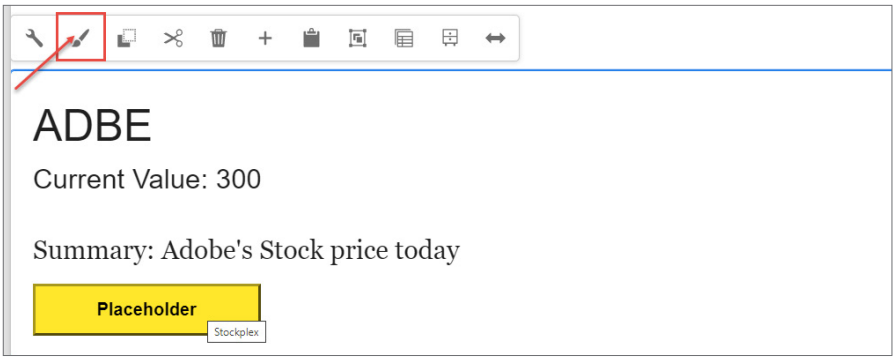


25. Click **Done**. The **Content Page** template editor opens.

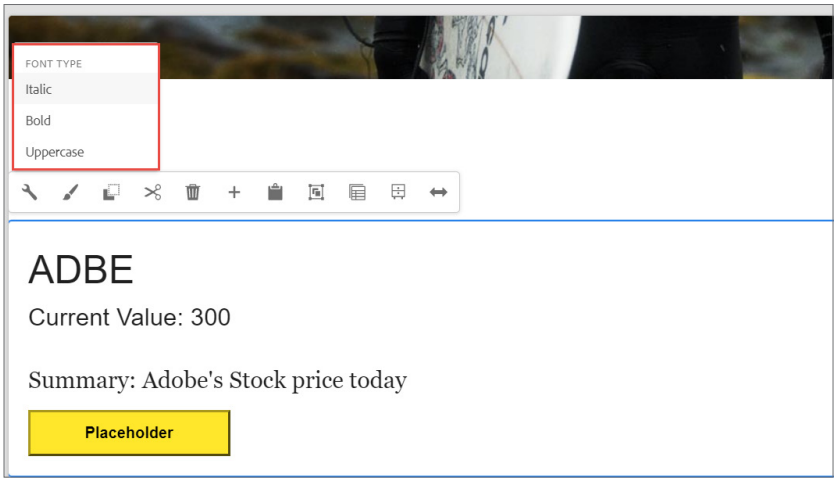
Now that you have the CSS developed by the designer, and the style system is added by the template editor, authors can use the style system.

26. Navigate to the **English** page of the We.Train site, and refresh it.

27. Select the **Stockplex** component. You should now see the **Styles** icon (paintbrush icon) in the component toolbar, as shown:



28. Click the **Styles** icon from the actions bar and click **Italic**, **Bold**, or **Uppercase**, as shown, and observe the changes. Notice that you can select more than one style at once. This is because, you selected the **Styles can be combined** option when defining the policy.
You have now modified your component's script to use the CSS styles from the component's client library. After modifying, you noticed changes in the appearance of the component. If you are unable to view any changes to the styles in the stockplex component, refresh the English page.

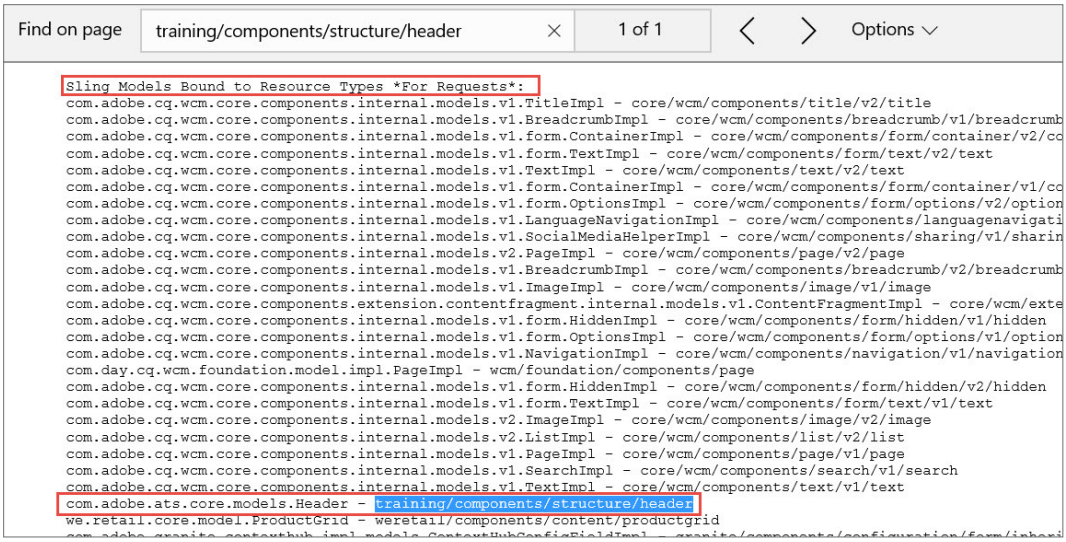


Exercise 5: Add a Sling model as the business logic

In this exercise, you will use a Sling model as the business logic of the stockplex component. This will allow the component to be used for both HTML rendering and JSON output.

Note: You have already installed the Sling Model you will use in this exercise. The Stockplex.java class was installed with the we-train-bundle.zip content package. If you would like to review the Stockplex.java model code installed, you can find the file in your Exercise Files folder under core/src/main/java/com/adobe/training/core/models/Stockplex.java.

1. To verify if the Stockplex.java Sling Model was installed successfully, click http://localhost:4502/system/console/bundles. The **Adobe Experience Manager Web Console Bundles** page opens.
2. Click **Status** on the header bar and select **Sling Models** from the drop-down menu. The **Adobe Experience Manager Web Console Sling Models** page opens.
3. Press Ctrl+F (Windows) or Command+F (Mac) on your keyboard and type **training/components/structure/header** in the search field. 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, press Ctrl+F (Windows) or Command+F (Mac). This confirms the bundle is installed successfully.



To use the Stockplex model, you must update the stockplex.html script:

4. Ensure you are in **CRXDE Lite** or click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.
5. Navigate to the **/apps/training/components/content** node.
6. Select the **stockplex** node and double-click **stockplex.html** to open the script for editing.
7. Delete the existing content.
8. In the Exercise Files folder provided to you, navigate to **ui.apps/src/main/content/jcr_root/apps/training/components/content/stockplex**.
9. Open the **slingmodel_stockplex.html** file in Notepad ++ (or any editor of your choice), copy its content and paste it in the **stockplex.html** editor in **CRXDE Lite**.
10. Click **Save All** from the actions bar. Review the code you added. Notice `data-sly-use= "com.adobe.training.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.

To observe the JSON output:

11. Navigate to the **English** page or click http://localhost:4502/editor.html/content/we-train/en.html. The **English** page of We.Train site opens.
12. 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:

To view the output:

13. 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 on the same tab, as shown. You modified your component to receive data from a Java Sling Model and exposed the Sling Model's JSON Export capability in your component's interface.

14. Additionally, you can view the stockplex JSON as a part of the entire page's JSON output. Open the English page in edit mode, and click **Page Information** > **View as Published**. From the URL, remove **.html?wcmmode=disabled** and add **.model.json**. This is the JSON output of the entire page including the stockplex sling model.

(Optional) Exercise: Include a data source to the Stockplex component

In this exercise, you will add a data source to the Stockplex component. The wetrain-core-1.0-SNAPSHOT.jar has several java classes that listen for new stock symbols created (StockListener.java), create new stock schedulers (StockImportScheduler.java), and then write new stock to the JCR (StockDataWriterJob.java). You can review these three java classes in detail in the Exercise Files folder under core/src/main/java/com/adobe/training/core. These java classes are discussed in detail in the Extend and Customize AEM course. In this exercise, You will observe their functions. As the bundle is already installed, you need to configure it to start importing stock data.

1. Install **stockplex-backend.zip** package from the Exercise files folder (training-files/additional-component-features folder) provided to you. The package creates:

- A service user with permissions (training-user)
- An OSGi config for a Service User Mapper
- The initial /content/stocks folder

2. Click localhost:4502/sites.html. You will see **stocks** > **ADBE** folders. If you do not see the ADBE folder, create it under the Stock folder by clicking **Create** > **Folder**. This triggers the listener that creates an OSGi config for the scheduler to run. You can find this config in http://localhost:4502/system/console/configMgr under **Training Stock Importer**.

Note: The Stock you are importing is dummy data within the github project. You can also try other symbols such as AMZN, MSFT, APPL, GOOG, and WDAY. The scheduler creates a new Sling job every two minutes. The Sling job then reaches out to the API endpoint and writes the stock data into the JCR. To achieve this, the StockDataWriter uses a service user (line 187). To map the service user, a configuration called ServiceUserMapper is required. This configuration maps service user to your project and is located under /apps/training/config.author.

3. The training-user account is a special service user account that can be created from the http://localhost:4502/crx/explorer console. You can also manage permissions.
4. After two minutes, refresh http://localhost:4502/crx/de/index.jsp#/content/stocks/ADBE and you should now see a **trade** node that contains the imported data.
5. To use the back-end data, open the **We.Train** > **English** page.
6. Add the **Stockplex** component to the page.

7. Click the **Configure** icon (wrench icon), enter **ADBE** in the **Stock Symbol** box, select the **Include Stock Details** check box, and click **Done**. You should see the real stock data for ADBE.

Exercise 6: 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.

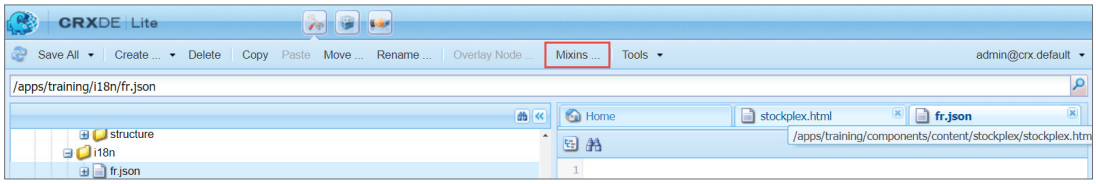
1. In **CRXDE Lite**, navigate to the **/apps/training/components/content/stockplex** node.
2. In the Exercise Files folder provided to you, navigate to ui.apps/src/main/content/jcr_root/apps/training/components/content/header/stockplex.
3. Open the **i18n_stockplex.html** file in Notepad ++ (or any editor of your choice), copy its content, and paste it in the **stockplex.html** editor in **CRXDE Lite**. In the code snippet, observe the i18n content. You can easily identify it based on @ i18n

```
<sly data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html" data-sly-call="${clientlib.css @ categories='we.train.stockplex'}" />

<div class="cmp-stockplex"
    data-sly-use.template="core/wcm/components/commons/v1/templates.html"
    data-sly-test.symbol="${properties.symbol}"
    data-sly-use.stockplex="com.adobe.training.core.models.Stockplex">

    <div class="cmp-stockplex__column1">
        <div class="cmp-stockplex__symbol">${stockplex.symbol}</div>
        <div class="cmp-stockplex__currentPrice">${'Current Value:' @ i18n} ${stockplex.currentPrice}</div>


        <div class="cmp-stockplex__summary" data-sly-test.summary="${stockplex.summary}">
            <h3>${'Summary:' @ i18n} ${stockplex.summary}</h3>
        </div>

        <div class="cmp-stockplex__button">
            <a href="${resource.path @ selectors='model', extension='json'}">
                <button>${'See this stock via JSON' @ i18n}</button>
            </a>
        </div>
    </div>
    <div class="cmp-stockplex__column2">
        <div class="cmp-stockplex__details" data-sly-test="${stockplex.showStockDetails}">
            <ul data-sly-list.dataKey="${stockplex.data}">
                <li class="cmp-stockplex__details-item">
                    <span class="cmp-stockplex__details-title">${dataKey @ i18n}:</span>
                    <br />
                    <span class="cmp-stockplex__details-data">${stockplex.data[dataKey]}</span>
                </li>
            </ul>
        </div>
    </div>
```

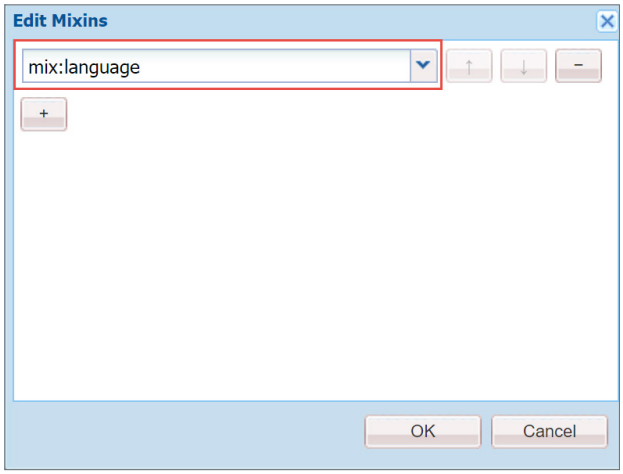
To create internationalized keys for the Stockplex component:

4. Navigate to the **/apps/training** folder, and create a new folder by clicking **Create** > **Create Folder**. The **Create Folder** dialog box opens.
5. Enter **i18n** in the **Name box**, and click **OK**. The folder is created.
6. Click **Save All** from the actions bar.
7. Right-click the **i18n** folder, and click **Create** > **Create File**. A new file is created.

8. Name it **fr.json** and click **OK**.
9. Click **Save All** from the actions bar.
10. Select the **fr.json** file, and click **Mixins** from the actions bar, as shown. The **Edit Mixins** dialog box opens.



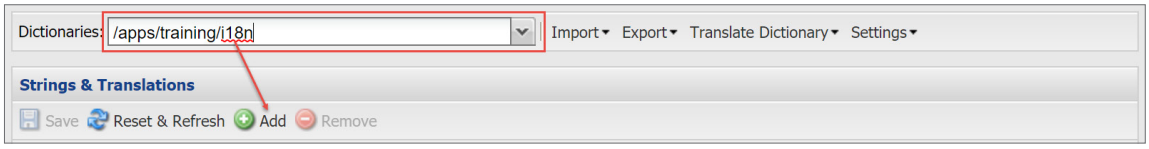
11. Click the + icon in the **Edit Mixins** dialog box.
12. Select **mix:language** from the drop-down menu, as shown:



13. Click **OK**.
14. Click **Save All** from the actions bar.
15. Select the **fr.json** node and add the following property:

Name	Type	Value
jcr:language	String	fr

16. Click **Save All**.
17. Click http://localhost:4502/libs/cq/i18n/translator.html. The **Strings & Translations** page opens.
18. Select **/apps/training/i18n** from the **Dictionaries** drop-down menu, and click **Add** from the actions bar, as shown. The **Add string** dialog box opens.



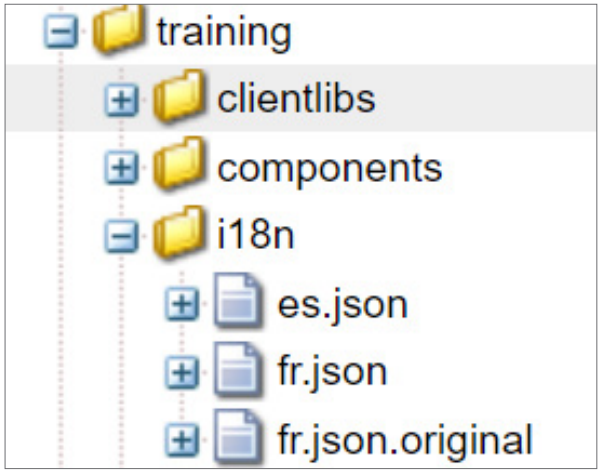
19. Enter the following information in the appropriate fields:

Field Name	Value
String	Summary
ES	Sommaire
FR	Resume

20. Click **OK**.

21. Click **Save** from the actions bar.

22. Refresh the **CRXDE Lite** page and navigate to the **/apps/training/i18n** folder. Notice that the translator has created two new file nodes, es.json and fr.json. The original fr.json is retained and renamed as fr.json.original. It can be deleted.



23. Examine the contents and properties of the es.json and fr.json files. Both are the nodes of type nt:file and carry the mixin type of mix:language. Each file also has a jcr:language property. The files contain the localization string data.
    So far, you have created an i18 node structure in the component and added elements to your component to localize it. To test the localized content:

24. Navigate to the **We.Train** > **French** page, or click http://localhost:4502/editor.html/content/we-train/fr.html. The **French** page of the We.Train site opens in edit mode.

25. Drag the **Stockplex** component from the left panel onto the **Drag components here** area of the page.

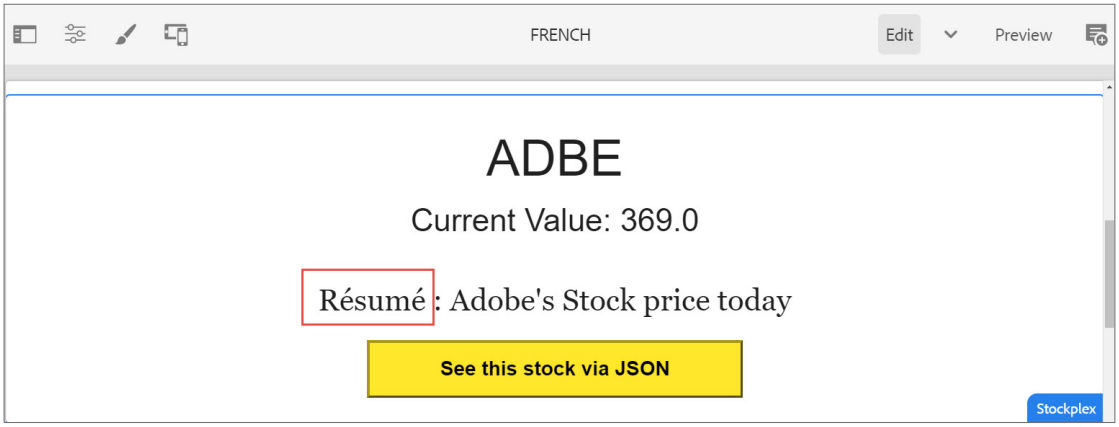
26. Select the Stockplex component and click the Configure icon (wrench icon). The **Stockplex** dialog box opens.

27. In the **Stock Symbol** box, enter **ADBE**.

28. In the **Summary of stock** box, enter **Adobe's Stock price today**.

29. Click **Done**. The stock details are updated on the page.

Notice that the word Summary is translated to the French word Resume.



(Optional) Exercise: Create additional supported languages

You now know how to quickly create and update i18n json files with the translation.html tool. As a developer, you might end up getting a json file with key value pairs from a translation team and need to only update them in the JCR. In this optional exercise:

1. Create another i18n json file called **it.json**.
2. In the Exercise Files folder provided to you, navigate to the ui.apps/src/main/content/jcr_root/apps/training/i18n folder and replace the json files in the JCR with the files in this folder.
3. Create an Italian (it) language branch and a Spanish (es) language branch and see if the content is translated.

Preparing for Production

Introduction

Before implementing Adobe Experience Manager (AEM) as the Content Management System (CMS), you must prepare AEM for production and optimize the performance of AEM during deployment.

Objectives

After completing this module, you will be able to:

- Customize the AEM UI
- Explain how to prepare AEM for production
- Complete the empty page template type
- Create production templates
- Create content package
- Add your work to a Maven project

Customize the AEM UI

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.

Overlays

Sling's Resource Resolver searches 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 the default functionality is called an overlay.

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. The Resource Resolver retrieves the resource from /apps, failing which, it retrieves from /libs.

An overlay involves taking the predefined functionality and imposing your own definitions over that to override the standard functionality.

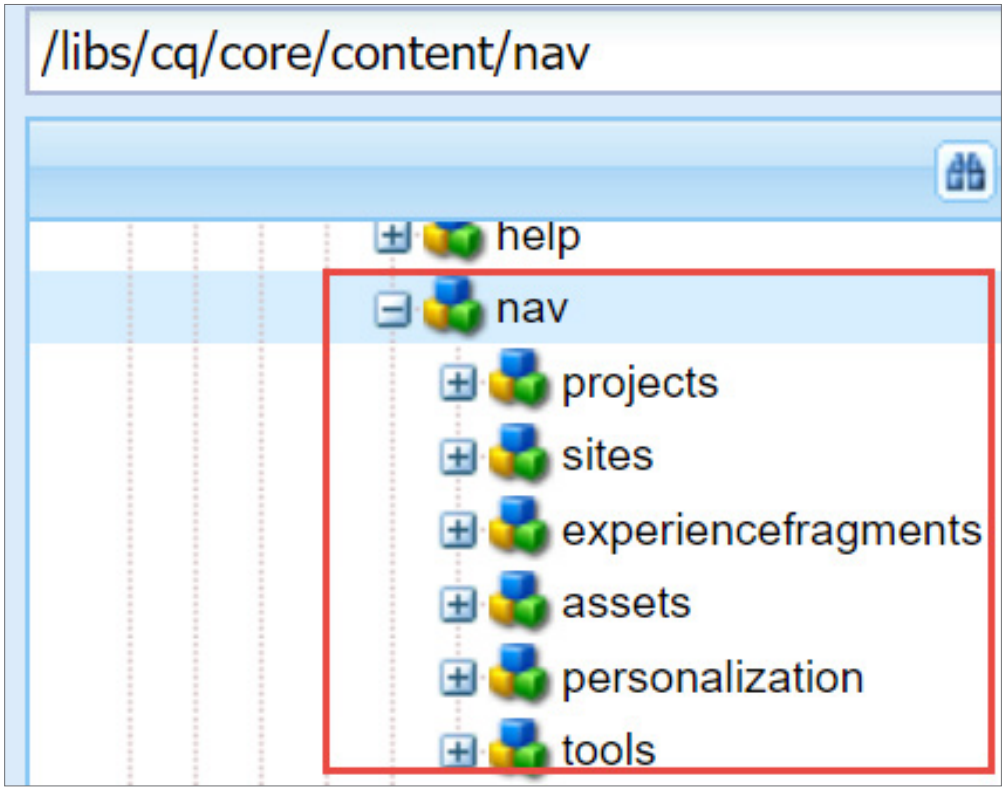
Exercise 1: Extend the navigation UI

In this exercise, you will modify the navigation UI buttons by using the Sling Resource Merger.

1. Click http://localhost:4502/aem/start.html. The **Navigation** page opens. Notice the **Sites** navigation button.

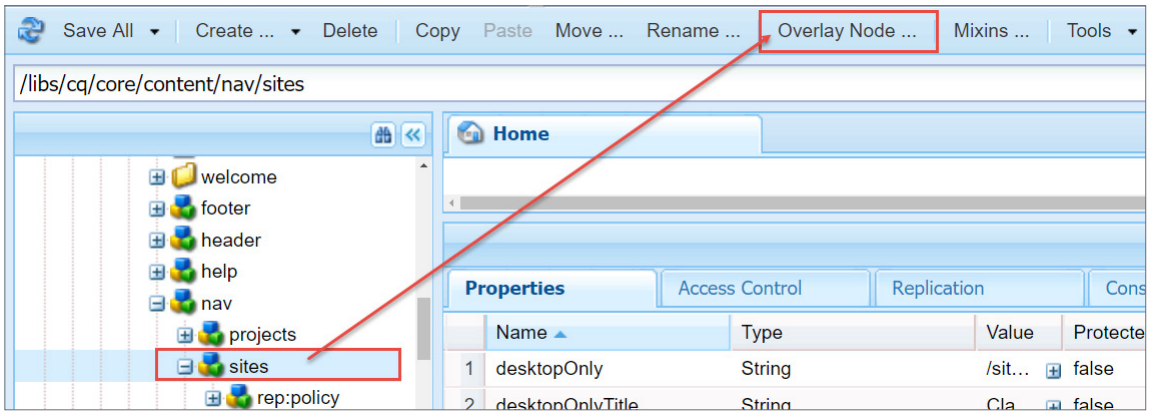
To modify the **Sites** navigation button:

2. In **CRXDE Lite** 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.

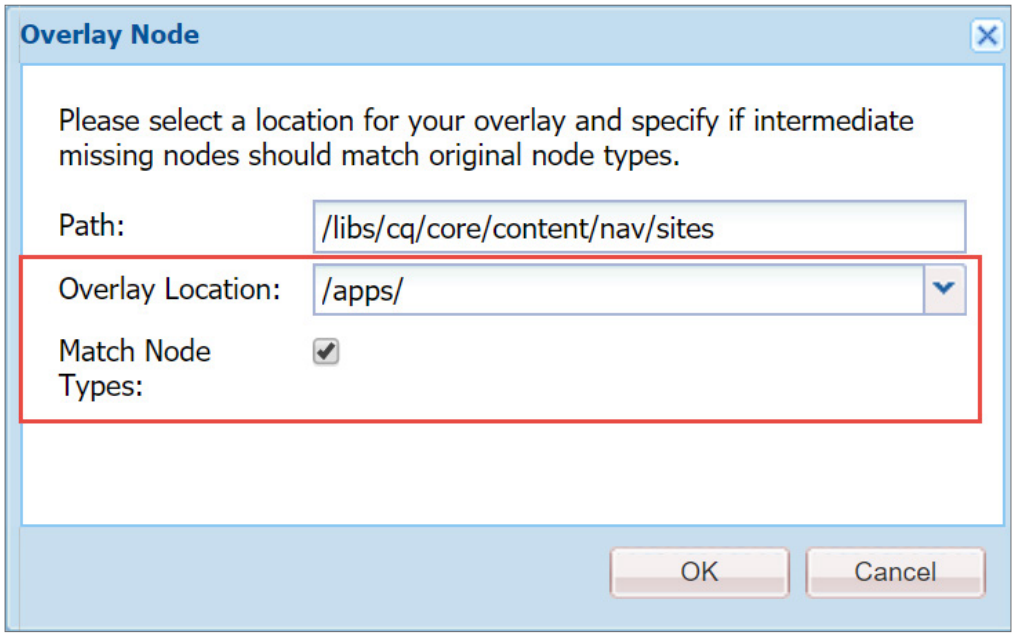


To modify the navigation icons, hide icons, or add a new icon, you must change the list under **/libs/cq/core/content/nav**. It is a best practice to recreate the structure in **/apps** by using the Sling Resource Merger to avoid any changes to the **/libs** structure.

3. 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.

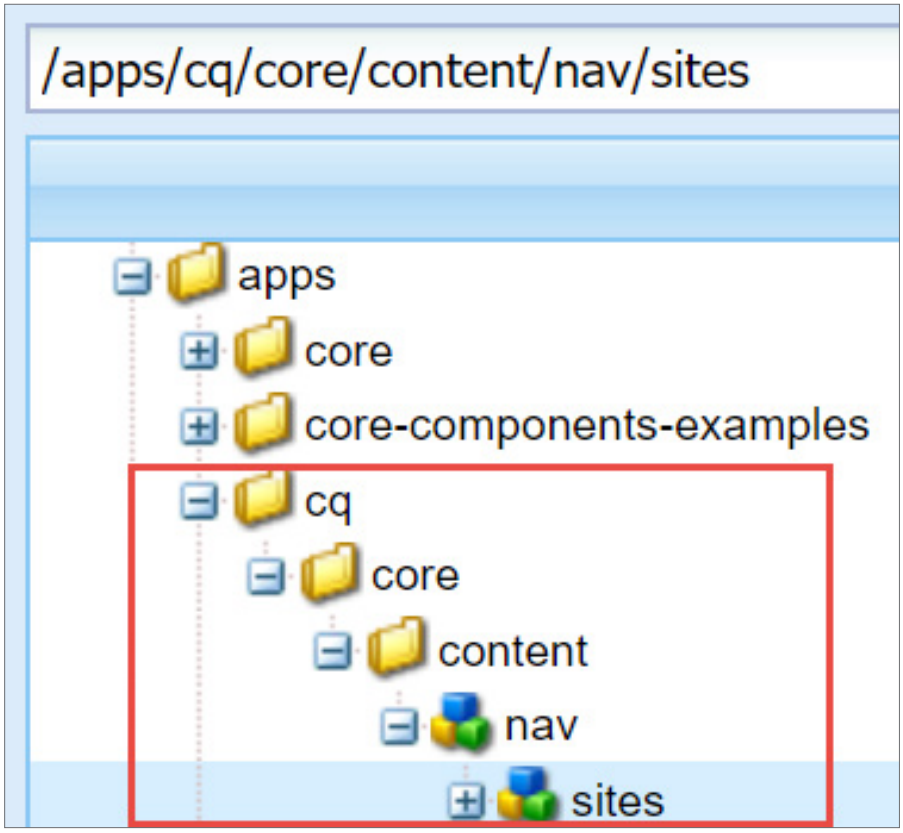


4. Ensure the **Overlay Location** field has **/apps/** as its value, select the **Match Node Types** check box, as shown, and click **OK**.



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

6. Navigate to the **/apps** folder and notice that the **/cq/core/content/nav/sites** structure is created. If you cannot view the **/apps/cq** folder structure, refresh the **CRXDE Lite** page.



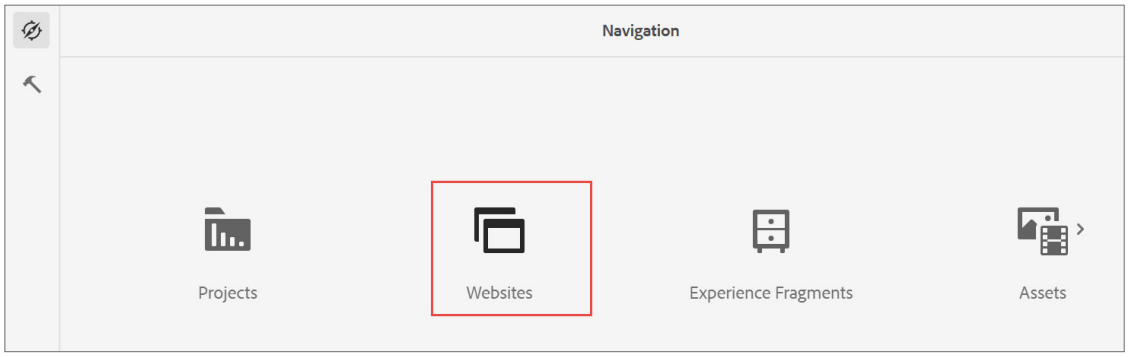
Note: The overlay node structure is reproduced, but the properties are not created. Remember, the Sling Resource Merger will merge properties and nodes.

7. Add the following property to sites under the **/apps/cq/core/content/nav/sites** node:

Name	Type	Value
jcr:title	String	Websites

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

9. Open the tab where the **Navigation** page is open. Refresh the page and notice that the **Sites** button is now renamed to **Websites,** as shown. You used the Sling Resource merger to modify the AEM application navigation.



AEM in Production

A developer must move their code into production or Quality Assurance (QA) after the development is completed. The process involves finishing the template-type, creating any initial editable templates for production, finalizing/creating the skeleton website structure, packaging the code, and moving the code through environments to production. You must move the changes from the template to the template type if you do not want the template authors to recreate all configurations and sync allowed components, clientlibs, and thumbnails that you have set up.

Developers create the first few templates of the project. This helps create the initial site structure, documentation on company customizations, and sample templates for template authors.

After the initial templates are moved to the production, template authors will maintain and create new templates based on the business need. The developers set the standards, and template editors adhere to those standards.

Typically, you would develop codes 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 2: Complete the empty page template type

After testing all structure components (assuming they have gone through a full QA process), you can complete the template-type.

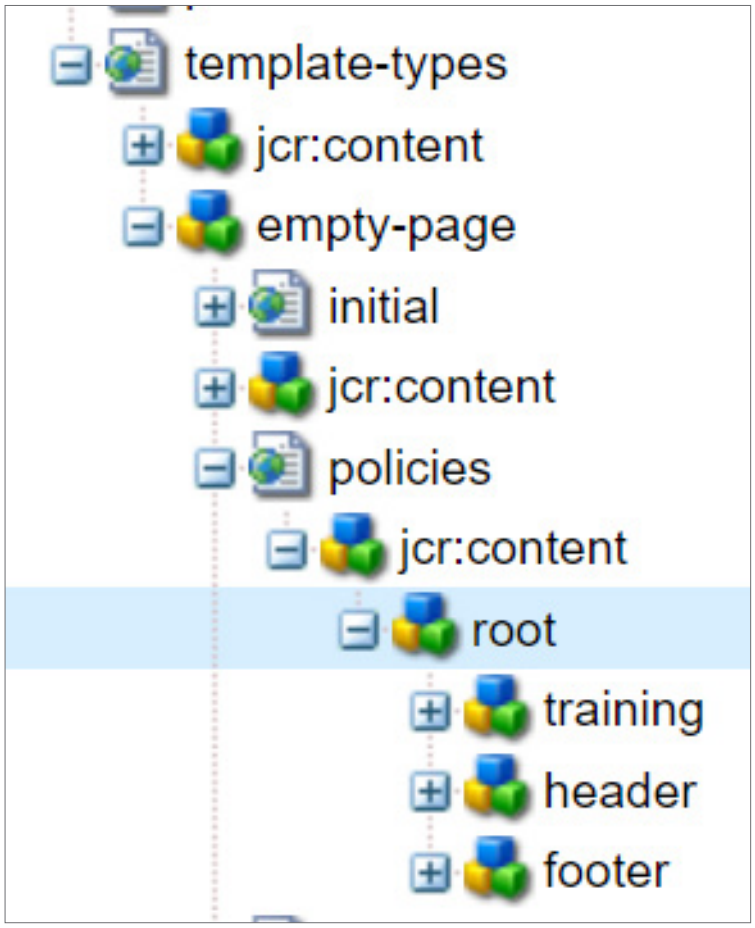
In this exercise, you will enable template authors to use the content components in the template along with the structure components.

1. In your AEM author service, click the **Tools** icon.
2. Click **General** > **Templates**. The **Templates** console opens.
3. Click the **We.Train** folder. The **We.Train** folder opens
4. Select the **Content Page** template and click **Edit** (***e***) from the actions bar. The **Content Page** template editor opens on a new tab.
5. Select the **Layout Container [Root]** and click the **Policy** icon from the component toolbar. The **Layout Container** wizard opens.
6. In the **Policy** section, notice the structure components you added previously to the policy.
7. In the **Properties** section, under **Allowed Components**, notice that the **WE.TRAIN** component group is selected.

You must copy the updates that you have made to the template back to the template-type.

8. If not already open, open **CRXDE Lite** (http://localhost:4502/crx/de).
9. Navigate to the **/conf/we-train/settings/wcm/templates/content-page/policies/jcr:content** node.

10. Select the **root** node under it 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.
11. Navigate to the **/conf/we-train/settings/wcm/template-types/empty-page/policies/jcr:content** node and click **Paste** from the actions bar.
12. Click **Save All** from the actions bar.
13. Navigate to the **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. If you need to auto-add any other configurations or components to a new template, you can copy those nodes and properties to the template-type. The node structure should look as shown:



Exercise 3: Create production templates

You have completed the development of a template-type and a 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 templates created by the development team.

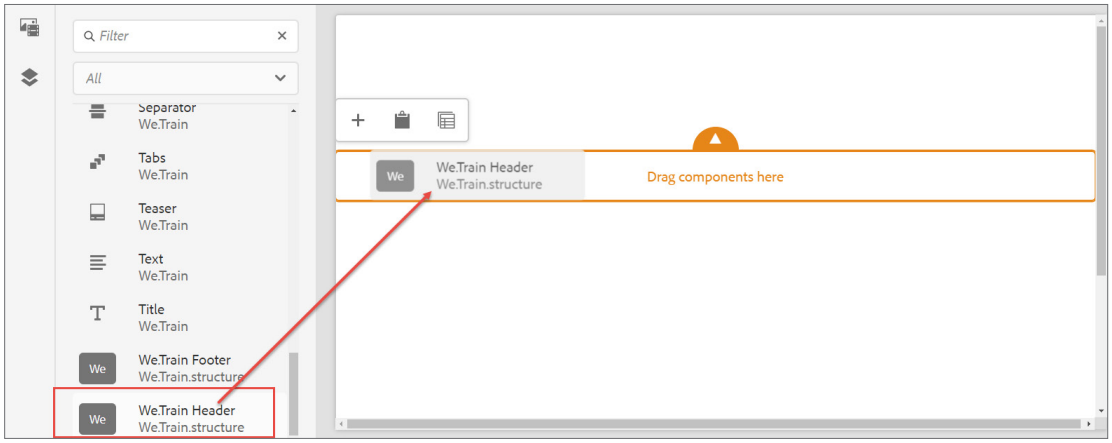
You will create the following two new templates:

- Sidebar Page template
- Stockplex template

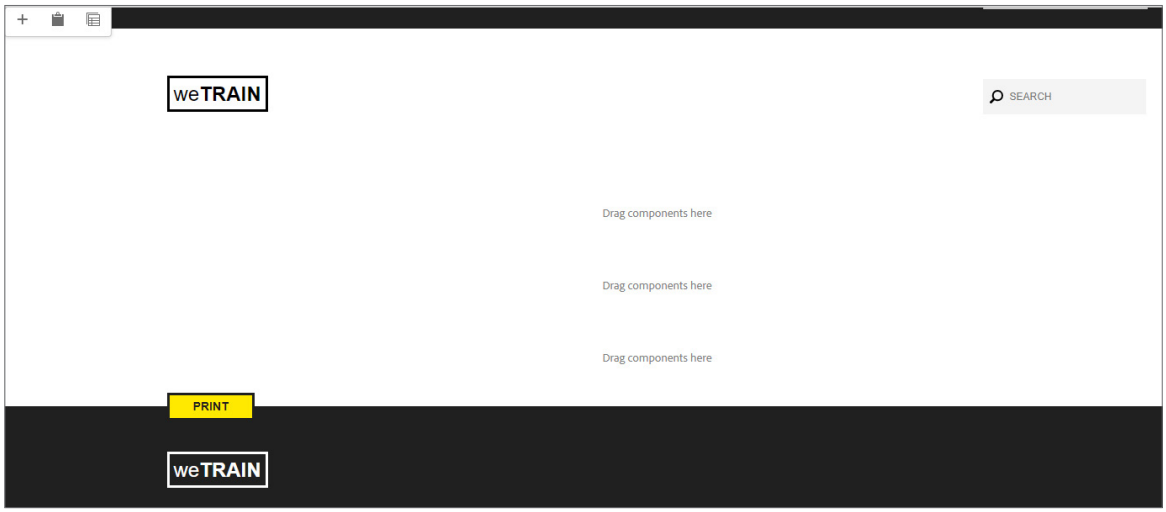
To create a Sidebar Page template:

1. Navigate to **Adobe Experience Manager** > **Tools** > **Templates**, if you are not already in it. The **Templates** console opens.
2. Click the **We.Train** folder. The **We.Train** folder opens.
3. Click **Create** from the actions bar. The **Create Template** wizard opens.
4. Select the **We.Train Empty Page** template and click **Next**. The **Template Details** wizard opens.
5. In the **Template Title** box, type **Sidebar Page Template** and click **Create**. 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 panel** opens. Ensure the **All** option is selected on the drop-down menu in the **Components** panel.

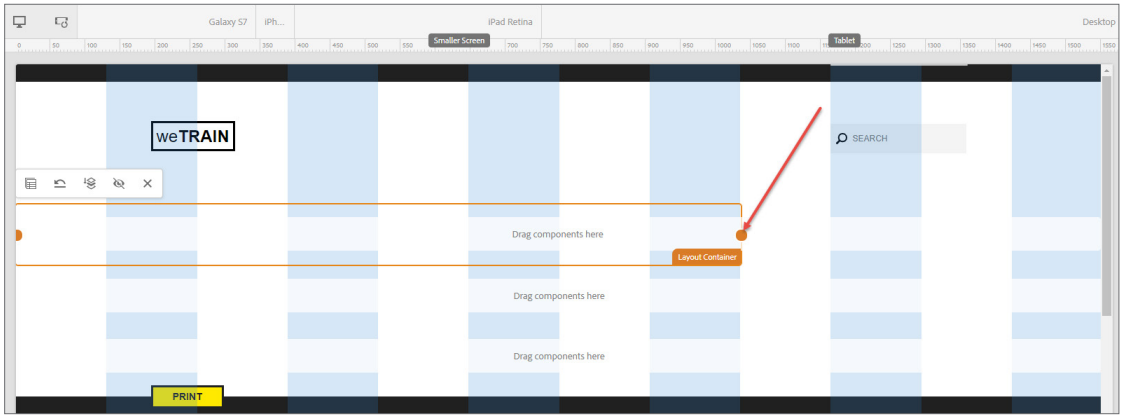
10. Drag the **We.Train Header** component from the **Components** panel to the **Drag components here** area placeholder, as shown. The component is added.



11. Drag the **Layout Container** component from the **Components** panel below the header component. The component is added.
12. Similarly, add two more **Layout Container** components.
13. Drag the **We.Train Footer** component from the **Components** panel to the **Drag components here** area placeholder. The component is added. The page should look as shown:

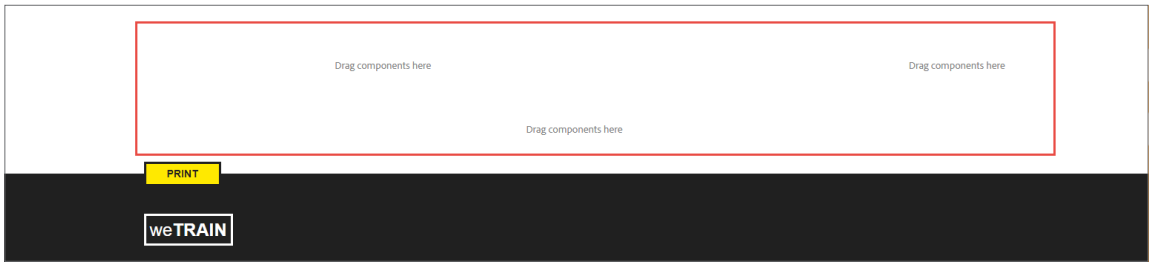


14. Select the first **Layout Container**, click the **Layout** icon (double-headed arrow) from the component toolbar, and drag the right orange handler up to 2/3rd of the grid system, as shown. The **Layout Container** resizes accordingly.



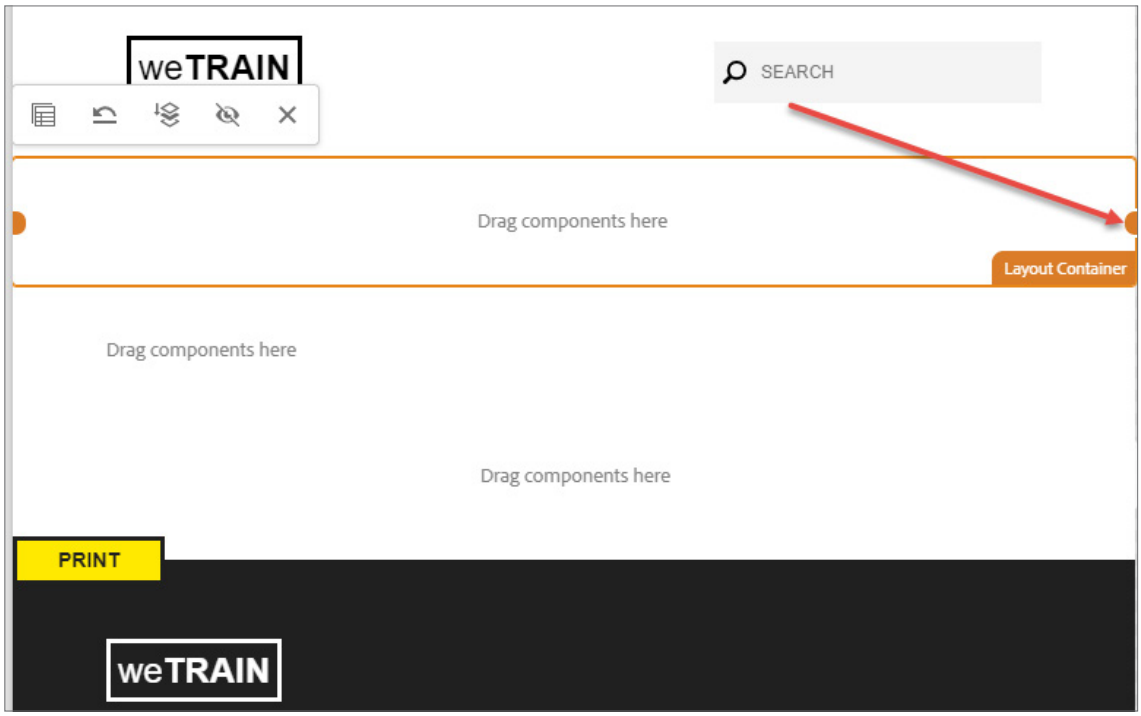
15. Select the middle **Layout Container**, click the **Layout** icon, and drag the right orange handler up to 1/3rd of the grid system. The **Layout Container** resizes accordingly.

16. Retain the last **Layout Containe**r as it is (12/12). You will have a 2/3, 1/3, and 12/12 layout, as shown:



17. Click the **Emulator** icon from the toolbar, and click the **iPad Retina** device from the group. The template rearranges itself to the device size.

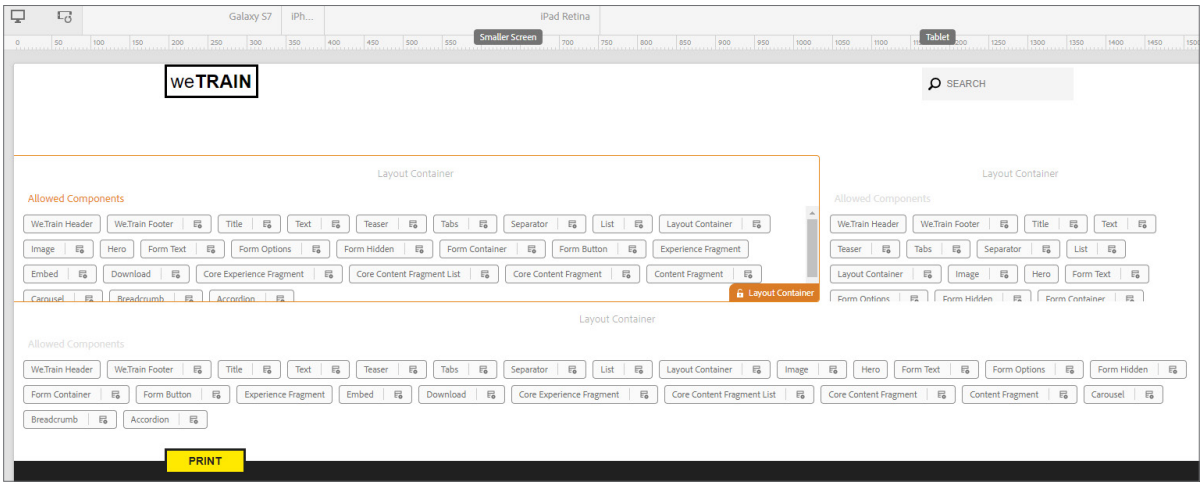
18. Select the first **Layout Container**, click the **Layout** icon and span the container to take up the entire grid system, as shown:



19. Similarly, adjust the second **Layout Container** to take up the entire grid system. You will have a 12/12, 12/12, and 12/12 layout.
20. Click the **Emulator** icon from the toolbar, and click the **Galaxy S7** device. The template rearranges itself to the device size.
21. Select the top **Layout Container**, click the **Layout** icon and adjust the container to take up the entire grid system.
22. Click the **Emulator** icon from the toolbar, and click the **Desktop** device from the group. The template rearranges itself to the device size.

23. For each **Layout Container** you added, perform the following actions:
    a. Select the **Layout Container** component and click the **Unlock structure component** (lock) icon from the component toolbar. The component is unlocked and reads as **No allowed components for Layout Container**.
    b. Select the **Layout Container** component and click the **Policy** icon from the component toolbar. The Layout Container wizard opens.
        i. In the **Policy** section, from the **Select policy** drop-down menu, select the **We.Train Content Policy**.
        ii. Click **Done**. You will be taken back to the template.

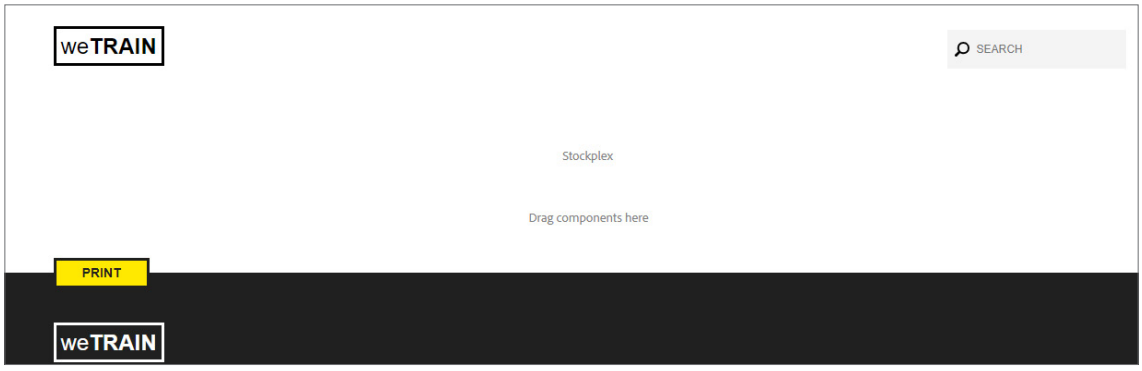
Your template should look as shown:



24. Navigate to the tab where the **Templates** console is open.
25. Select the **Sidebar Page Template** and click **Enable** from the actions bar. The **Enable** dialog box opens.
26. Click **Enable**. The template is enabled.
27. With the **Sidebar Page Template** already selected, click **Publish** from the actions bar. The **Publish** dialog box opens.
28. Ensure all check boxes are selected and click **Publish**. The **INFO The page has been published** success message appears.

Similarly, create a Stockplex template:

29. Navigate to **Templates** > **We.Train** folder.
30. Click **Create** from the actions bar. The **Create Template** wizard opens.
31. Select the **We.Train Empty Page** template and click **Next**. The **Template Details** wizard opens.
32. Enter **Stockplex Template** in the **Template Title** box and click **Create**. The **Success** dialog box opens.
33. Click **Open**. The **Stockplex Template** opens on a new tab.
34. Ensure the template is open in the **Structure** mode.
35. Click the **Toggle Side Panel** icon from the toolbar. The panel opens.
36. Click the **Components** icon. The **Components panel** opens.
37. Ensure the **All** option is selected on the drop-down menu in the **Components** panel.
38. Drag the **We.Train Header** component from the **Components** panel to the **Drag components here** area placeholder. The **Header** component is added.
39. Drag the **Stockplex** component from the **Components** panel and drop it below the **Header** component. The **Stockplex** component is added.
40. Drag the **Layout Container** component from the **Components** panel and drop it below the **Stockplex** component. The **Layout Container** component is added.
41. Drag the **We.Train Footer** component from the **Components** panel and drop it below the **Layout Container** component. The footer component is added. Your page should look as shown:

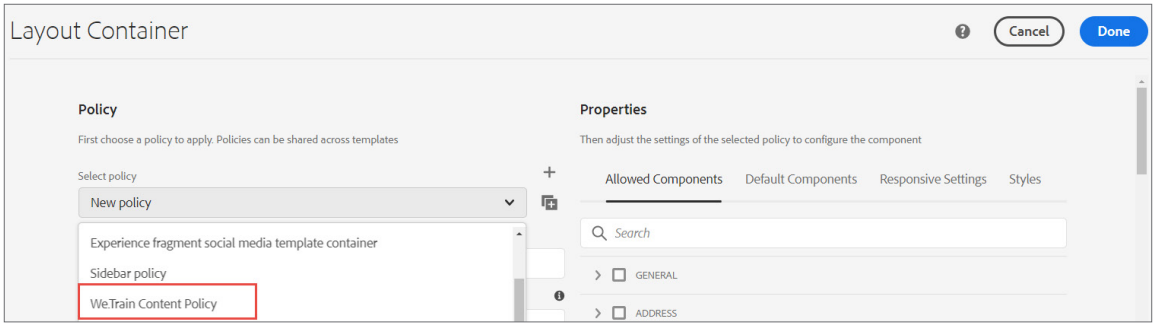


42. Select the **Stockplex** component and click the **Unlock structure component** (lock) icon from the component toolbar. The component is unlocked.

43. Select the **Layout Container** component and click the **Unlock structure component** (lock) icon from the component toolbar. The component is unlocked.

44. Select the **Layout Container** component and click the **Policy** icon from the component toolbar. The **Layout Container** wizard opens.

45. In the **Policy** section, select the **We.Train Content Policy**, as shown, from the **Select policy** drop-down menu.



46. Click **Done**. The policy is updated.

47. Navigate to the tab where the **Templates** console is open.

48. Select the **Stockplex Template**, and click **Enable** from the actions bar. The **Enable** dialog box opens.

49. Click **Enable**. The template is enabled.

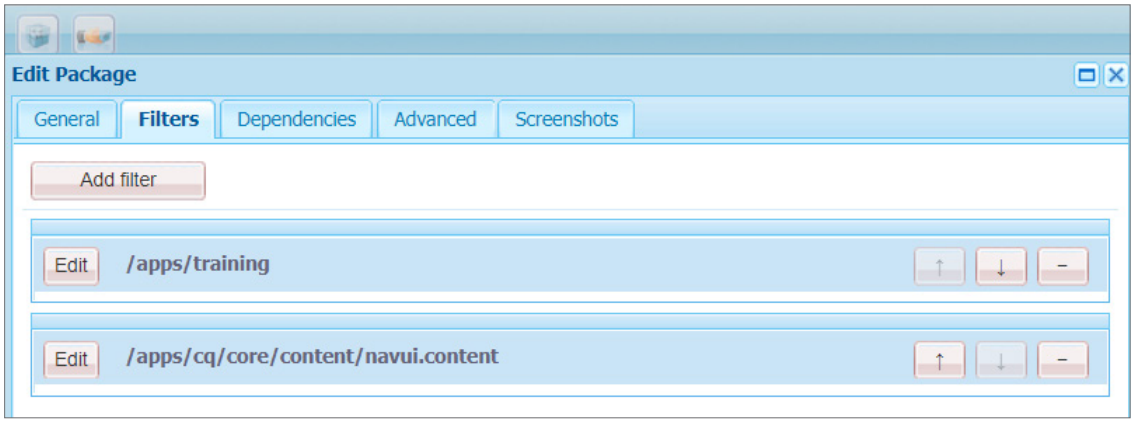
50. With the **Stockplex Template** already selected, click **Publish** from the actions bar. The **Publish** dialog box opens.

51. Ensure all check boxes are selected and click **Publish**. The **INFO The page has been published** success message appears.

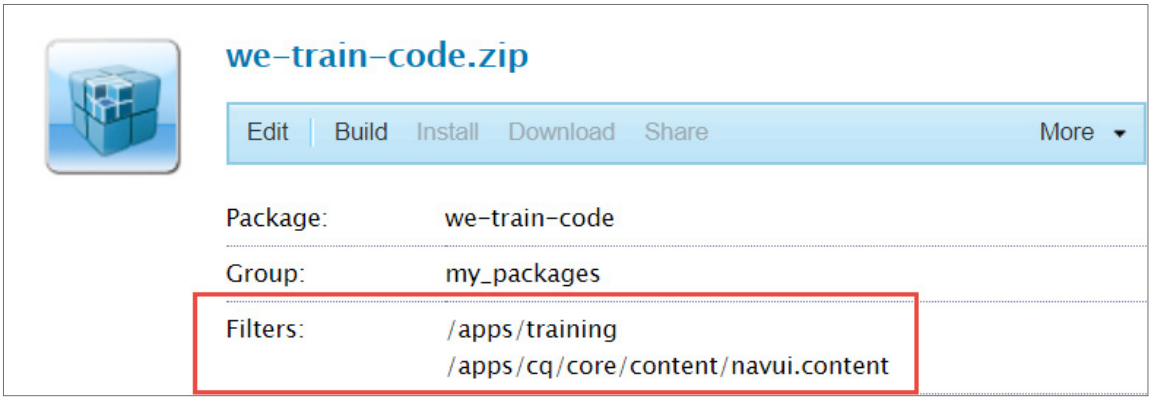
Exercise 4: Create We.Train content packages

This entire course has been developed using **CRXDE Lite**. In a realistic development scenario, all of your code changes should be done locally (files/folders), and then pushed or synchronized with a local AEM instance. To download the code and content from your local AEM instance, you need to create a content package. In this exercise, you will create two content packages—one for mutable content and one for immutable content in the JCR.

1. Click http://localhost:4502/crx/packmgr/index.jsp. The **Package Manager** opens.
2. Click **Create Package**. The **New Package** dialog box opens.
3. In the **Package Name** box, enter **we-train-code** and click **OK**.
4. Click **Edit** in the package you created now. The **Edit Package** dialog box opens.
5. In the Exercise Files folder provided to you, navigate to the **training-files** folder and open the **course-filter.txt** file. This file has all the JCR paths that you created or updated during this course.
6. On the **Filters** tab in **CRXDE Lite**, click **Add Filter** and add all the immutable paths, as shown, from the **course-filter.txt** file.



7. Click **Save**. The content package is created with applied filters, as shown:

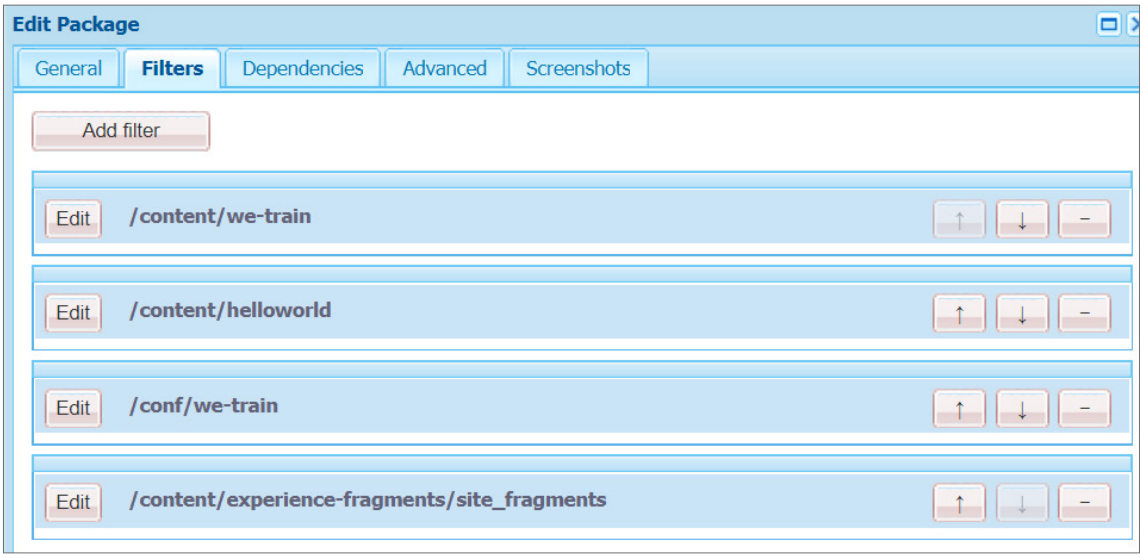


8. Click **Build.** The package is created.
9. Click **Download.** The **we-train-code** package is downloaded to your computer's default Downloads folder.

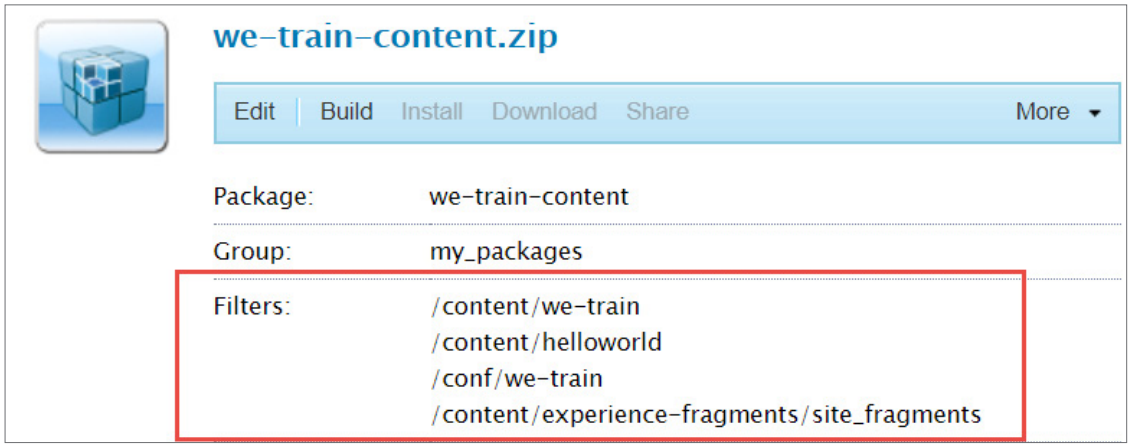
Note: If you do not remember how to create a content package or if you want to refer to detailed steps, you can refer the AEM Technical Basics student guide > Developer Tools module > Exercise 2: Create, build, and download packages.

---

10. Repeat steps 2 through 4 and create another content package called **we-train-content**.
11. On the **Filters** tab in **CRXDE Lite**, click **Add Filter** and add all the mutable paths, as shown, from the **course-filter.txt** file:



12. Click **Save**. The content package is created with applied filters, as shown:



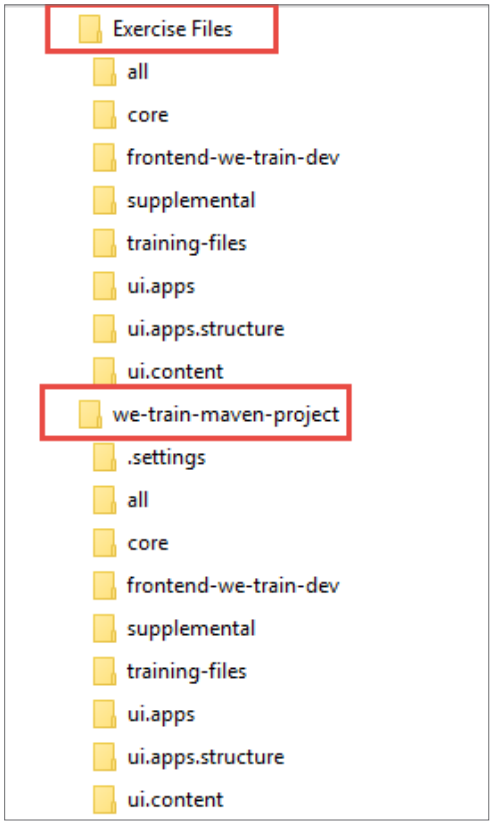
13. Click **Build.** The package is created.
14. Click **Download.** The **we-train-content** package is downloaded to your computer's default Downloads folder.

You should now have two packages—we-train-code.zip and we-train-content.zip. You will use these in the next exercise.

Exercise 5: Add your work to a Maven project

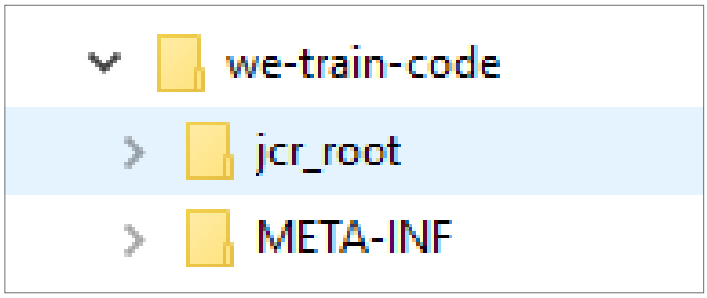
In the previous exercise, you created two content packages (**we-train-code.zip** and **we-train-content.zip**) of all of the code and content you created in this course. In this exercise, you will add these files to a valid Maven project to continue future development. The exercise folder you have been using during this course is a Maven project that contains a completed version of this course's code. To add your content packages to the Maven project, you will create a copy of the project and replace the code with your unique content packages.

1. Navigate to the Exercise Files folder provided to you and make a copy of the entire folder.
2. Rename the copy as we-train-maven-project, as shown:

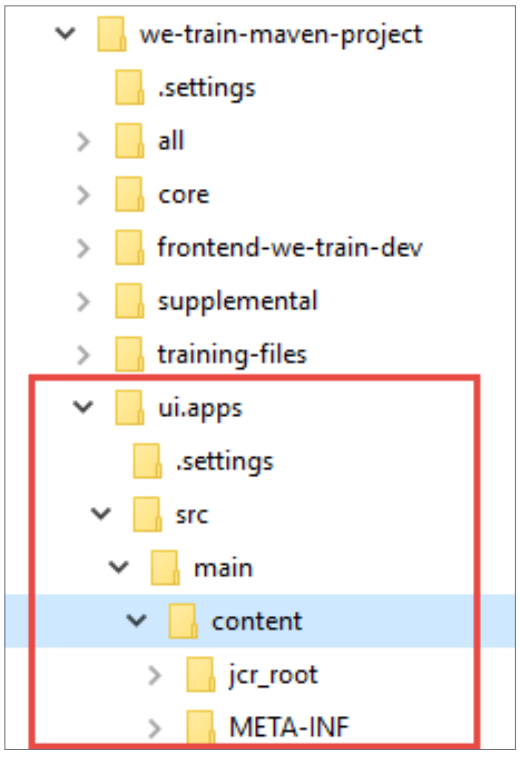


Note: All folders except training-files and supplemental are maven modules within the project. **ui.apps** is for immutable content and **ui.content** is for mutable content.

3. In another file explorer, navigate to the **we-train-code.zip** you downloaded in the previous exercise and unzip it.
4. Open the unzipped folder and copy the **jcr_root** and **META-INF folders**.



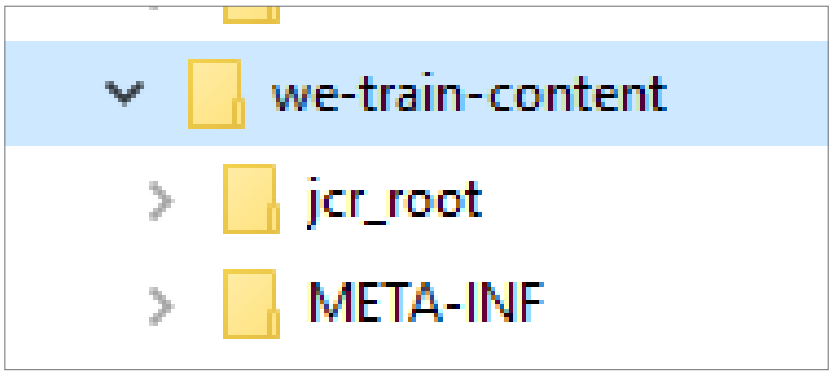
5. Navigate to the **we-train-maven-project** folder and replace the folders under ui.apps/src/main/content with the folders you copied in step 4.



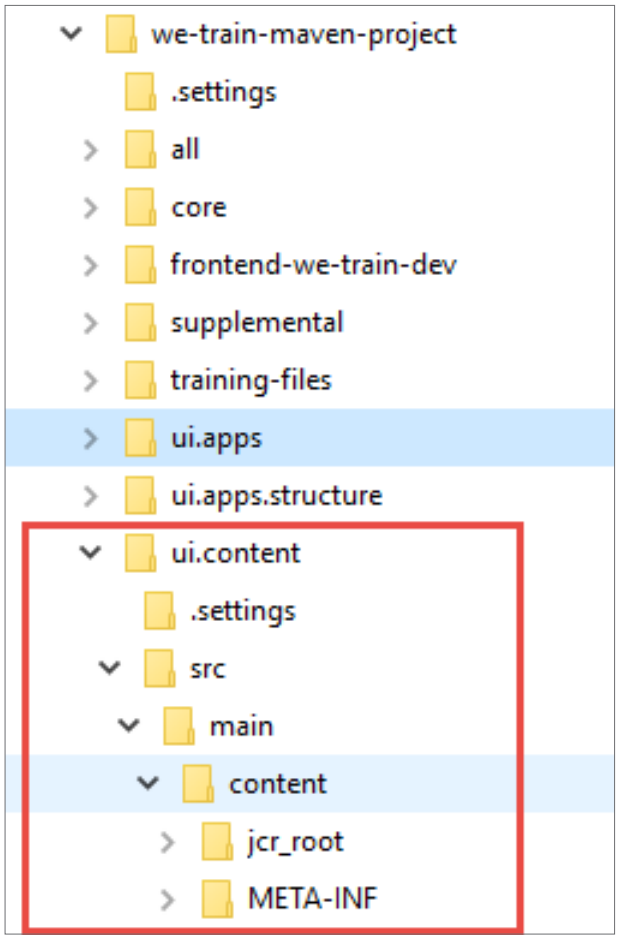
At this point, you have effectively replaced the immutable code with the unique immutable code you created during this course.

6. In the we-train-maven-project folder, navigate to ui.content/src/main/content and delete the jcr_root and META-INF folders.
7. In another file explorer, navigate to the we-train-content.zip you downloaded in the previous exercise and unzip it.

8. Open the unzipped folder and copy the jcr_root and META-INF folders.



9. Navigate to the we-train-maven-project folder and paste the copied folders under ui.content/src/main/content.



Now, you have effectively replaced the course content with the unique content you created during this course.

You now have a complete maven project with your code from this course in it. You can build and install this code with Maven on any AEM instance. You can also check this code into git or continue development with an IDE of your choice. To deploy your code onto a local AEM service running on port 4502, run the maven command: mvn clean install `-Padobe-public,autoInstallSinglePackage`

Note: In the next module, you will learn about front-end development using aemfed. Refer the *AEM Technical Basics* student guide.

Appendix

AEM and GDPR Compliance

The General Data Protection Regulation (GDPR) is the European Union's (EU) privacy law that defines data protection requirements.
"The EU General Data Protection Regulation 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 applies 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 AEM customers, so AEM 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
- AEM Foundation GDPR support