Develop Websites and Components in Adobe Experience Manager

STUDENT WORKBOOK

ADOBE DIGITAL LEARNING SERVICES

Develop Websites and Components in Adobe Experience Manager

05/10/2019

Playlist

Develop Websites and Components in Adobe Experience Manager

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

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

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

Module Number	Course Name	Module Name
1	AEM Technical Basics	Adobe Experience Manager Architecture
2	AEM Technical Basics	Adobe Experience Manager Installation
3	AEM Technical Basics	Developer Tools
4	AEM Technical Basics	Adobe Experience Manager Sites Authoring Basics
1 to 9	Develop Websites and Components in Adobe Experience Manager	All modules
7	AEM Technical Basics	Using Brackets for Development

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 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
- Create a base page component
- Create content to render on the page
- Explain the Sling resolution process
- Search for a rendering script
- Manipulate selectors
- Explain how Sling inheritance works
- Implement Sling inheritance through resourceType
- Modularize content by including other scripts

JCR Nodes, Properties, and Namespaces

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

Nodes

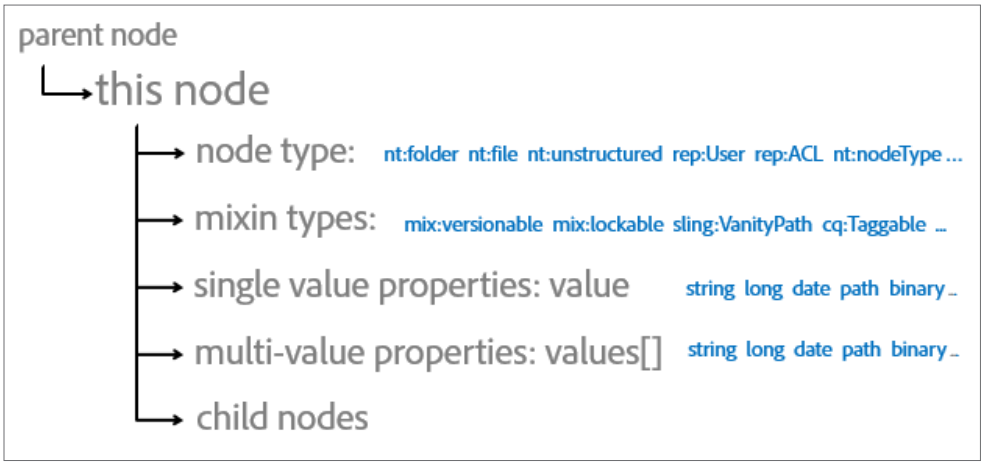
Nodes provide the structure and properties to store the data. The nodes of JCR can have:

- 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 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:

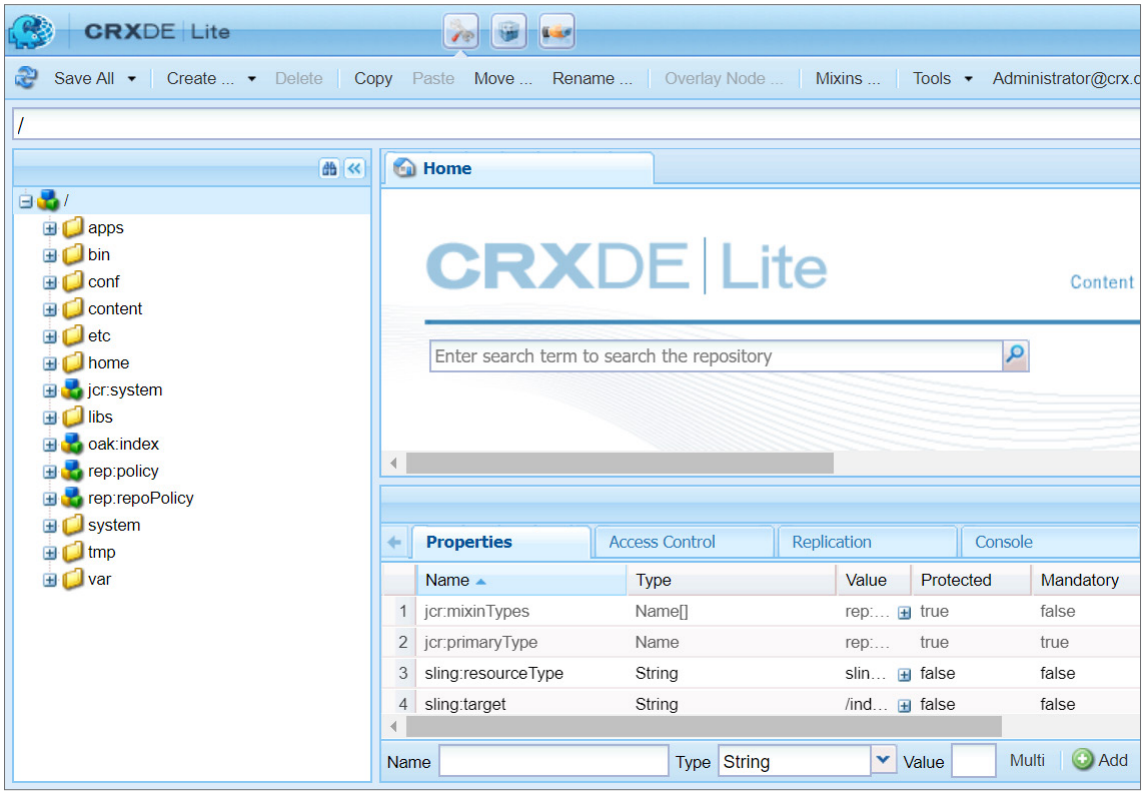
Folder	Description
/apps	Contains all project codes such as components, overlays, client libraries, bundles, i18n translations, and static templates created by an organization.
/conf	Contains all configurations for your website. This folder is used to store the dynamic templates and policies for your 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. You must not change the /libs path. For configuration and other changes, copy the item from /libs to /apps and make changes within /apps.

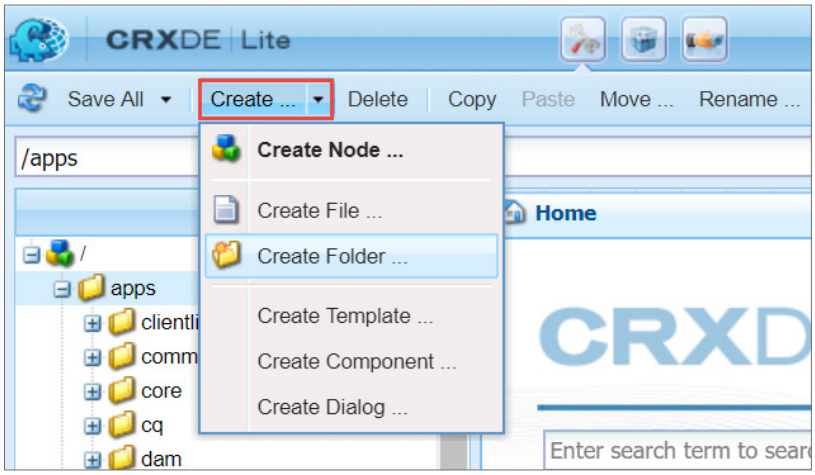
Exercise 1: Create a project structure in JCR

To create a project structure in JCR:

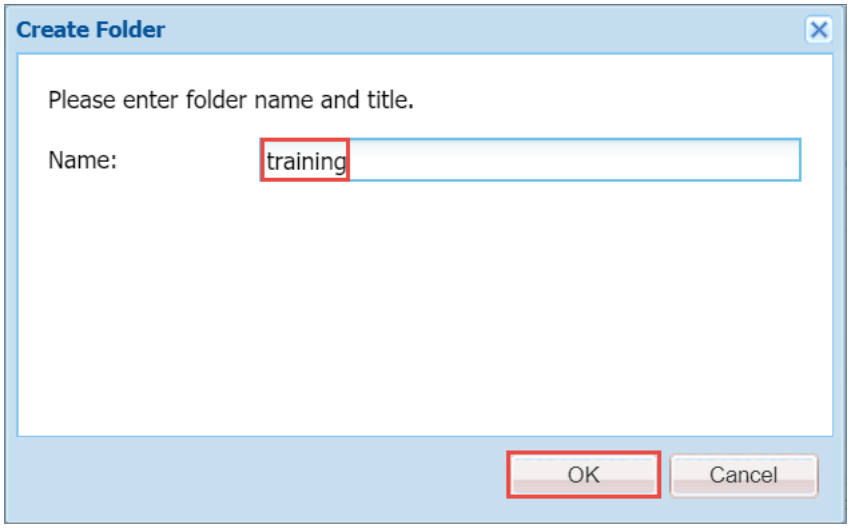
1. In the AEM author instance, click the **Adobe Experience Manager** icon in the upper left and click the **Tools** icon. The **Tools console** opens.
2. Click **CRXDE Lite**. The **CRXDE Lite** page opens in a new browser tab, as shown. You can also open http://localhost:4502/crx/de in a browser to 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.

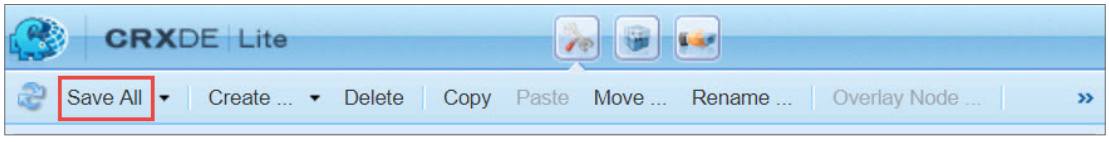


4. Enter **training** in the **Name** field and click **OK**, as shown. The training folder is created under the /apps folder.



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

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

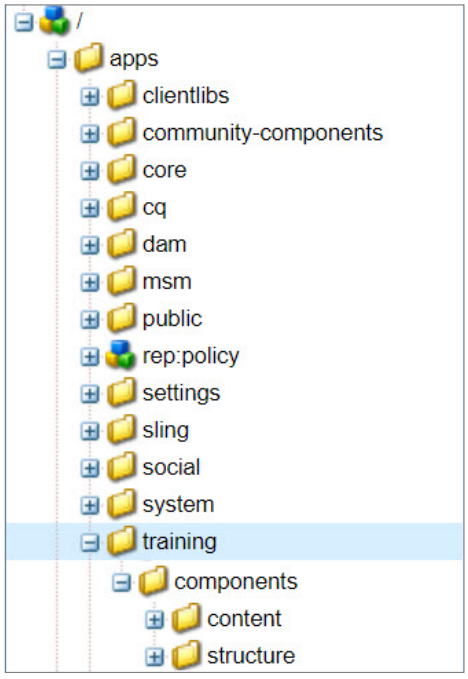


6. Navigate to **/apps**, and select the **training** folder you created now.

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** field, 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:

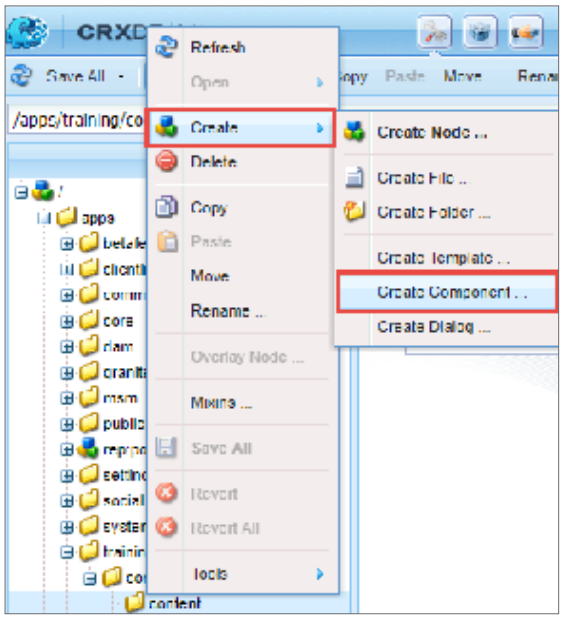
- cq:editConfig (cq:EditConfig): Defines the edit properties of the component and enables the component to appear in the components browser
- cq:childEditConfig (cq:EditConfig): Controls the author UI aspects for child components that do not define their own cq:editConfig
- cq:dialog (nt:unstructured): 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

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

Exercise 2: Create a HTL Component

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

1. In CRXDE Lite, navigate to **/apps/training/components/content**.
2. Right-click the **content** folder and click **Create** > **Create Component**, as shown. The **Create Component** dialog box opens.



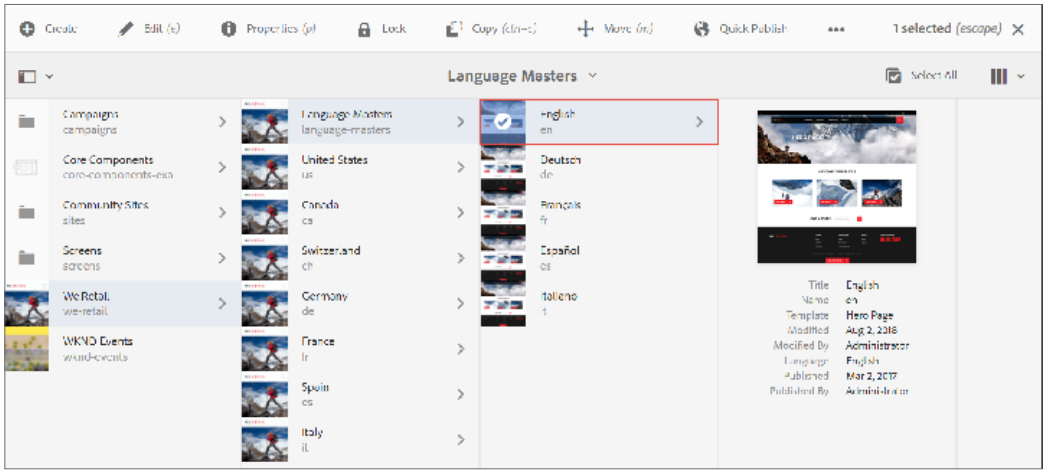
3. Enter the following details:
   a. In the **Label** field, enter **quote**.
   b. In the **Title** field, enter **Quote**.
   c. In the **Description** field, enter **My First Component**.
   d. Leave the **Super Type** field empty.
   e. In the **Group** field, enter **We.Retail** and click **Next**. The **Advanced Component Settings** are displayed.
   f. Leave the **Advanced Component Settings** blank.
   g. Click **OK**. The component is created.
   h. Click **Save All** to save the changes.

Note: The We.Retail componentGroup was used so that the component can be added to the We.Retail website.

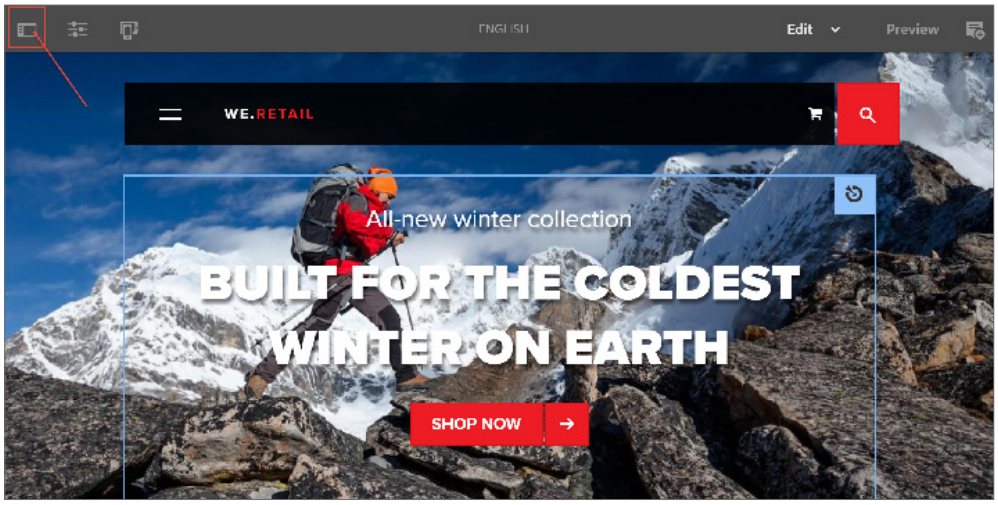
4. Expand the Quote component you created now by clicking the + icon next to it.
5. Right-click **quote.jsp,** and click **Rename** to rename it.
6. Rename it as **quote.html**.
7. Double-click **quote.html**. The editor opens on the right.
8. Delete the existing content in quote.html and replace it with the code from start-quote.html from your exercises folder.
9. Click **Save All** to save the changes.

To create a dialog, where components can be added:

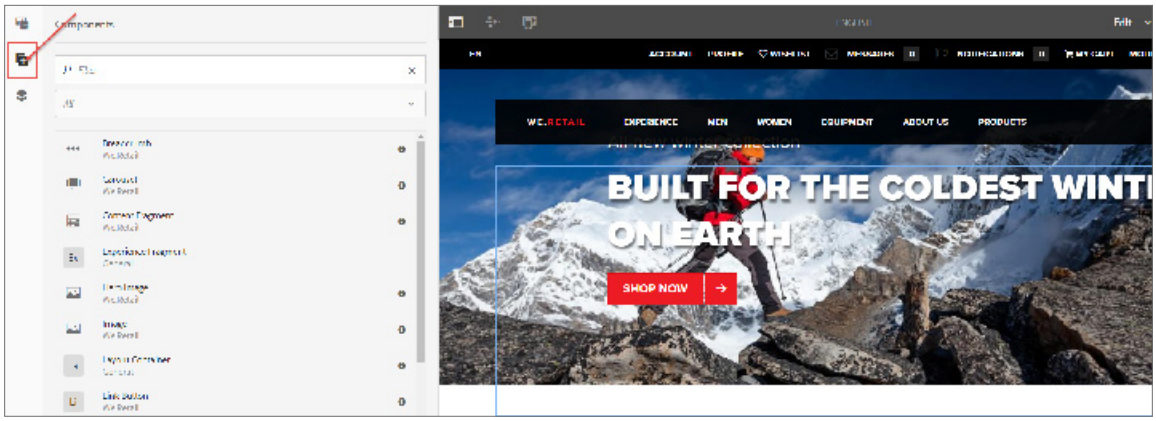
10. Right-click the **quote** component, and click **Create** > **Create Node**. The **Create Node** dialog box opens.
11. In the **Name** field, enter **cq:dialog**.
12. In the **Type** field, select **nt:unstructured** if it is already not selected.
13. Click **OK**. The node is created
14. Click **Save All** to save the changes.
15. In the AEM author instance, navigate to **Sites** > **We.Retail** > **Language Masters** > **English**, as shown:



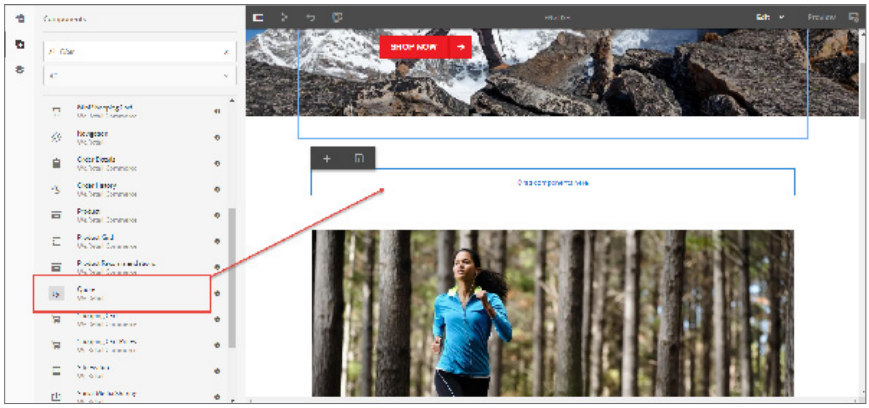
16. With the English page selected, click **Edit (*e*)** from the top menu bar to open the English page.
17. In the upper left, click the **Toggle Side Panel** icon, as shown, if you do not see the left panel.



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

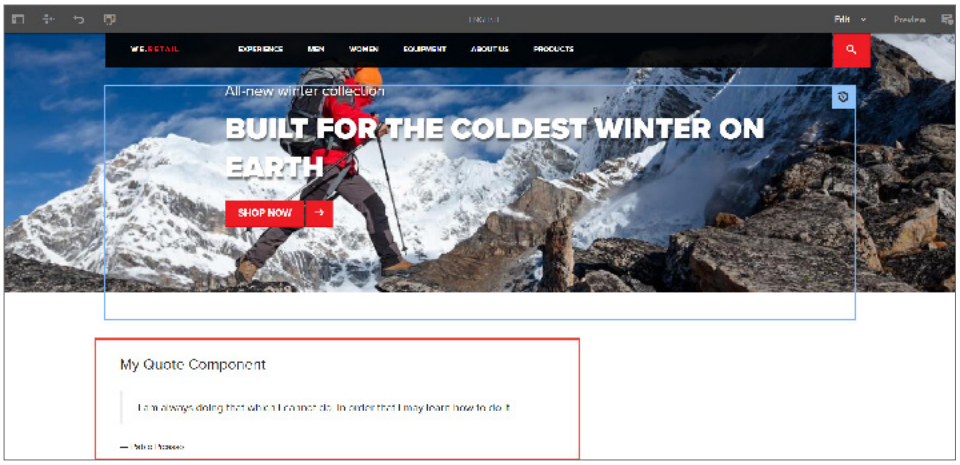


19. Look for the **Quote** component and drag and drop the component, as shown, onto the layout container:



Note: Even though the component was created in the /apps/training code base, you were still able to see and use the component on the We.Retail website. This is because the property componentGroup is We.Retail 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 tabs help the dialog to be short and sort the input fields.

Coral UI and Granite UI

The Coral UI and the Granite UI define the look and feel 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.

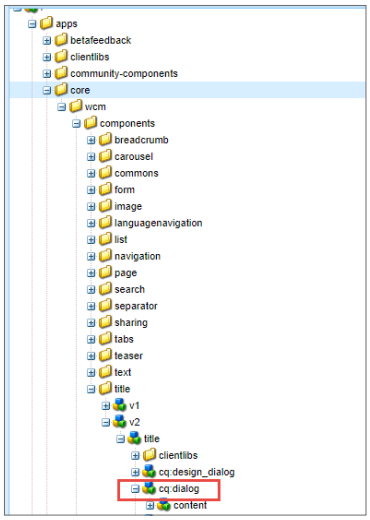
The cq:dialog node:

- 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 3: Create an edit dialog

In this exercise, you will create a dialog box that enables an author to add a value into 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:

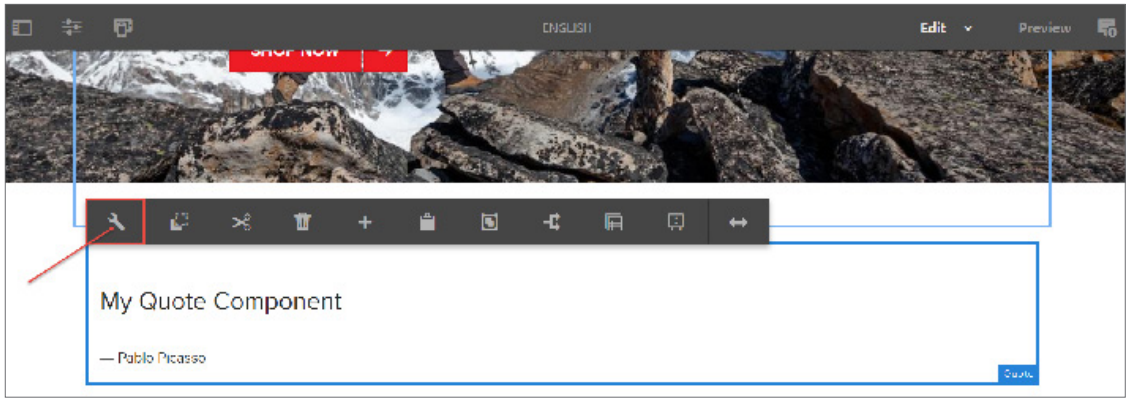


5. Right-click the **cq:dialog** node, and click **Copy**. The node is copied.
6. Navigate to **/apps/training/components/content/quote**.
7. Right-click the folder and click **Paste** to paste the node under the quote folder, as shown.
8. Select the **cq:dialog** node. You will see the properties of the cq:dialog node in the **Properties** tab on the right. 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.

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.

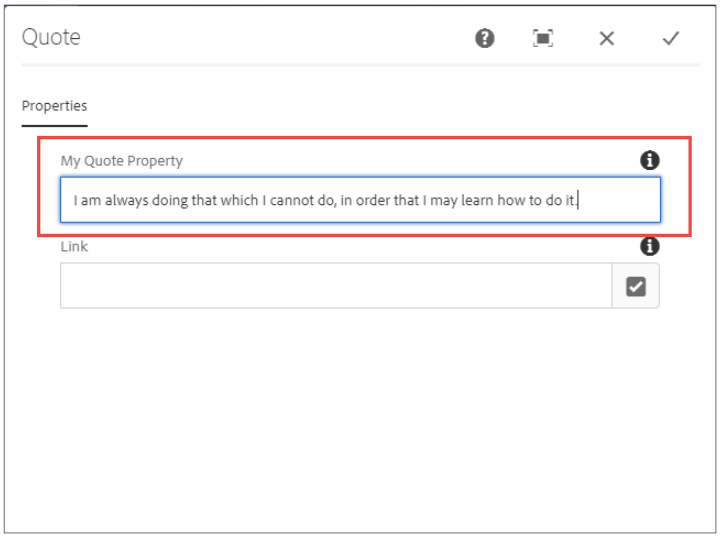
12. Double-click **quote.html**. The editor opens on the right.
13. Delete the existing content in quote.html and replace it with the code from the **dialog-quote.html** file provided to you. Notice the code `${properties.myQuote}`. By using `${}` expression, you are able to insert an object into the HTML code. properties is a global object that gives access to all properties on the current resource.
14. Click **Save All** to save the changes.
15. In the AEM author instance, navigate to **Sites** > **We.Retail** > **Language Masters** > **English**.
16. With the English page selected, click **Edit (e)** from the top menu bar to open the English page.
17. Click the **Quote** component on the page, and click the **Configure** icon, as shown. The **Quote dialog box** opens.



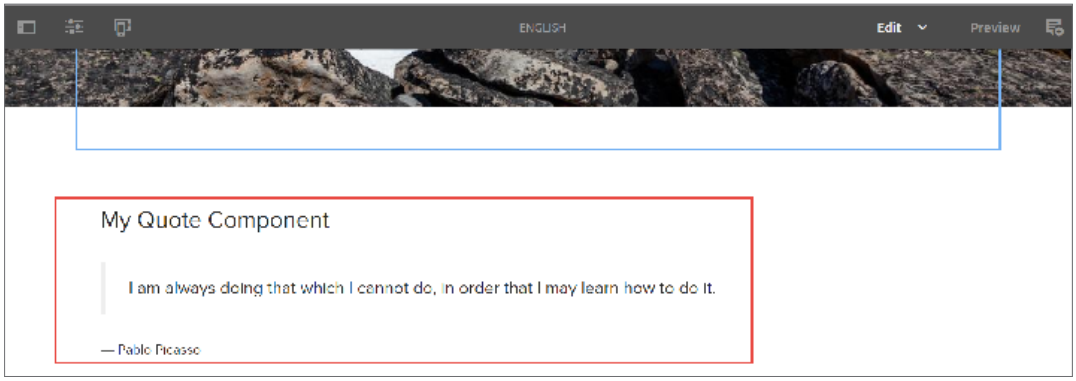
18. Add the following values to the dialog:

a. **My Quote Property**: I am always doing that which I cannot do, in order that I may learn how to do it.

b. Click **Done** (the checkmark at the top-right of the dialog).



19. Observe the rendered dialog values on the page.



Create a Base 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 realize a specific function.

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

Exercise 4: Create a base page component

To create a base page component:

1. In CRXDE Lite, navigate to the **/apps/training/components/structure** folder.
2. Right-click the **structure** folder and click **Create** > **Create Component**. The **Create Component** dialog box opens.
3. Enter the following values in their respective fields:

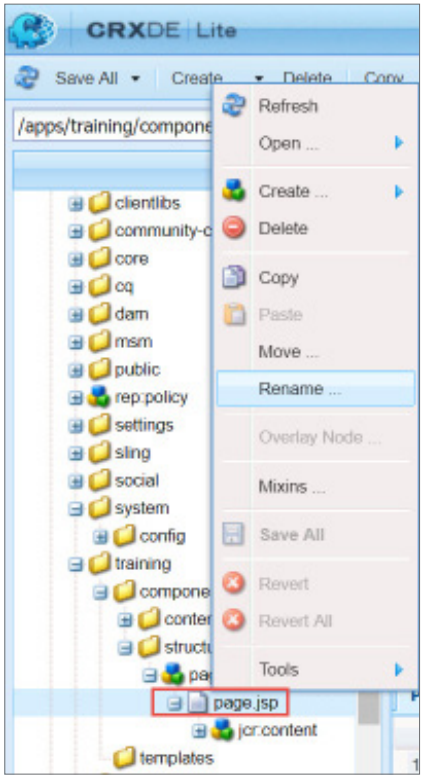
Field	Value
Label	page
Title	We.Train Page
Description	Initial We.Train page rendering component
SuperType	core/wcm/components/page/v2/page

4. Click **Next**. The advanced component settings are displayed.
5. Leave the **Advanced Component Settings** blank.
6. Click **OK**. The component is created.
7. Click **Save All** to save the changes.

Note: You need to 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.

8. Navigate to the **training/structure/page** node, and click the **+** (plus) sign to expand the page node.
9. Double-click **page.jsp**. The editor opens on the right.
10. Delete the existing content and click **Save All**.

11. Right-click **page.jsp**, and then click **Rename**, as shown. The field name becomes editable.



12. Rename **page.jsp** to **page.html**. Renaming the file helps populate the page content with HTML.
13. Click **Save All**.

NOTE: In a real development project, you will use an external IDE to develop your code. Adobe has an AEM sync tool for Brackets. Brackets is an enhanced text editor for front-end developers. If you want to try and use Brackets during this course, you can learn more from the last module of this workbook.

14. Double-click **page.html**. The editor opens on the right.
15. Replace the existing content with the code from the **page.html** file provided to you.
16. Click **Save All**.
17. Open **page.html** to view the new code that you added.

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

Note: If you are creating a new blank file, first save the file and then double-click the jcr:data property.

Exercise 5: Create content to render on the page

In this exercise, you will create a content node that will render the page component. This teaches the basics of rendering the code (the component) with content. Creating content through CRXDE Lite is not typical though. Typically, authors will create the content using the AEM Sites console. You saw an example of this with the quote component and the We.Retail website. Page components are different from typical content components. They require a page template to auto-create the content with the Sites console. You will create this in a later exercise. For now, use CRXDE Lite to quickly create a testing content node.

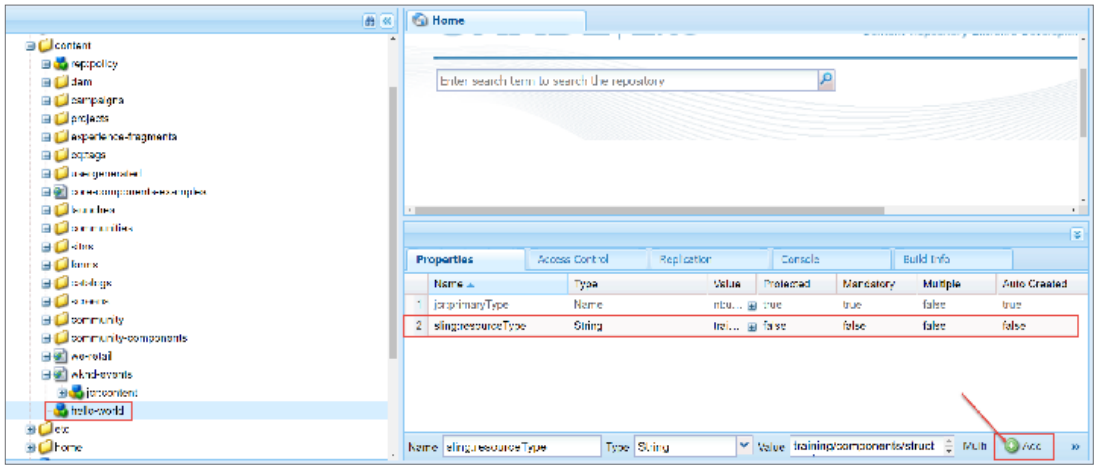
1. From CRXDE Lite, navigate to the **/content** folder.

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

2. Right-click the **/content** folder 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 node is created.
5. Click **Save All**.
6. Select the **hello-world** node and add the following properties under the **Properties** tab:

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

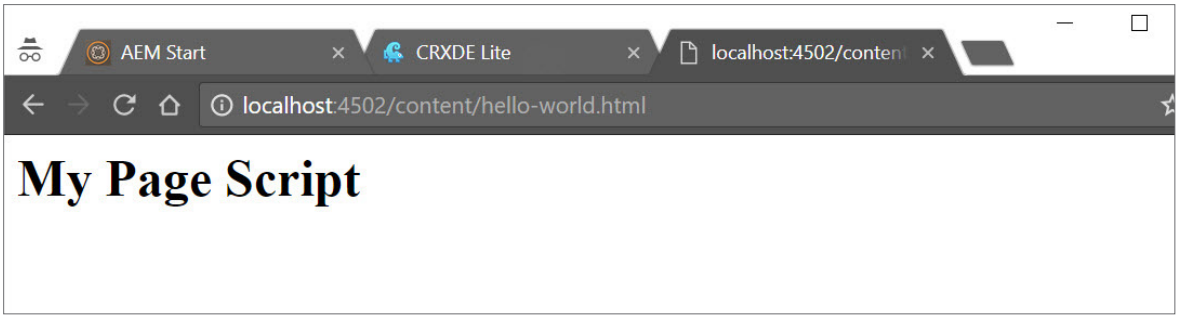
7. Click **Add**, as shown. The property is added to the node.



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 the page component (page.html) on the hello-world.html 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 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

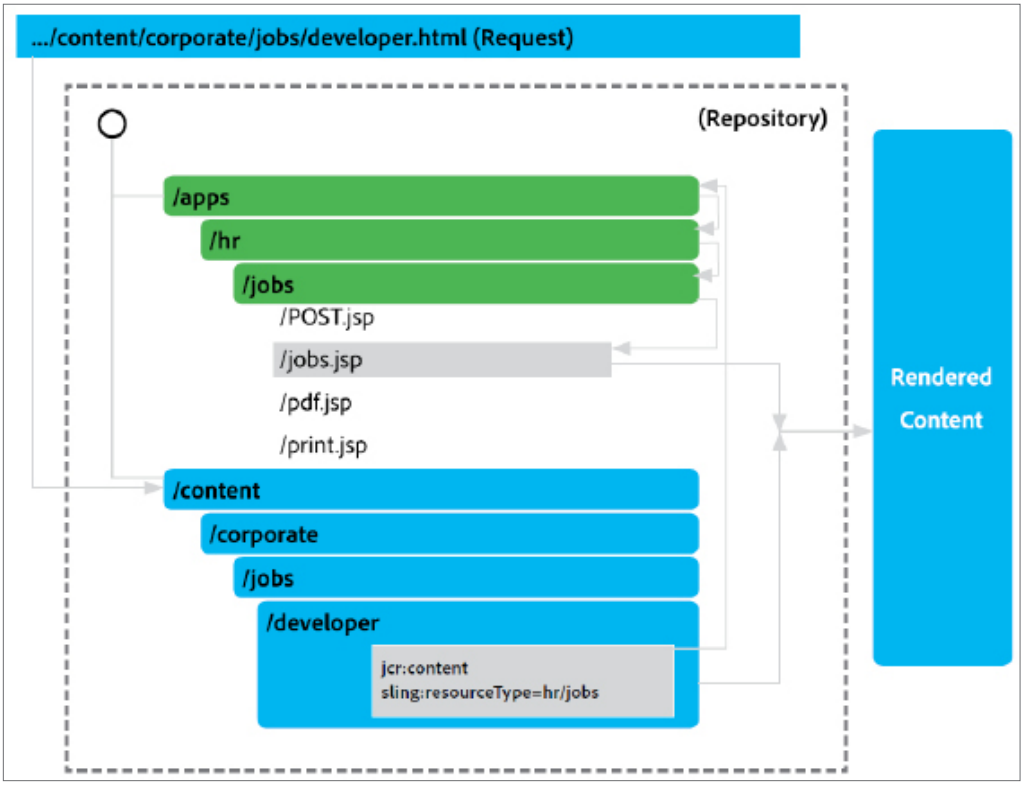
Resolving Requests to Resources

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:

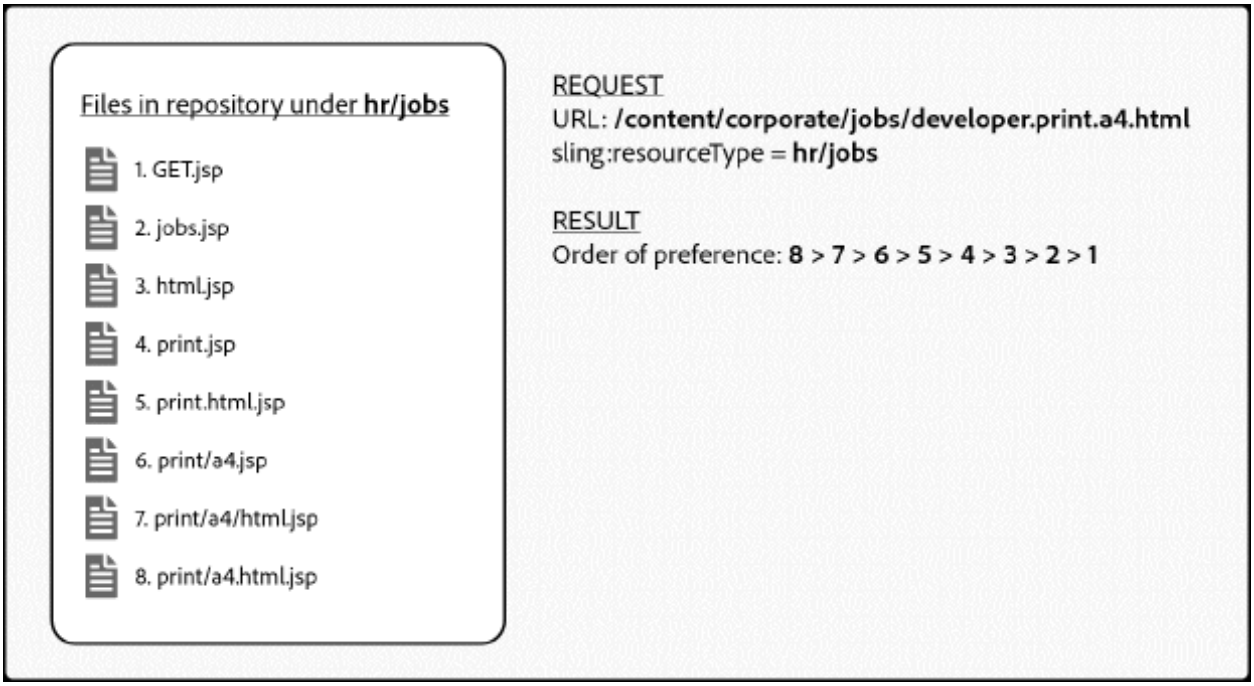
1. The Sling Resource Resolver process first looks for a redirection rule such as a vanity URL or a servlet. If the rule/servlet does not exist or is not found, the script resolution process begins.
2. As part of the script resolution process, Sling searches for the spy node.
3. If a node is found, the 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:



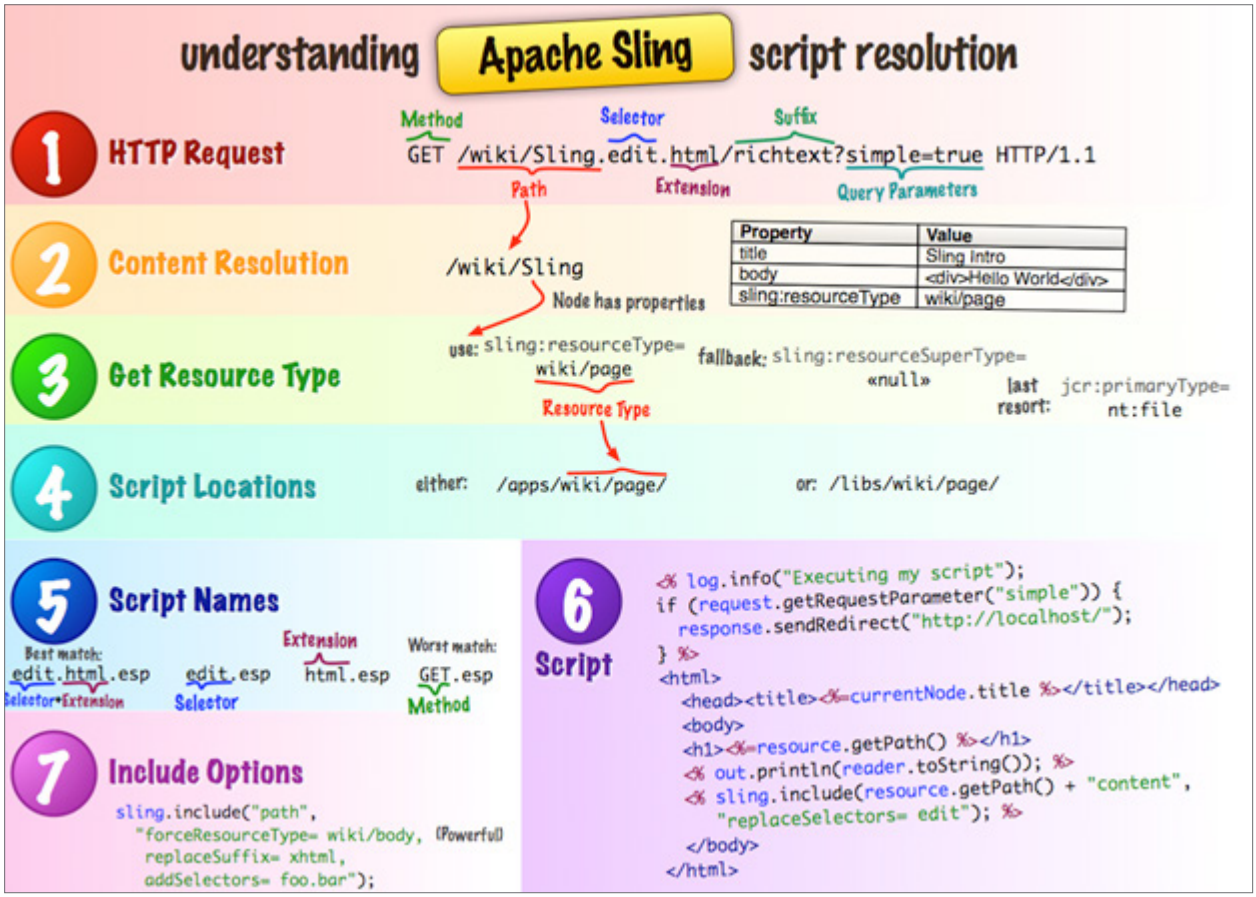
Locating and Rendering Scripts

When the resource is identified from the URL, its resource type property is located, and the value 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.



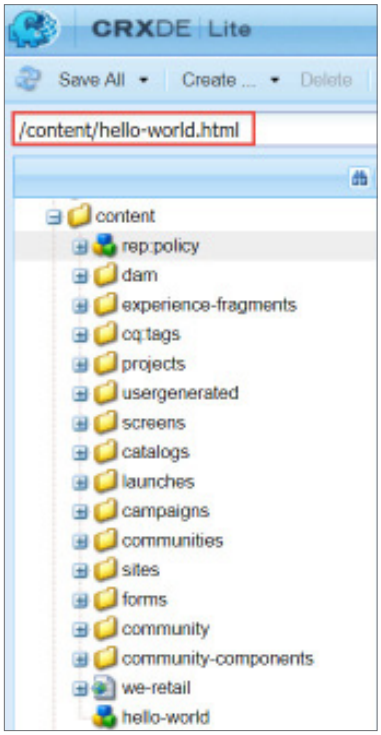
Exercise 6: Search for a rendering script

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

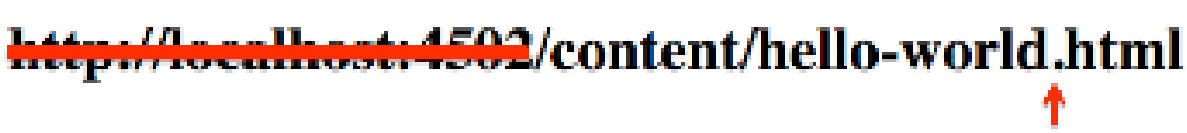
1. Disregard the protocol, server, port, and any suffix from the URL.
2. Examine the remaining portion of the URL, use the information contained therein to assist with resource solution, and find the associated rendering script.

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.

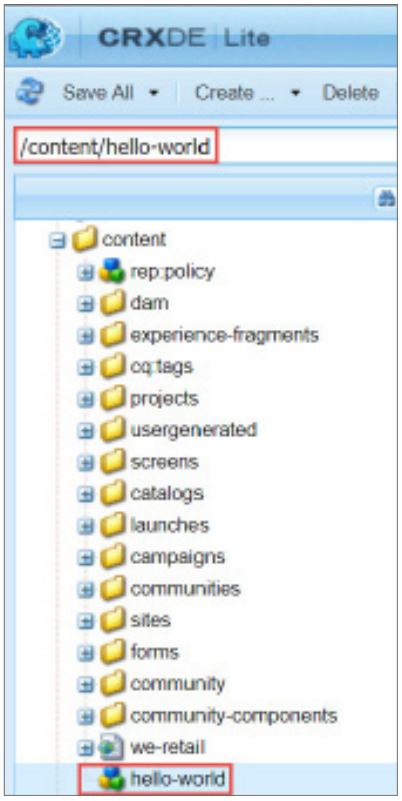
1. Consider the request (URL) http://localhost:4502/content/hello-world.html
2. Disregard **http://localhost:4502**.
3. Ensure the **CRXDE Lite** page is open. If not, click http://localhost:4502/crx/de to open the CRXDE Lite page.
4. Navigate to **/content/hello-world.html**. Notice that the path does not exist in the repository.



At this point, Sling will back off the remaining part of the request until the first ".", 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 have successfully resolved a request URL to a resource that is represented by a node in the repository.

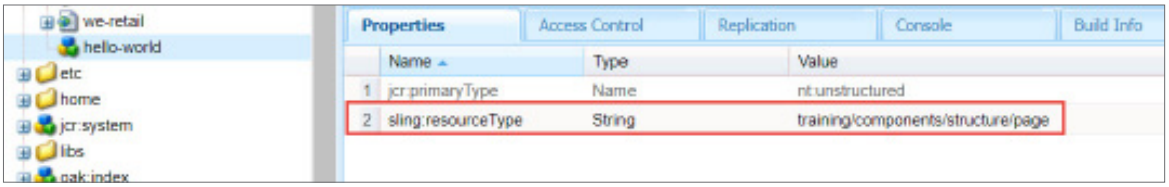


Note: You can map resources to the items in the relational databases and/or the file system. The Apache Sling specification does not mandate a JCR database.

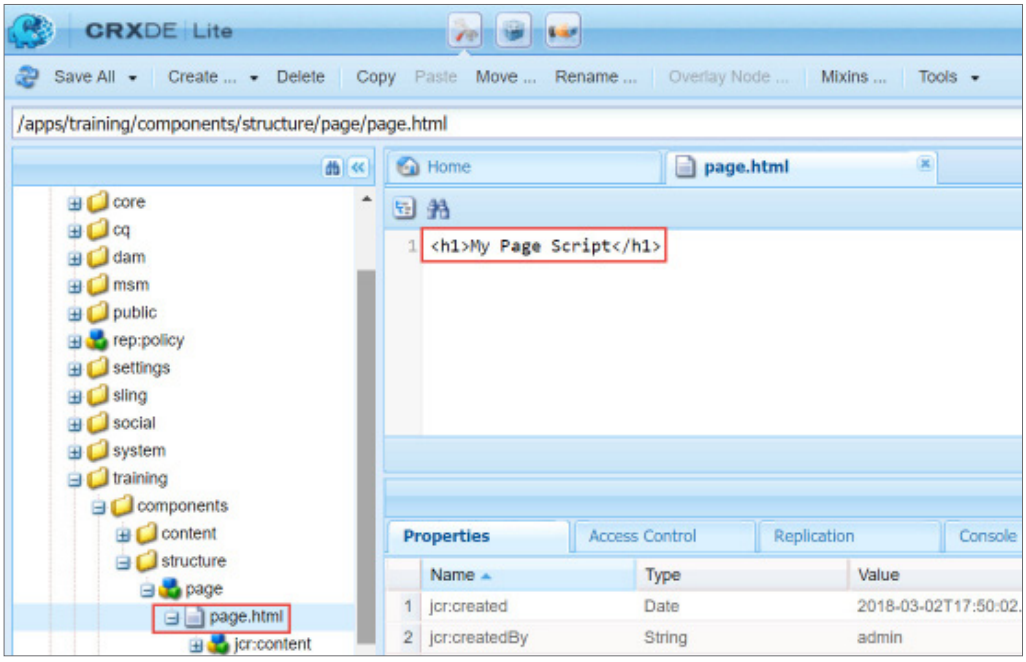
Now, you must find the rendering script.

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.

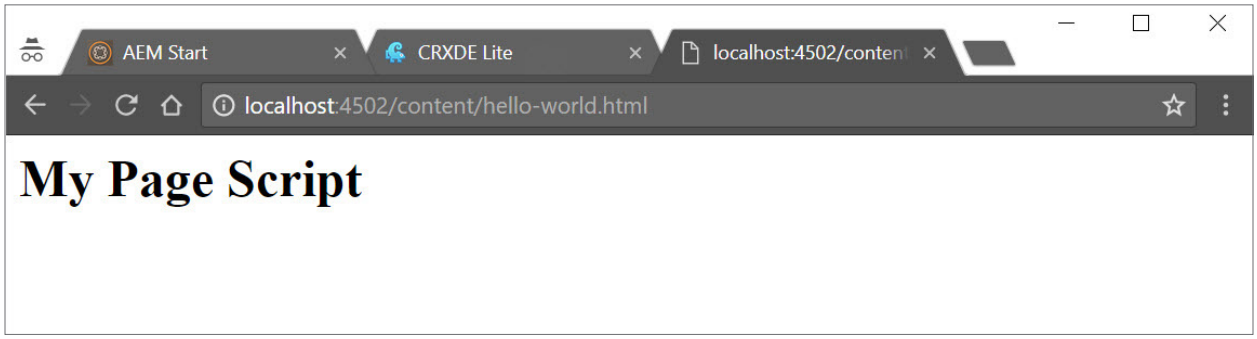
Notice that the **sling:resourceType** property points to the page-rendering script that you created previously:



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



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:



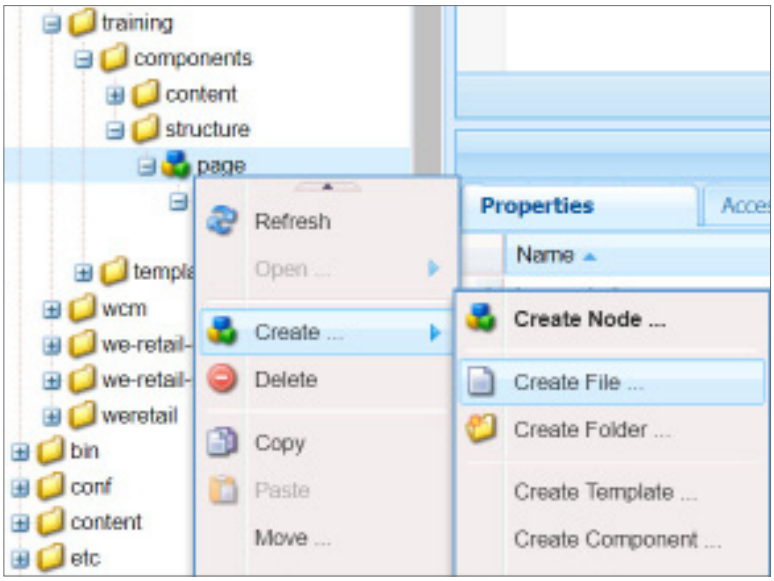
In this exercise, you have successfully resolved the resource, found the rendering script, and invoked the rendering chain.

Exercise 7: Manipulate selectors

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

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

1. From **CRXDE Lite**, navigate to **/apps/training/components/structure/page node**.
2. Select the **page** node, right-click it, and click **Create** > **Create File**, as shown. The **Create File** dialog box opens.



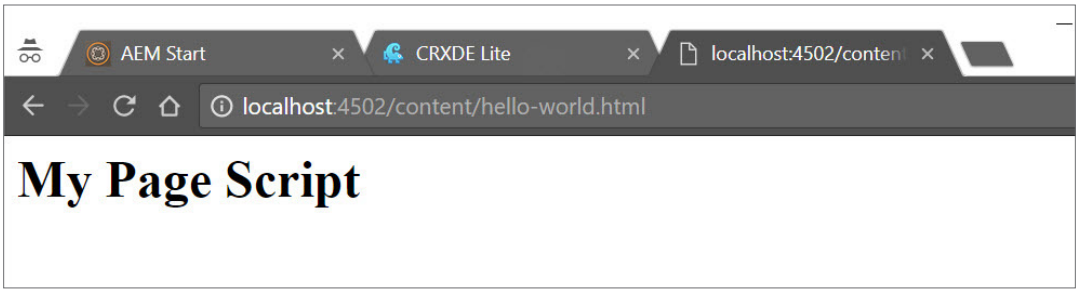
3. Enter **blue.html** in the **Name** field, and then click **OK**. The blue.html page opens in the left side panel.
4. Click **Save All**.

To add code to the blue.html page:

5. Double-click **blue.html**. The editor opens on the right.
6. Add the code from the blue.html file located in the Introduction to Content Rendering exercise folder provided to you.
7. Click **Save All**.

To test the selector:

8. Open http://localhost:4502/content/hello-world.html in a browser. The hello-world node is rendered, as shown:



9. Add blue to the URL (http://localhost:4502/content/hello-world.blue.html) and refresh the page. Notice the difference. 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, and so on), and dialog box layout (inline, floating, and so on) 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 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 helps extend and customize a component.

Exercise 8: Inheritance with resourceSuperType

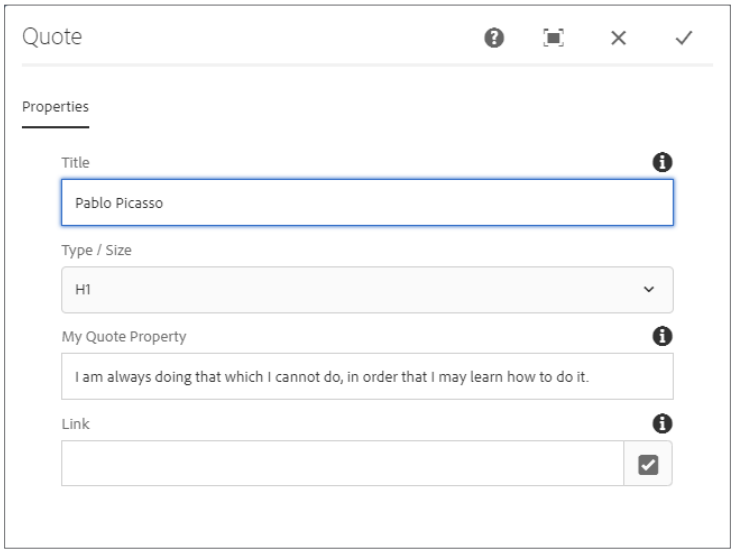
In this exercise, you will extend the quote component to inherit from the core title component. The sling:resourceSuperType property will be used to inherit from the core component rather than recreating code and dialog boxes.

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

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** > **We.Retail** > **Language Masters** > **English** in the AEM author instance.
6. Click the **quote** component you added earlier, and click **Configure** (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 could also use these fields in quote.html if needed.

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

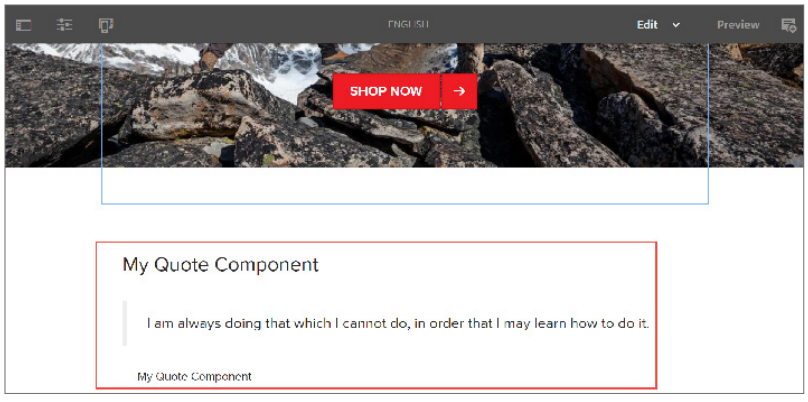


8. Click **Done**. (The checkmark on the top-right of the dialog.)

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.

---

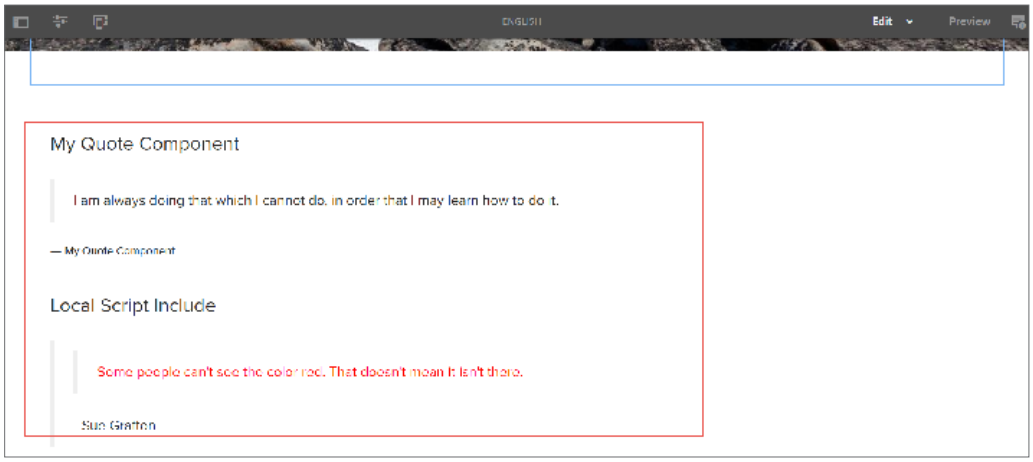
9. Navigate back to CRXDE Lite.
10. Navigate to **apps/training/components/content/quote**.
11. Double-click **quote.html**. The editor opens on the right.
12. Delete the existing content in quote.html and replace it with the code from the supertype-quote.html file provided to you.
13. Click **Save All** to save the changes.
14. In AEM, navigate to **Sites** > **We.Retail** > **Language Masters** > **English**. With the English page selected, click **Edit (*e*)** from the top menu bar to open the page.
15. Observe the new title.



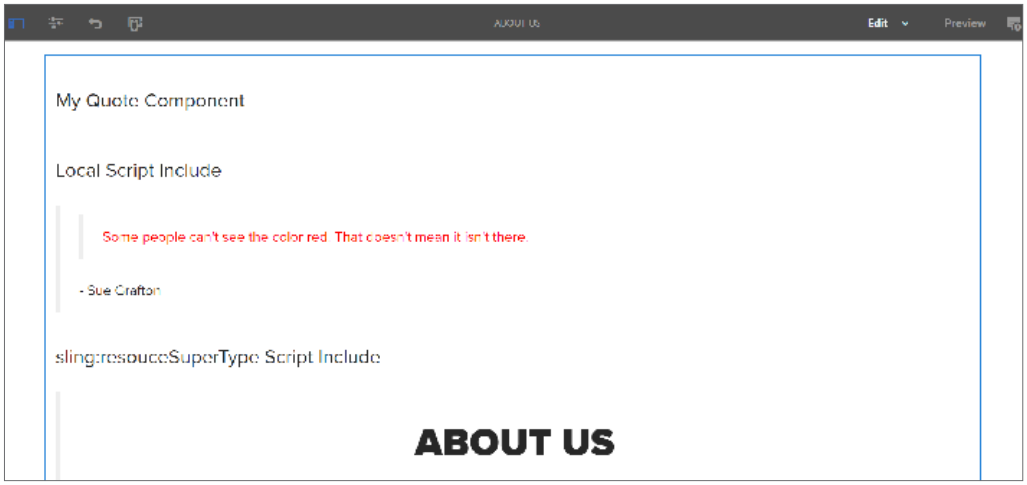
Exercise 9: Modularize content by using multiple scripts

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** field, enter **red.html** and click **OK**. The file is created.
4. Double-click **red.html**. The editor opens on the right.
5. Add the code from the red.html exercise file provided to you.
6. Click **Save All** to save the changes.
7. Open **quote.html** and replace the existing code with **includelocal-quote.html** from the exercises files provided to you.
8. Click **Save All** to save the changes.
9. To view the newly included script, navigate to **Sites** > **We.Retail** > **Language Masters** > **English**. 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.



10. 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.
11. Open **quote.html**, and replace the exisitng code with the code from the **include-quote.html** exercise file provided to you.
12. Click **Save All**.
13. To view the newly included script, navigate to the **Sites** > **We.Retail** > **Language Masters** > **English** > **About Us** page. With the About Us page selected, click **Edit (*e*)** from the top menu bar to open the page.
14. Click the **Toggle Side Panel** icon.
15. 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

Use the following links for more information on:

- Script Resolution:

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

- URL Decomposition:

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

- Sling Cheat sheet:

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

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. The AEM reference site—We.Retail that is available out-of-the-box—is built by using HTL.

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

HTL differs from other templating systems in the following ways:

- HTL is HTML5: A template created in HTL is a valid HTML5 file. All HTL-specific syntax is expressed within a data attribute or within HTML text. Any HTL file opened as HTML in an editor will automatically benefit from 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 of the basic HTL statements, expressions, and tags are:

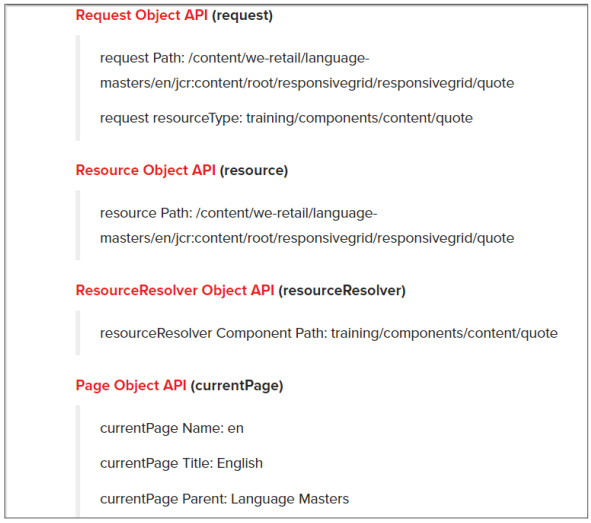
- Comments (HTL comments are HTML comments with additional syntax. They are delimited as shown below):
  - Example: `<!--/* An HTL Comment */-->`
- Expressions:
  - Examples: `${true}, ${properties.text}`
- URI Manipulation:
  - Example: `${'example.com/path/page.html' @ scheme='http'}` (Resulting output: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

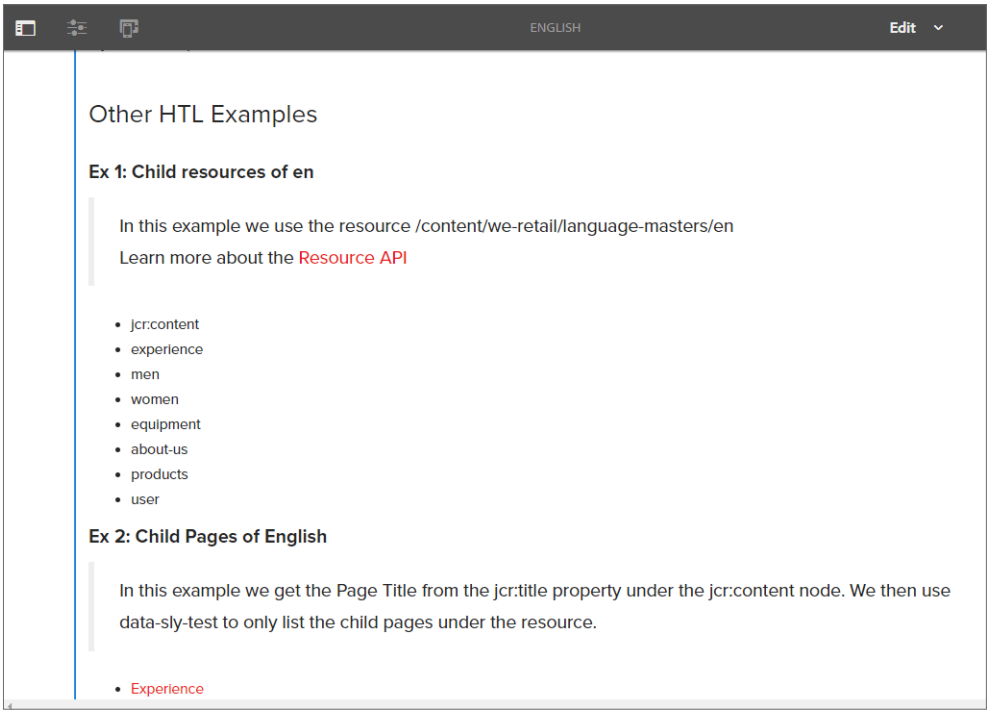
1. Open CRXDE Lite (http://localhost:4502/crx/de) If it is not already open.
2. Navigate to the **/apps/training/components/content/quote** component.
3. Double-click **quote.html** to update the code. The editor opens on the right.
4. Replace the existing code with the code from **globalobjects-quote.html** located in the Introduction to HTML Template Language exercise folder provided to you.
5. Click **Save All** to save the changes.
6. In the AEM author instance, navigate to **Sites** > **We.Retail** > **Language Masters** > **English**.
7. With the English page selected, click **Edit** (***e***) in the actions bar to open the page in edit mode.
8. Examine the Quote component on the page. If you do not see the Quote component on the English page, drag and drop the **Quote** component from the **Components** tab on the left panel onto 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 the desired values needed.



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 observe 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.
3. Delete the existing content and replace it with the code from **htl-quote.html** located in the Introduction to HTML Template Language exercise folder provided to you.
4. Click **Save All**.
5. In the AEM author instance, navigate to **Sites** > **We.Retail** > **Language Masters** > **English**. Observe the newly rendered quote script. Notice how the different HTL attributes allow for differently rendered HTML. Carefully inspect the comments in your new quote.html script to better understand what these attributes are accomplishing.



References

Use the following links for more information on:

- HTL resources:

https://helpx.adobe.com/experience-manager/htl/using/update.html
https://helpx.adobe.com/experience-manager/htl/using/overview.html
https://helpx.adobe.com/experience-manager/htl/using/getting-started.html

- HTL Specifications:

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

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
- 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.
- 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 in the pages. Template authors create and manage editable templates.

Creating Templates: Roles

Template creation requires collaboration between the following roles:

- Admin: Creates a new folder for templates.
- Developer: Focuses on technical/internal details and provides the template author with 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

In AEM, components are the structural elements that constitute the content of the pages being authored. Core components make page authoring more 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 instance, create proxy components, load the core styles, and add components to the templates.

Core components must not be directly referenced from the content. To avoid direct reference, the Core components are added to a hidden component group (.core-wcm or .core-wcm-form). This prevents displaying 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 refers to a Core component as their super type. The proxy components do not contain anything and serve mostly to define the version of a component to use in the site. However, when customizing the Core components, these proxy components play an essential role for markup and logic customizations.

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

Responsive Layout Editing

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

Responsive Design

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

Responsive design provides:

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

Making Content Responsive

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

Traditional Workflow

In a traditional workflow:

- A designer mocks different breakpoints
- A developer implements the breakpoints for a specific template
- 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:

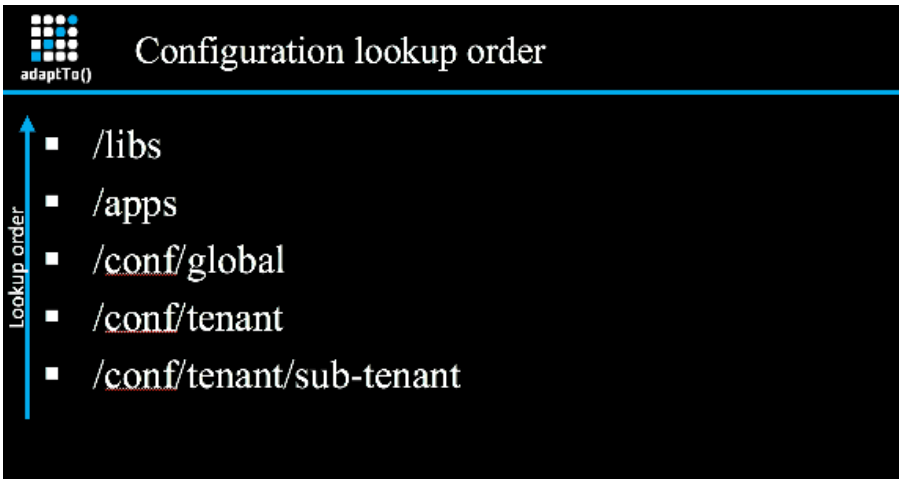
- Define differing content layouts based on device width (related to device type and orientation) by using breakpoints
- Ensure that the content is responsive to the size of the browser window on the desktop by using the same breakpoints and content layouts
- Place components in the grid, resize as required,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

Context-Aware Configurations

The context-aware configuration's default implementation consists of 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 of the following locations:

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

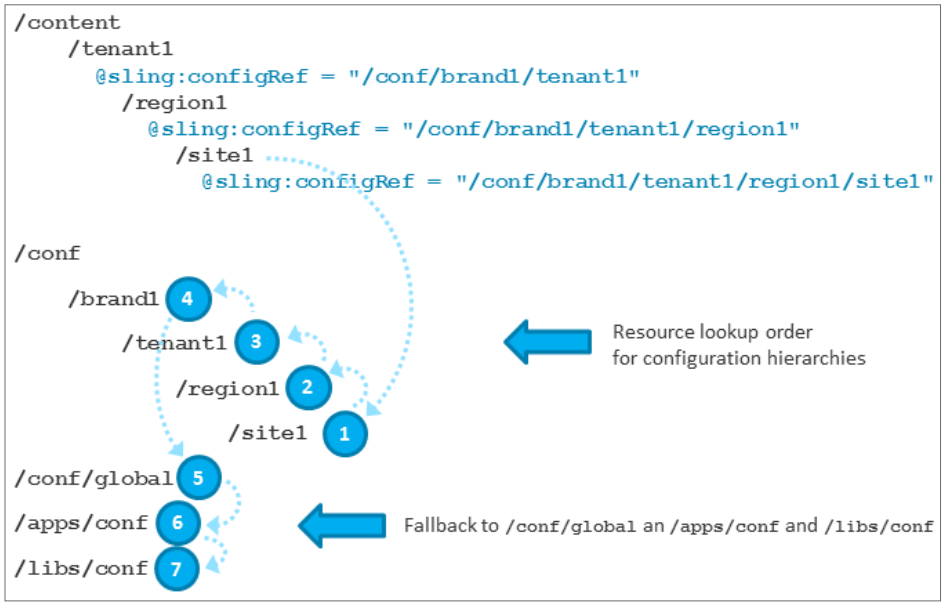
Currently, context-aware configurations support:

- 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:



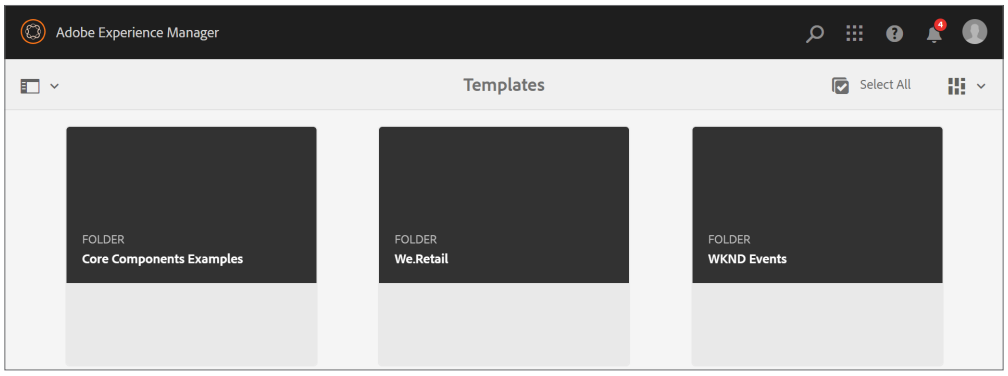
The rules you must follow in resource lookup are:

- 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 1: Create a context-aware configuration

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

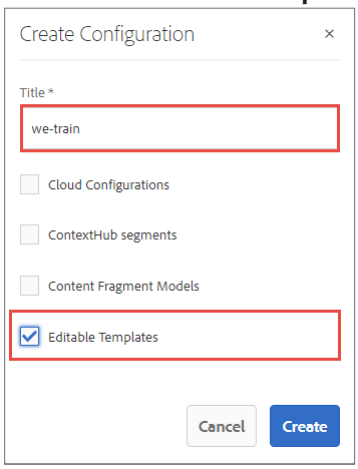
1. Click **Adobe Experience Manager** from the header bar to open the navigation pane.
2. Click the **Tools** icon, and click **Templates**. The **Templates** console opens. Notice that there is no way to create a new template folder in the **Templates** console.



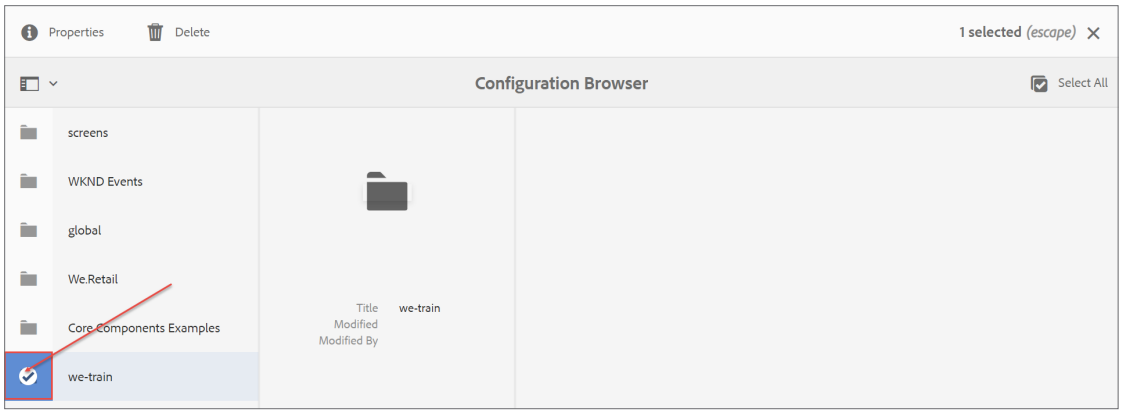
To create a new folder for editable templates, you must create a context-aware configuration.

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** field, enter **we-train**.
6. Select the **Editable Templates** checkbox.

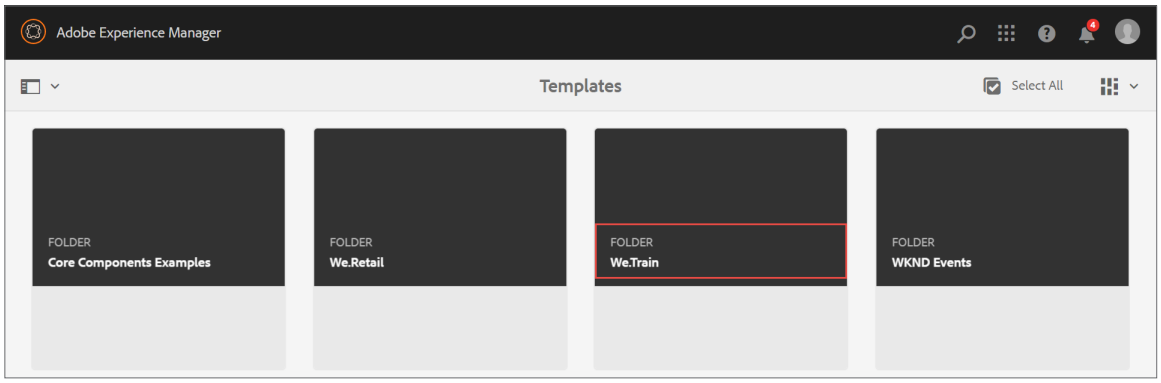


7. Click **Create**. A green success message appears at the top of the page, confirming the creation of the configuration.
8. Click the thumbnail beside the **we-train** configuration, as shown, and click **Properties** from the actions bar. The **Configuration Properties** page opens.



9. Update the title to **We.Train**.
10. Click **Save & Close**. A green success message appears at the top of the page.

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

Use the following links for more information on:

- Templates:
https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/templates.html
- Editable Templates:
https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/page-templates-editable.html
- Responsive Design:
https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/responsive.html
- Context-Aware Configurations:
http://sling.apache.org/documentation/bundles/context-aware-configuration/context-aware-configuration-default-implementation.html
- AEM Core Components:
https://github.com/adobe/aem-core-wcm-components

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 page(s) created from the template. Page authors can edit these unlocked components on the resultant page(s). In the initial content mode (and on the resultant pages), you can delete any unlocked components with an accessible parent, such as components within a layout container.
- Layout: 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 instance, the global folder exists in the Templates console. This folder contains the default templates and acts as a fallback if no policies and/or template-types are found in the current folder. You can add your default templates to global folder or create a new folder (recommended).
- 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

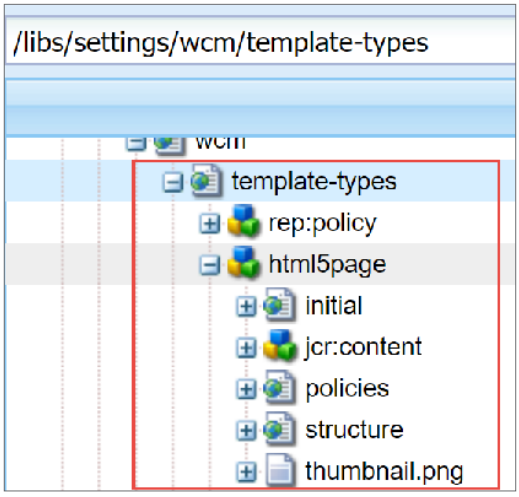
You can access the **Configuration Browser** console from the **Tools** console. You can create a multi-tenant environment where tenants can have different editable templates, configurations, cloud services, and data models.

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 in any pages created with the template.

The root (structure/jcr:content/root) node defines the list of components that will be available in the resulting page. The components defined in the template structure cannot be moved or deleted from

any resultant pages. After a component is unlocked, the editable property is set to true. After a component that already contains content is unlocked, 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 content is added to a component in Structure mode and that the component is subsequently unlocked (or vice versa), the content is used as initial content.

policies

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

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

Layout

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

Page Policies

Page policies help define the content policy for the page (main parsys) in either the template or 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 immediately. It is also possible to define the initial content for a template, which will be copied over to newly-created pages.

Enabling the Template

Before you use a template, you 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 path(s)** 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 out of the box.

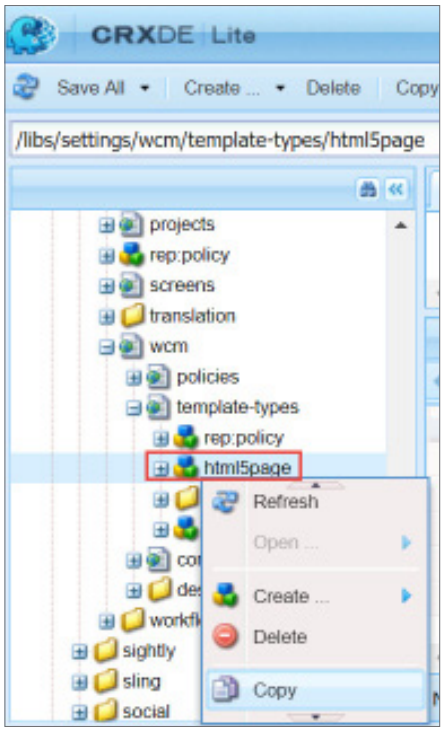
1. Navigate to **Adobe Experience Manager** > **Tools** > **Templates** if you are not in the **Templates** console already.
2. Click the **We.Train** folder and click **Create** from the actions bar. The **Create Template page** opens. Notice that two template-types are available now. These are the default template-types and if you used 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 global areas of the site are consistent. This includes loading of global CSS and Javascript as well as the inclusion of code that will ensure the page can be edited by using AEM authoring tools. You will use the training/components/structure/page component you created earlier.

To create a custom template type:

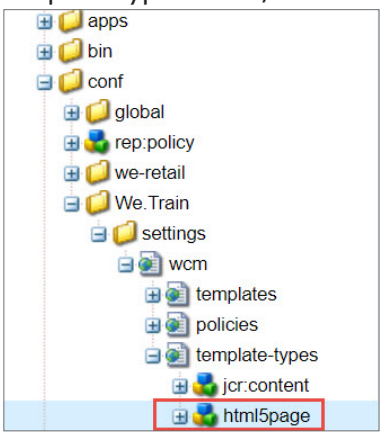
4. Click **Adobe Experience Manager** from the header bar. The **Tools** console opens.
5. Click **CRXDE Lite**. The **CRXDE Lite** page opens.
6. Navigate to the **/libs/settings/wcm/template-types/html5page** node.

7. Right-click the **html5page** node and click **Copy** from the drop-down menu, as shown:



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**.

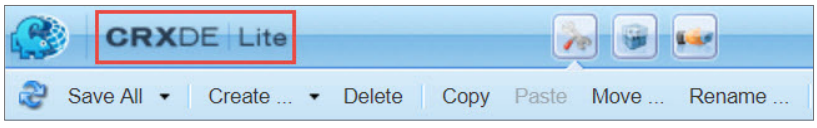
12. Rename the node to **empty-page**.

13. Select the**/jcr:content** node under **empty-page**.

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**
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**
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**
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 the **jcr:data**. The **Edit jcr:data** dialog box opens.
26. Click **Browse**. The **Open** dialog box opens.
27. Navigate to the **Creating Editable Templates and Pages in Adobe Experience Manager** exercise folder on your file system, 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.

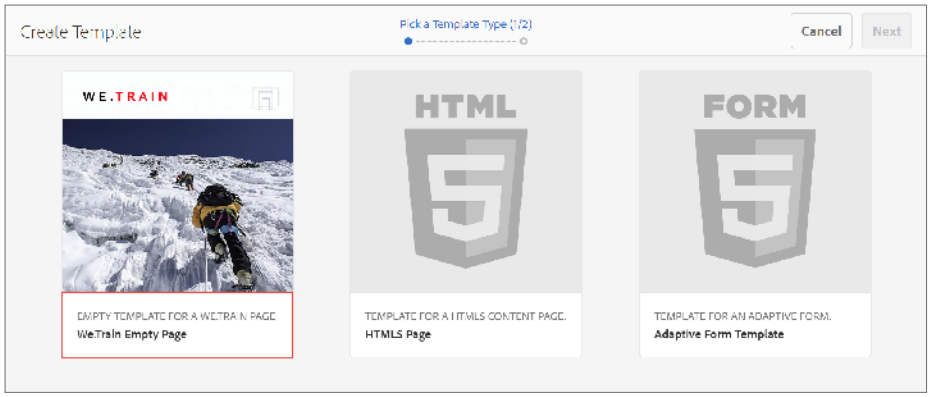
To test whether the template type is available for creating editable templates:

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. You should now see the **We.Train Empty Page** template type.



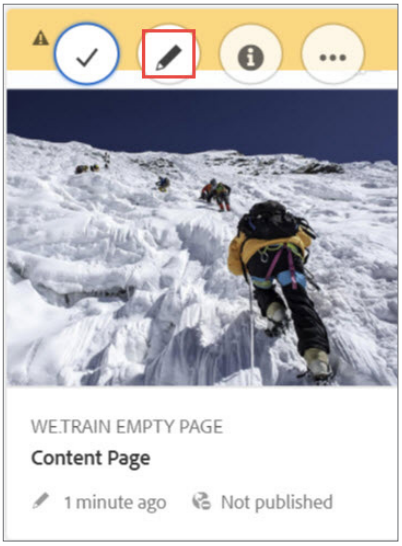
Task 2: Create a development template

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:

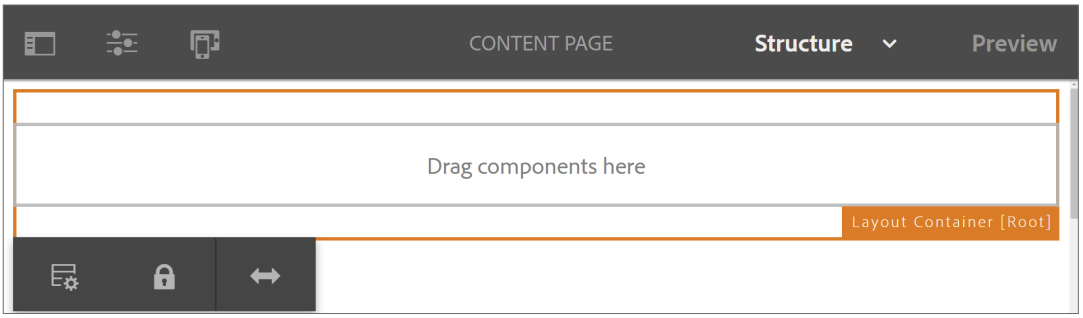
1. If you are not already in the **Create Template** wizard, navigate to **Adobe Experience Manager** > **Tools** > **Templates** > **We.Train** folder.
2. Click **Create** from the actions bar. The **Create Template** wizard opens.
3. Select the **We.Train Empty Page** template, and click **Next**.
4. In the **Template Title** field, 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 **Edit** icon, as shown. The **Content Page** template opens in a new browser tab.



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

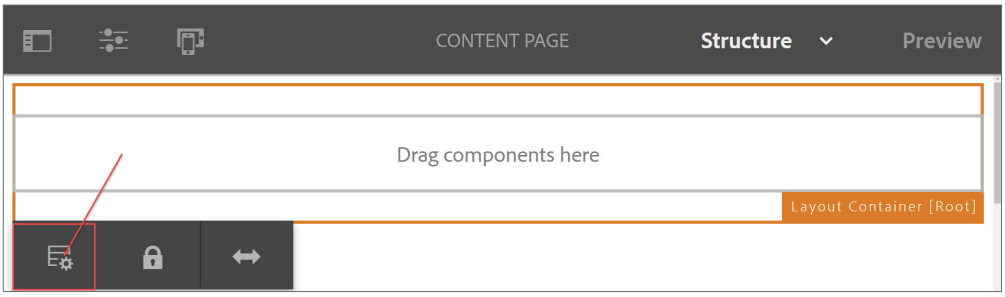
7. Click http://localhost:4502/crx/de/index.jsp. The **CRXDE Lite** page opens.
8. Navigate to the **/apps/training/components/structure/page** node.
9. Right-click **page.html** and click **Delete**. The page is deleted.
10. Click **Save All** from the actions bar.
11. In your browser, navigate to the tab where the Content Page template is open, and refresh the page. Notice that the page.html script does not appear and the **Layout Container [root]** is now visible, as shown. The Layout Container [root] helps configure the components that can be added to the template.



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

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

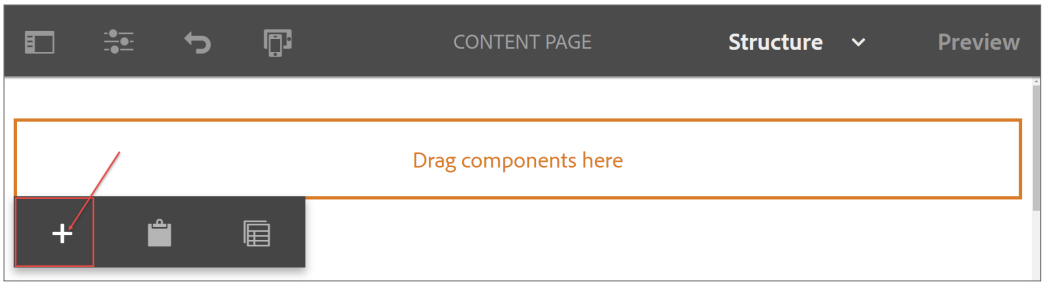
12. Select the **Layout Container [root]** and click the **Policy** icon, as shown. The **Layout Container** wizard opens.



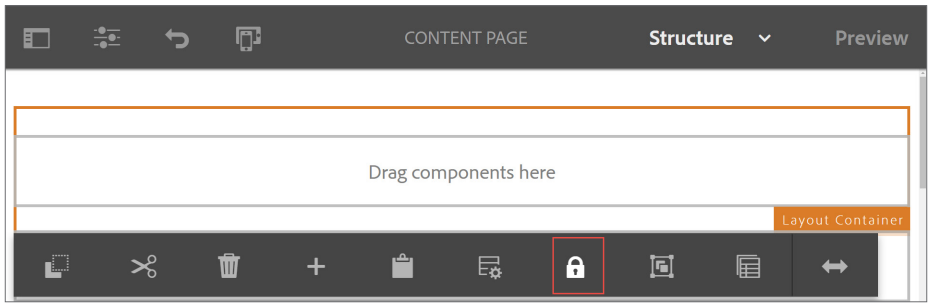
13. In the **Policy** section, in the **Policy Title** field, enter **We.Train Structure Policy**.
14. In the **Properties** section, ensure you are in the **Allowed Components** tab, and expand the **General** group. The list of components available in the group is displayed.
15. Select the **Content Fragment**, **Experience Fragment**, and **Layout Container** checkboxes.
16. Click **Done** (the checkmark icon) on the upper-right. 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 and drop onto a page.

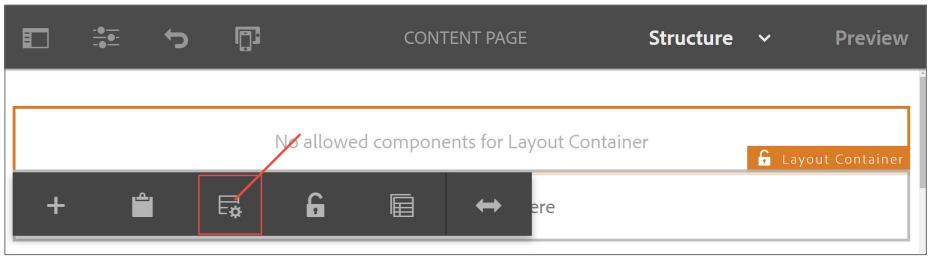
17. Refresh the page.
18. Ensure you are in the **Structure** mode, and click within the **Drag components here** layout.
19. Click the **Insert component** icon, as shown. The **Insert New Component** dialog box opens.



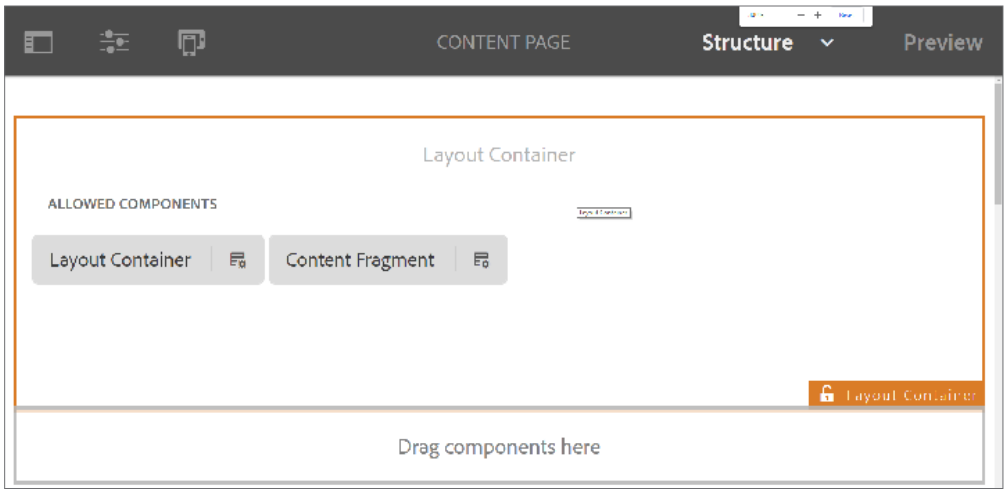
20. Select **Layout Container** from the list of components. The component is added to the template.
21. Select the inner Layout Container (not the root Layout Container) and click the **Unlock Structure Component** icon from the component toolbar, as shown. This enables you to add components to the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



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



23. In the **Policy** section, in the **Policy Title** field, enter **We.Train Content Policy**.
24. In the **Properties** section, ensure you are in the **Allowed Components** tab, and expand the **General** group. The list of components available in the group is displayed.
25. Select the **Content Fragment**, **Experience Fragment**, and **Layout Container** checkboxes.
26. Click **Done** (the checkmark icon) on the upper-right. The selected components are added to the policy.

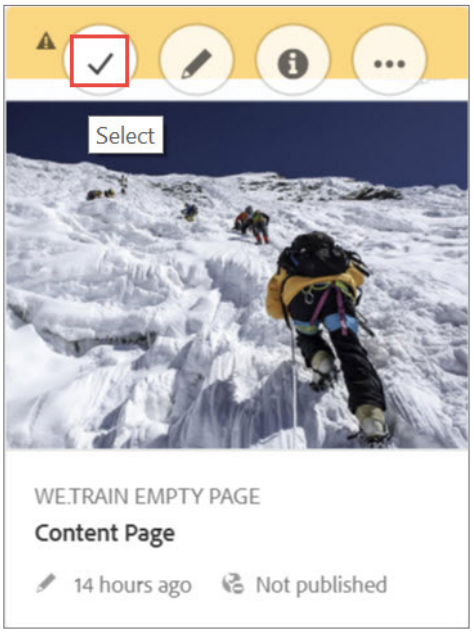


Task 3: Enable and publish the template

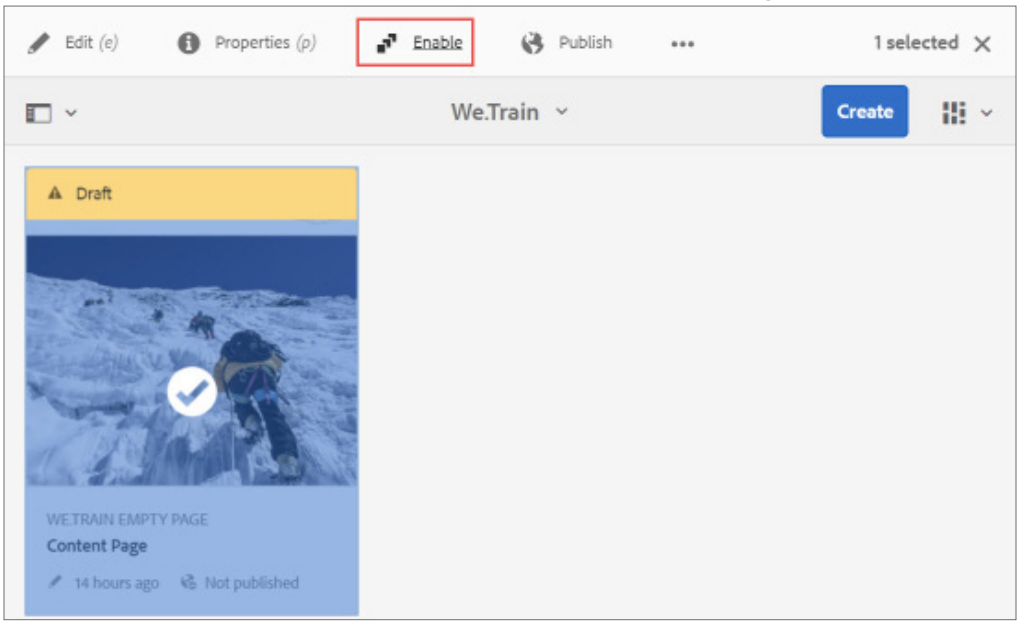
In this exercise, you will first enable the template so that it is available for authors to create pages on the author instance. Later, publish the template to make it available on the publish instance.

To enable the template:

1. In the AEM author instance, 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 checkboxes are selected and click **Publish**. The message **INFO The page has been published** appears.

Creating Pages from Editable Templates

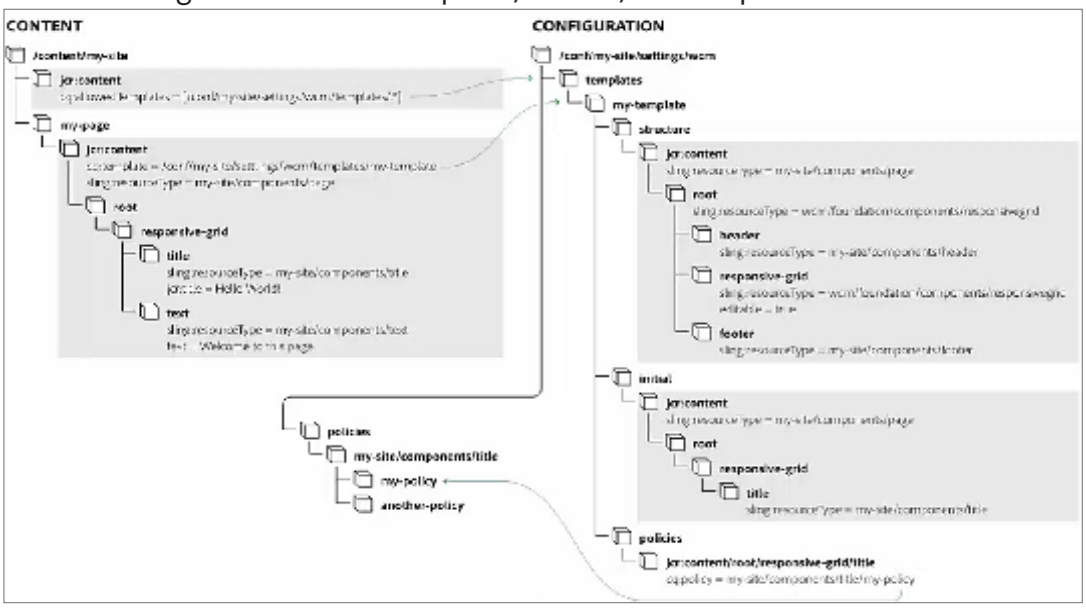
Before creating pages through the UI, a content root needs to be set up for your site. The content root will define the allowed templates for the site and is used to set other global configurations. By convention, the content root is not intended to be the home page for the site and instead will redirect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

Creating Pages

In the **Sites** console, you can create the pages of your website by using 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 the 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

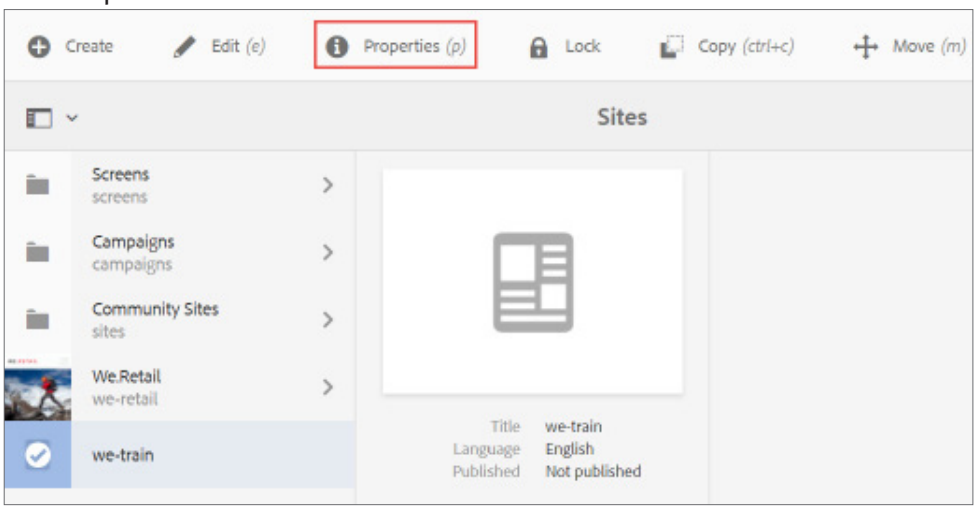
Task 1: Create a content root

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** field, 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** field, enter **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 instance, click **Adobe Experience Manager**.
17. Click the **Navigation** icon on the left panel and click **Sites**. The **Sites** console opens.
18. Click the thumbnail beside the **we-train** page and click **Properties** as shown. The **Properties** wizard opens.



19. Ensure you are in the **Basic** tab and enter **We.Train** in the **Title** field.
20. On the **Advanced** tab:
    a. In the **Redirect** field, enter **/content/we-train/en**.
    b. Scroll down and click **Add** under **Allowed Templates**.
    c. Enter **/conf/we-train/settings/wcm/templates/.*** in the **Allowed Templates** field.
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. In CRXDE lite, navigate to the **/content/we-train/jcr:content** node and observe the properties that you just added to the content root.

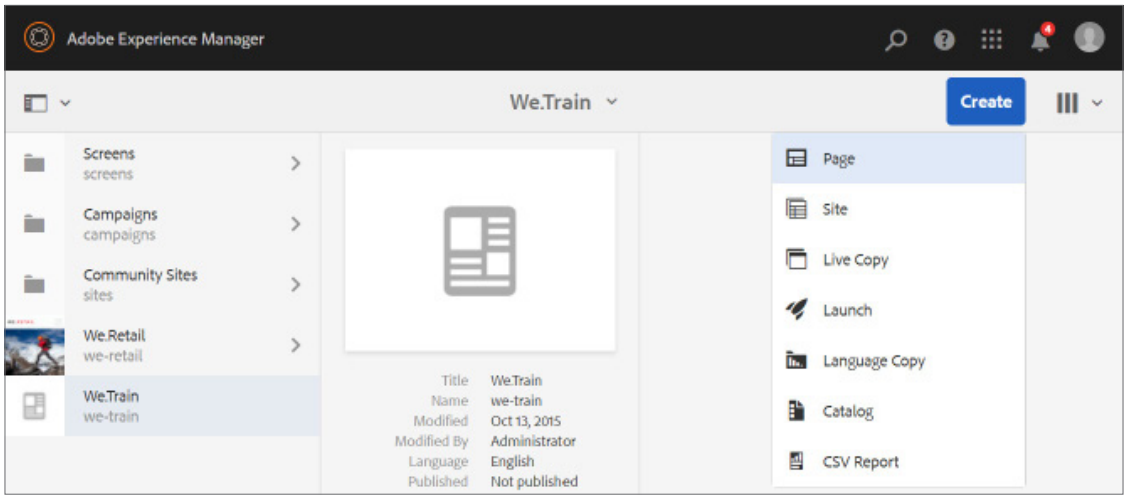
Task 2: Create a site structure

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

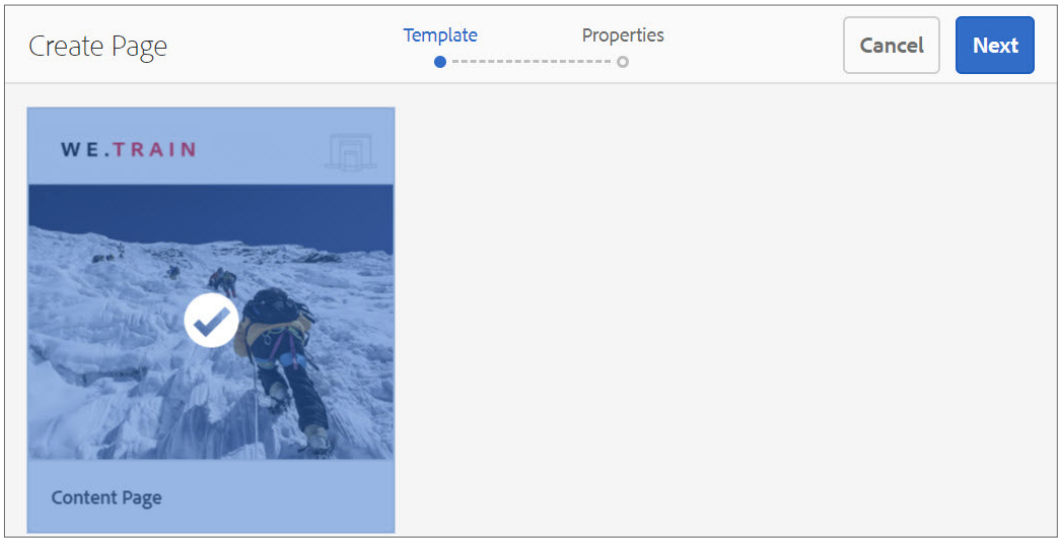
In this exercise, you will create the following website structure:
--> We.Train
--------> English
----------> About Us
----------> Experiences
----------> Equipment
----------> Activities
----------> Hiking
----------> Biking
----------> Running
----------> Skiing
----------> Climbing
--------> French

To create a page:

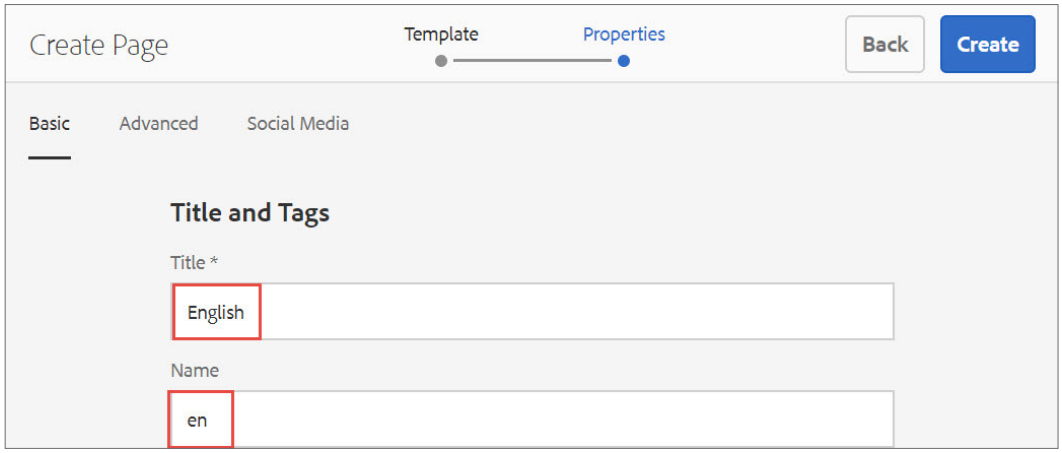
1. Click http://localhost:4502/sites.html/content. The **Sites** console page opens.
2. Select the **We.Train** page and click **Create** > **Page** from the actions bar, as shown. The **Create Page** dialog box opens.



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

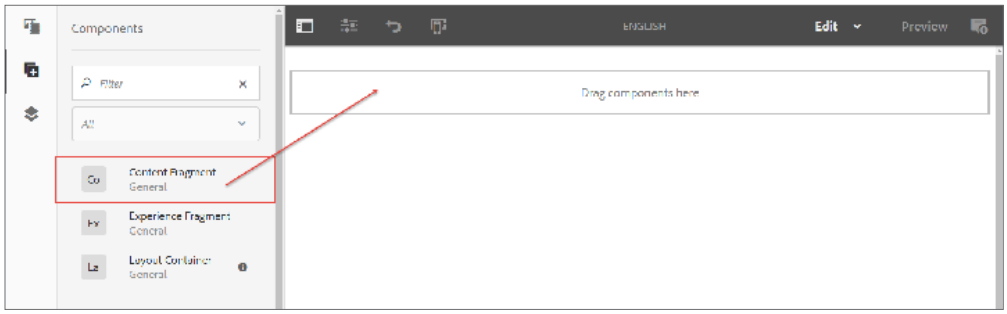


4. Enter **English** in the **Title** field and **en** in the **Name** field, and then click **Create** as shown. The **Success** dialog box appears.



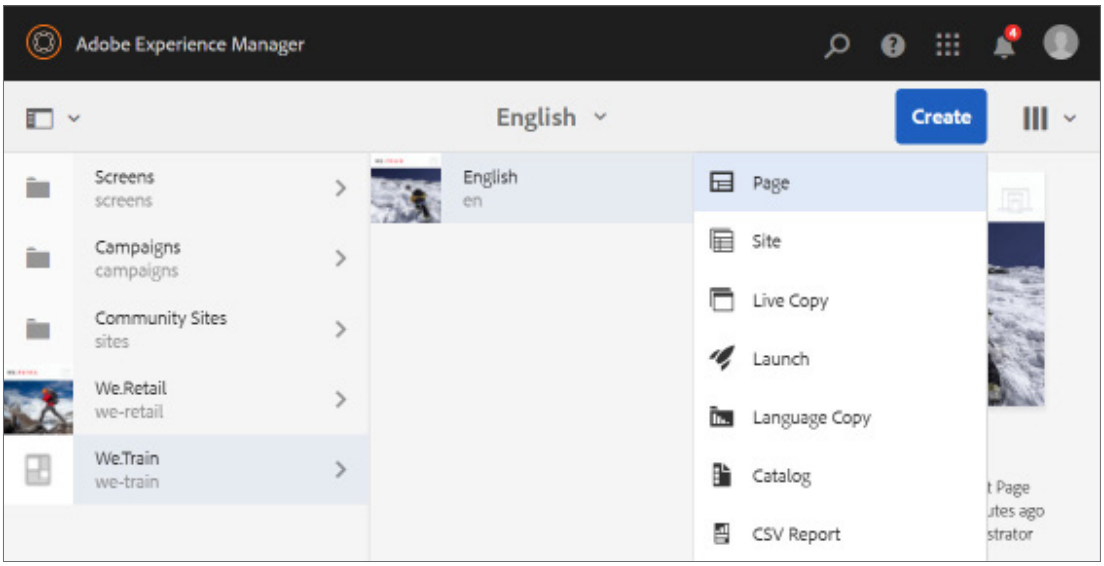
5. Click **Open**. The page opens in a new tab of your browser.
6. Ensure **Edit** is selected in the page toolbar.
7. Click the **Toggle Side Panel** icon from the page toolbar. The side panel opens.
8. Click the **Components** icon. All available components for the page are displayed in the Components 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:

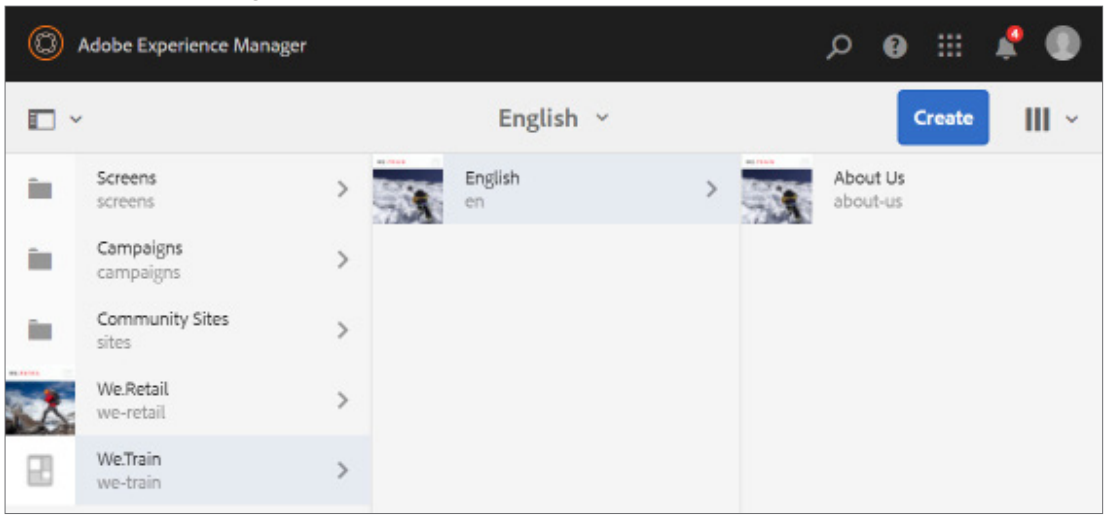
10. Ensure you are in **Sites** console
11. Navigate to the **We.Train** > **English** page.
12. Click **Create** from the actions bar and select **Page** from the drop-down menu, as shown. The **Create Page** dialog box opens.



13. Select the **Content Page** template and click **Next**. The **Properties** wizard opens.

14. Enter **About Us** in the **Title** field and **about-us** in the **Name** field, and then click **Create**. The **Success** dialog box opens.

15. Click **Done**. A subpage of **English** is created, as shown:

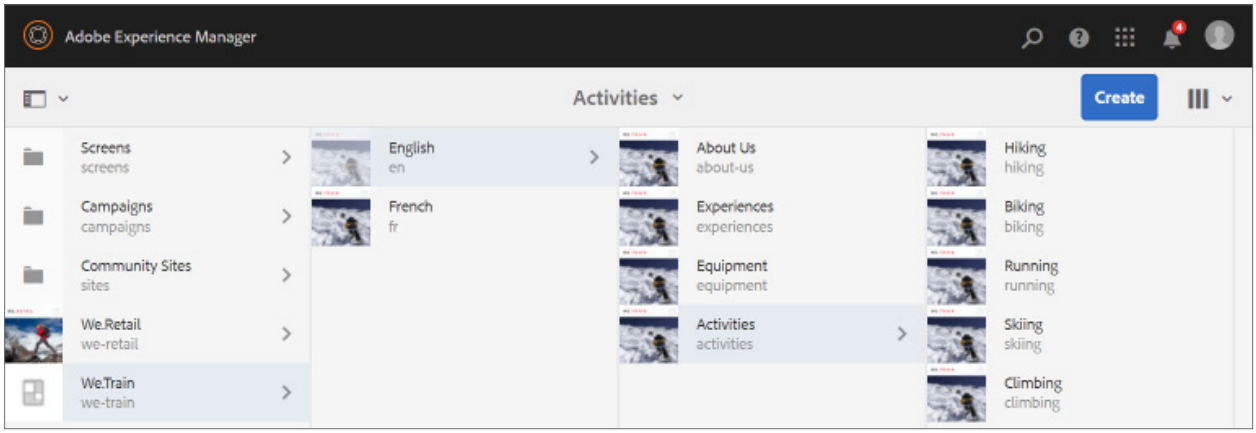


16. Perform steps 11 through 15 and create the below website structure with the following naming conventions (where, T stands for Title and N for Name):

--------> 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. Optionally, 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 client-side library to a page component
- Add a client-side library to a template

Client-Side Libraries

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

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

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

Structure of Client-Side Libraries

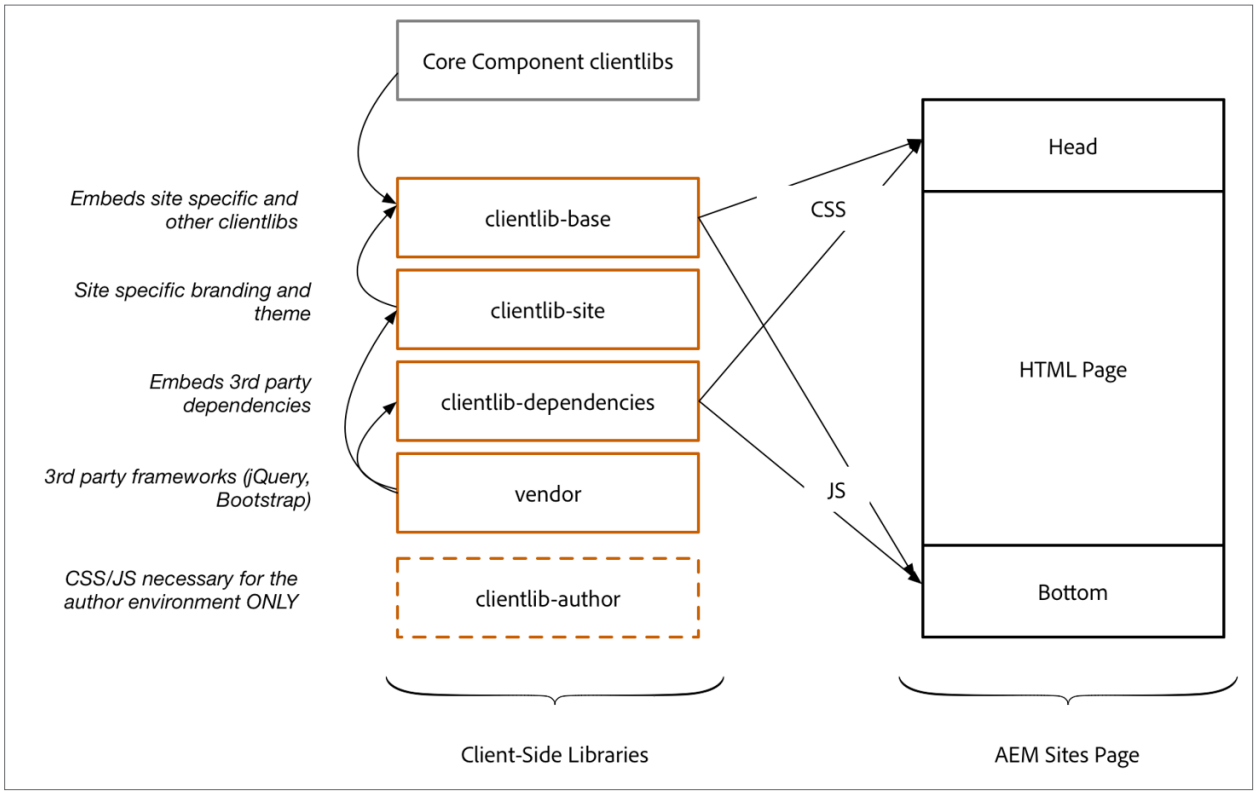
A client-side library folder is a repository node of type **cq:ClientLibraryFolder**. The typical node structure of a client library in AEM is:

```
[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 page and JS at the bottom of 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`:

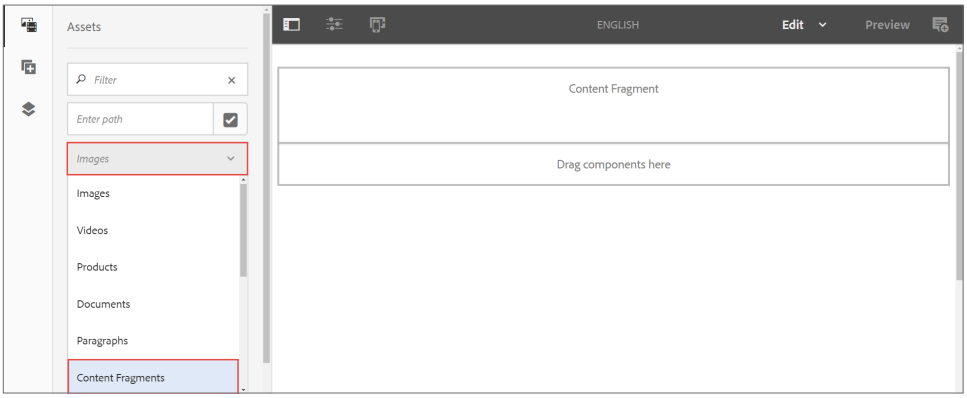
- css: Loads only the CSS files of the referenced client libraries.
- js: Loads only the JavaScript files of the referenced client libraries.
- all: Loads all files of the referenced client libraries (both CSS and JavaScript).

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.

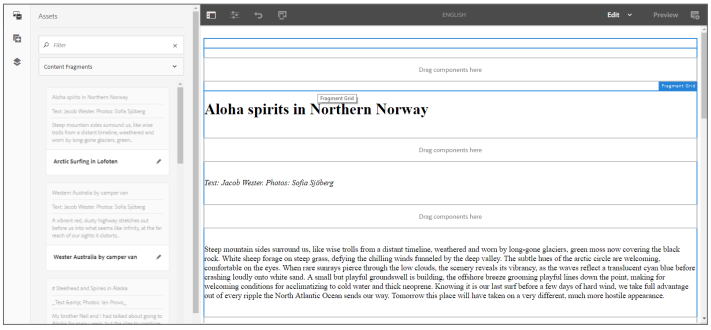
Exercise 1: Add Content to a page

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 instance, navigate to **Sites** > **We.Train** > **English** page.
2. Click the **English** page thumbnail, and click **Edit (*e*)** on the header bar to open the English page.
3. In the left panel, click the **Components** icon. The available components are displayed.
4. Drag and drop the **Content Fragment** component onto the layout container.
5. In the left panel, click the **Assets** icon. The available assets are displayed.
6. Filter the assets based on Content Fragments by clicking the drop-down menu and selecting **Content Fragments**, as shown:



7. Drag and drop the **Arctic Surfing in Lofoten** fragment onto the **Content Fragment** component. Notice how the content has no design currently.



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** field, 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 folder is created.
9. Click **Save All** from the actions bar.

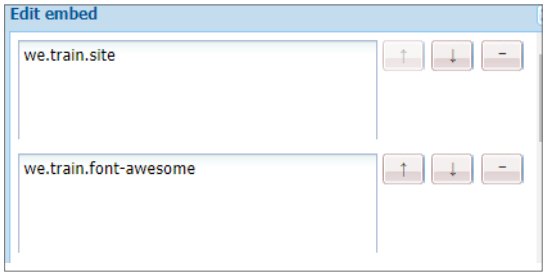
10. Select the **clientlib-base** folder, and add the following properties:

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

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

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

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

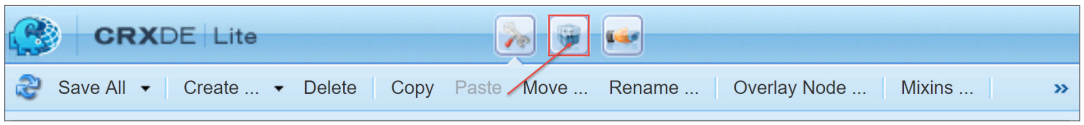


12. Click **Save All** from the actions bar.
13. Right-click the **clientlib-base** node, and click **Create** > **Create File**. The **Create File** dialog box opens.
14. Enter **css.txt** in the **Name** field, and click **OK**. The file is created under the clientlibs folder.
15. Click **Save All**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

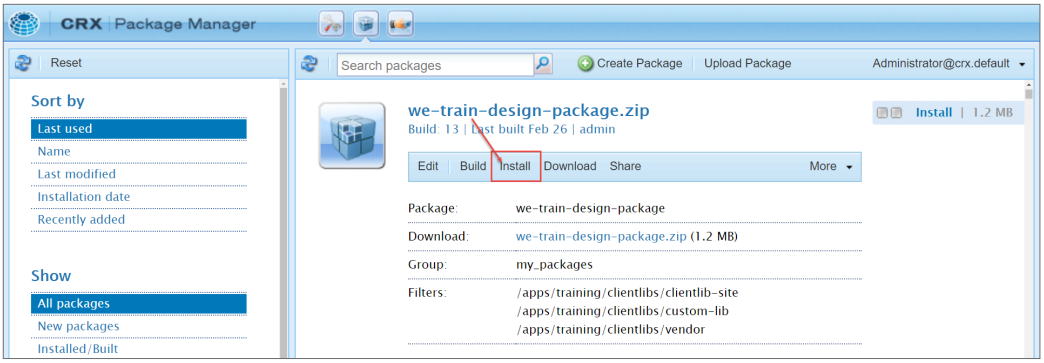
Next, you will upload the **we-train-design-package** to the JCR. This package contains the css/js that you will use in the We.Train site structure.

To install the we-train-design-package:

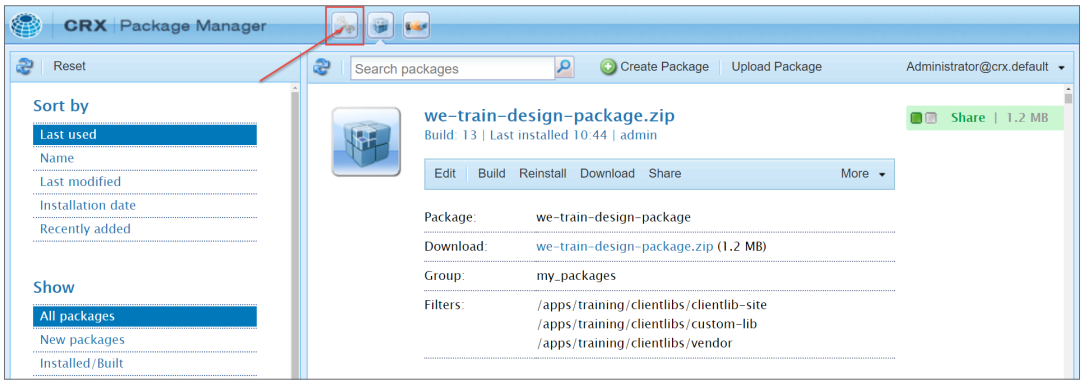
17. From CRXDE Lite, click the **Package** icon from the header bar, as shown. The **Package Manager** page opens.



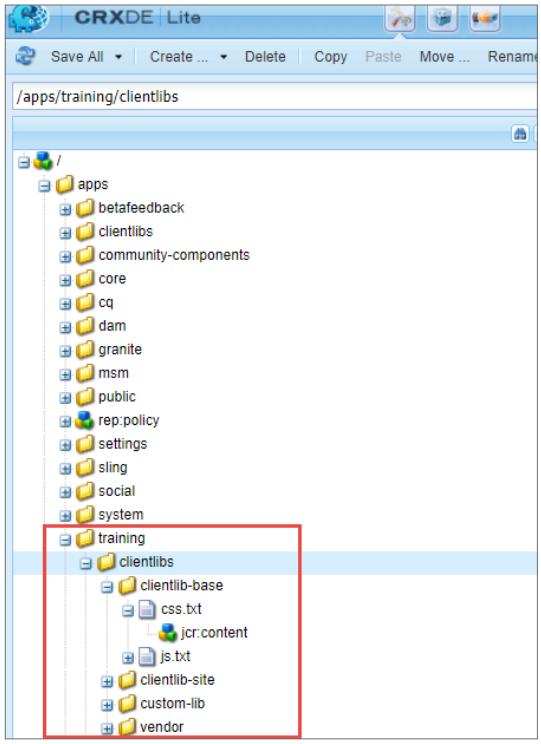
18. Click **Upload Package** from the actions bar. The **Upload Package** dialog box opens.
19. Click **Browse**.
20. Navigate to the **Creating Client-side Libraries in Adobe Experience Manager** exercise folder provided to you and double-click the **we-train-design-package.zip** package.
21. Click **OK** in the **Upload Package** dialog box. The package is uploaded.
22. 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.



23. Ignore the **Advanced Settings** and click **Install**. The package is installed on your instance.
24. Click the **Develop** icon from the header bar, as shown. The CRXDE Lite page is refreshed.



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



Exercise 3: Add a client-side library to a page component

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

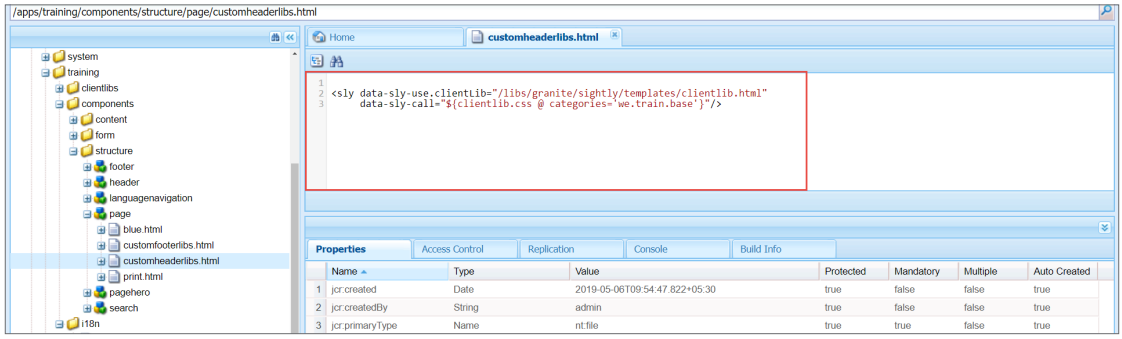
1. Click http://localhost:4502/crx/de/index.jsp if you are not already in CRXDE Lite. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/clientlibs/clientlib-base** node. Observe the property **categories=we.train.base**. You will use this category name when adding the clientlib to a 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. Navigate to the **Creating Client-Side Libaries in Adobe Experience Manager** exercise files, open **customheaderlibs.html** in Notepad++, copy the script, and paste it into **customheaderlibs.html** in CRXDE Lite, as shown:



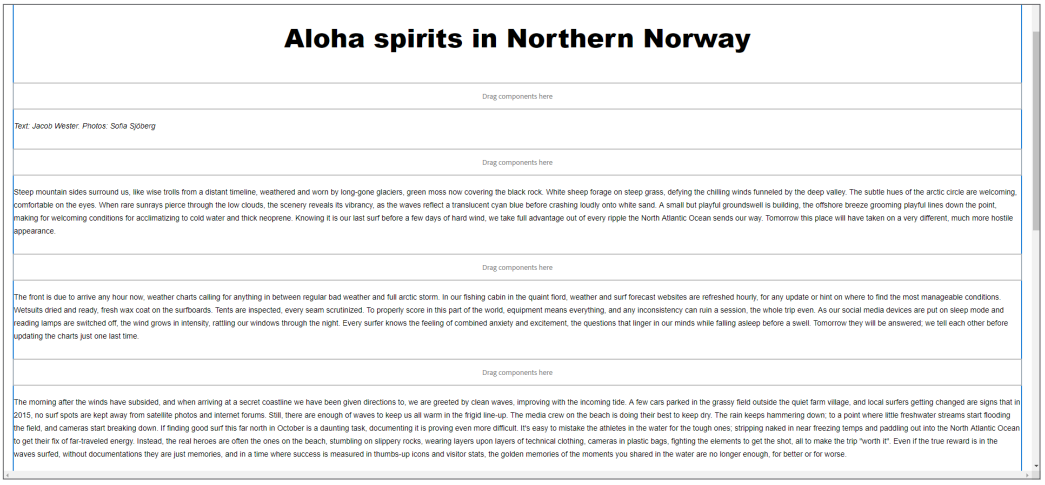
10. Click **Save All**.

11. Repeat step 9, but copy-paste the script from **customfooterlibs.html** to CRXDE Lite.
12. 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.

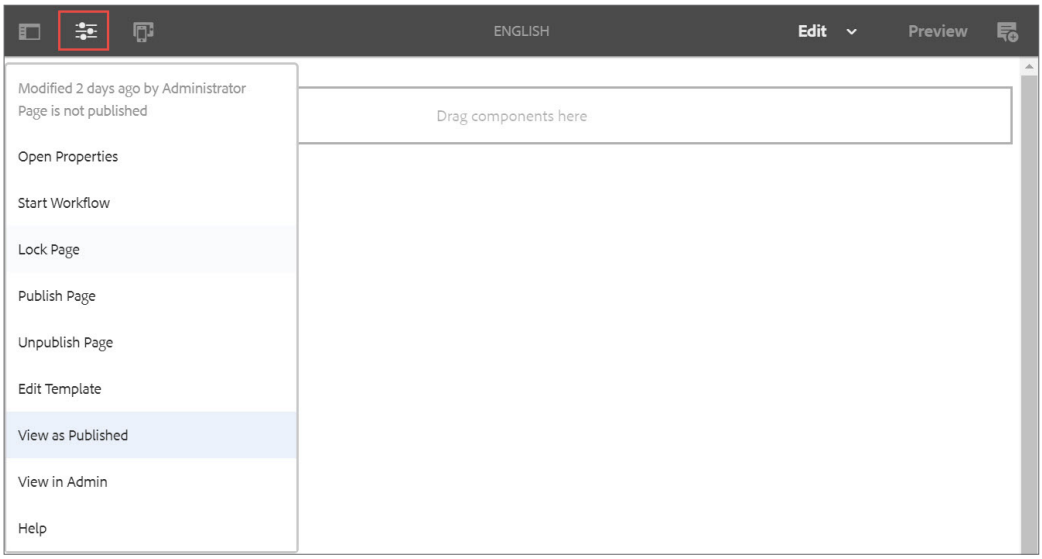
13. Click **CRXDE Lite** from the header bar. You will be taken to the AEM **Navigation** page.
14. Click **Sites** > **We.Train** site.
15. Click the thumbnail on the **English** page, and then click **Edit (e)** from the actions bar. The English page opens in a new tab.

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

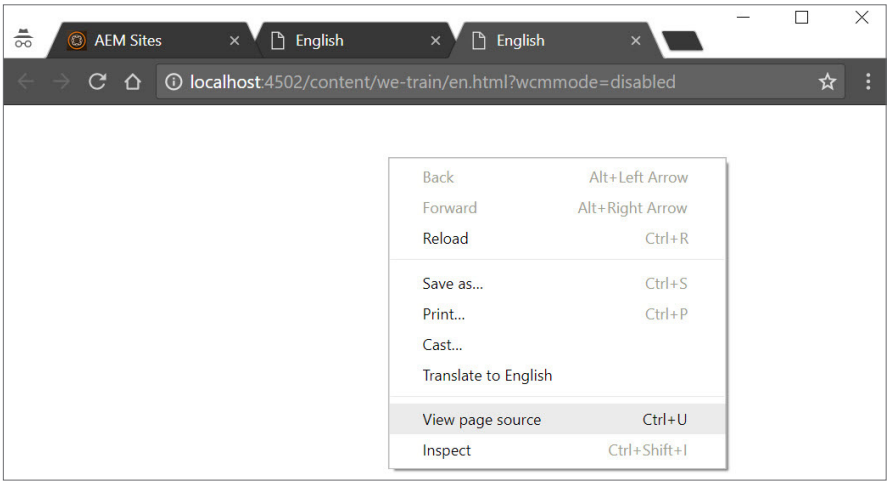


To verify that the client libraries are being loaded:

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



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



18. 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 that confirms the customheaderlibs.html is referenced in the page, as shown:

```
<!DOCTYPE HTML>
<html lang="en">
    <head>
    <meta charset="UTF-8"/>
    <title>English</title>


    <meta name="template" content="content-page"/>




<link rel="stylesheet" href="/etc.clientlibs/training/clientlibs/clientlib-base.css" type="text/css">
```

19. 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 that confirms the customfooterlibs.html referenced in the page, as shown:

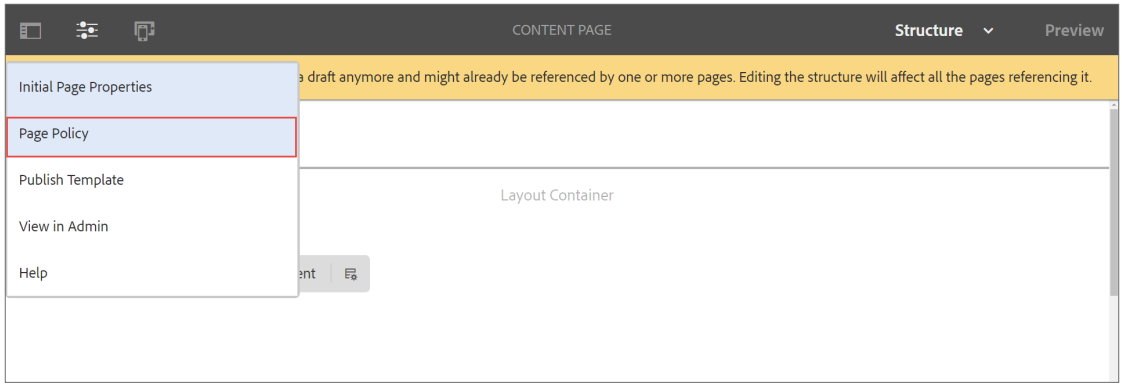
```
<script type="text/javascript" src="/etc.clientlibs/training/clientlibs/clientlib-base.js"></script>
```

Exercise 4: Add a client-side library to a template

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

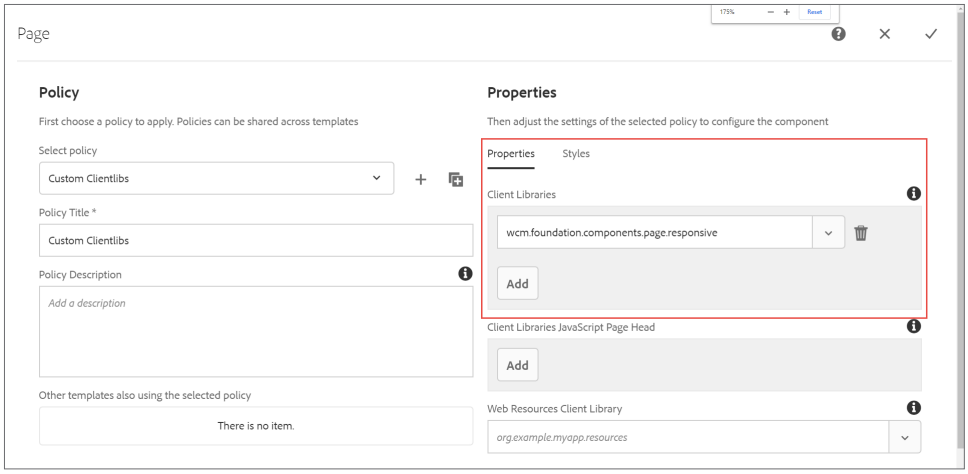
To add a client library to a template:

1. Navigate to the **Sites** > **We.Train** > **English** page in the AEM author instance.
2. Open the English page by clicking **Edit (*e*)** in the actions bar, click the **Page Information** icon from the page toolbar, and select **Edit Template** from the drop-down menu. The template opens on a new tab.
3. Ensure you are in the content page template, click the **Page Information** icon from the toolbar, and select **Page Policy** from the drop-down menu, as shown. The **Page Content Policy** opens.



4. In the **Policy** section, enter **Custom Clientlibs** in the **Policy Title** field.

5. In the **Properties** section, notice that the clientlib, **wcm.foundation.components.page.responsive**, is already added to the template, as shown. It is the default AEM responsive grid. Notice that you can also add new client libraries to the page template.



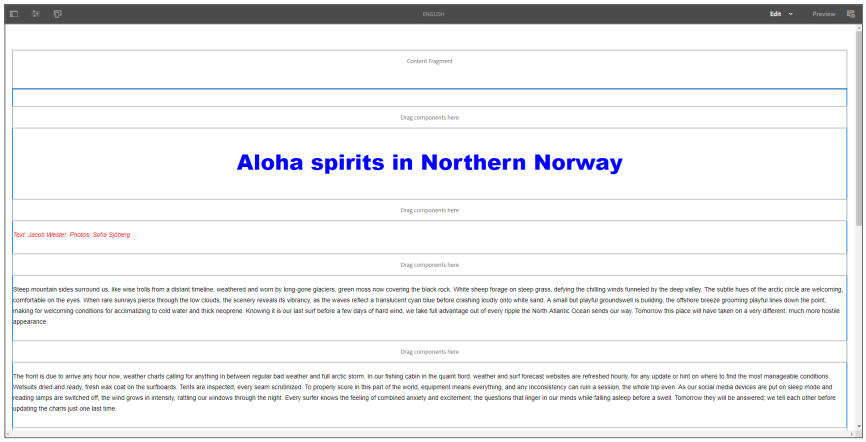
To add the custom clientlib to the template:

6. Click **Add**, and enter **we.train.custom.lib** in the field.
7. Click **Done** (checkmark icon in the upper right). The custom clientlib is added to the template.

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

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

8. Navigate to the **We.Train** > **English** page, and refresh the browser. In the content fragment you added earlier, you should now see *h1* tags in blue and *p* tags in red. This is the result of the client library being added to the template.



9. You can verify that the client library was added by checking the source of the page. Click the **Page Information** icon from the page toolbar, and click **View as Published**. The page opens in a new browser tab.
10. Right-click in the **English** page, and click **View page source.** The source code opens in a new browser tab.
11. Look for the line, `<link rel="stylesheet" href="/etc.clientlibs/training/clientlibs/custom-lib.css" type="text/css">` in the source code. Notice how the **href** tag points to the clientlibs path in JCR that confirms the custom clientlibs is added to the page, as shown. Click the links to examine its content.

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 sub-elements such as Granite UI components for the touch UI and widgets
- Can be developed using HTML Template Language (HTL) (recommended) or Java Server Pages (JSP)

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 the 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 (as of April 2019):

Core Component	Description
Page	Responsive page working with the template editor.
Breadcrumb	Page hierarchy navigation.
Carousel	Displays content panels on a page and helps create slides of varying component type.
Content Fragment	Adds a content fragment asset its elements and its variations to a page
Image	Helps add a single image asset to the page. Images are adaptive, and the relevant image width is selected for the screen size and lazy loading is available.
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.
Separator	Displays a horizontal line on the page for dividing sections of content.
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.
Teaser	Helps group an image, title, and description for promoting and linking to site content sections. One or more actions can also be defined.
Text	Displays rich text paragraph on the page and provides various formatting options.
Title	Creates heading for a page or page sections.
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.
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.

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:

- **Pre-configurable**: 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, provide ARIA labels, and support 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 refers 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 simply a node 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.

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

8. 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 2-8 to add the following components:

/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

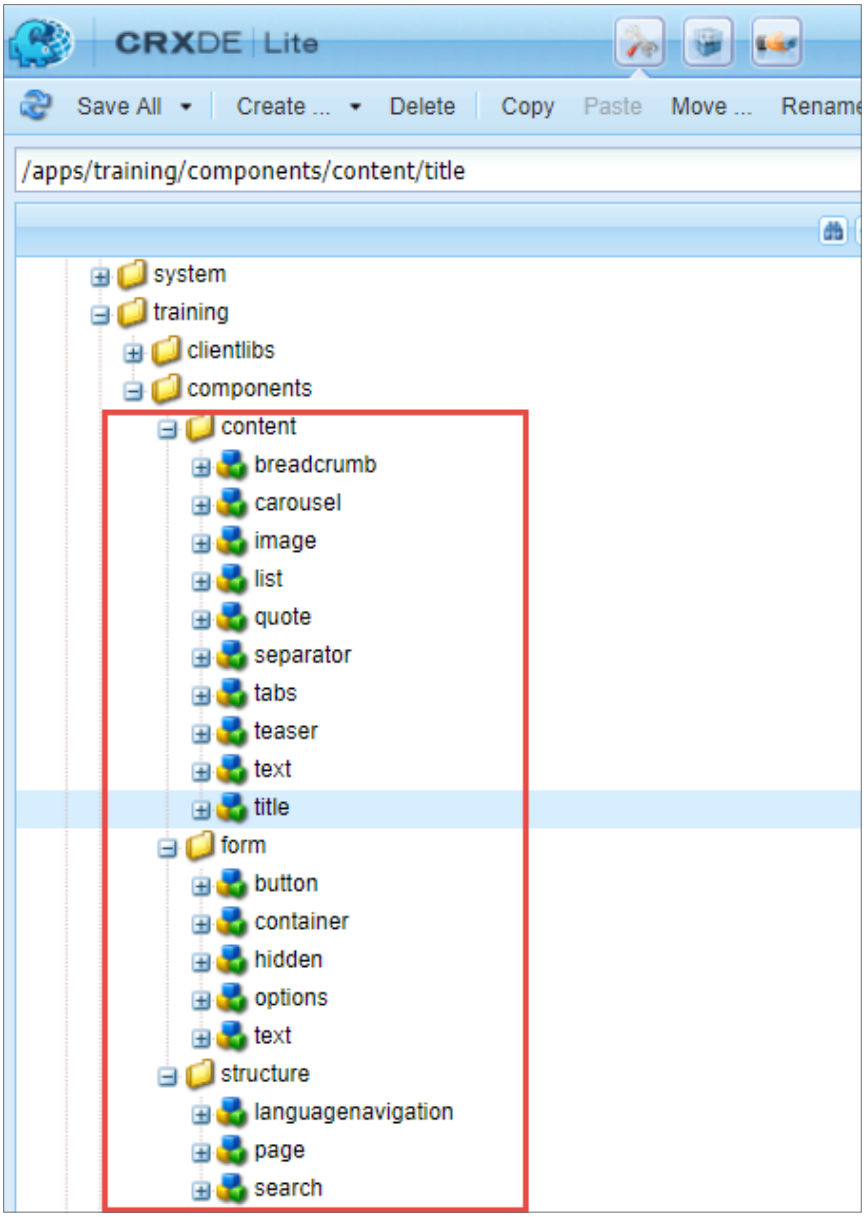
The Caret (^) symbol next to some of the components indicates that the components have additional node structures that are created with the content package.

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

9. To install the We.Train proxy components, click **http://localhost:4502/crx/packmgr/index.jsp**. The **CRXDE Package Manager** opens.
10. Click **Upload Package** from the actions bar. The **Upload Package** dialog box opens.
11. Click **Browse**.
12. Navigate to the **Core Components: An Overview** exercise folder provided to you and double-click the **we-train-proxy-components.zip** package.
13. Click **OK** in the **Upload Package** dialog box. The package is uploaded.

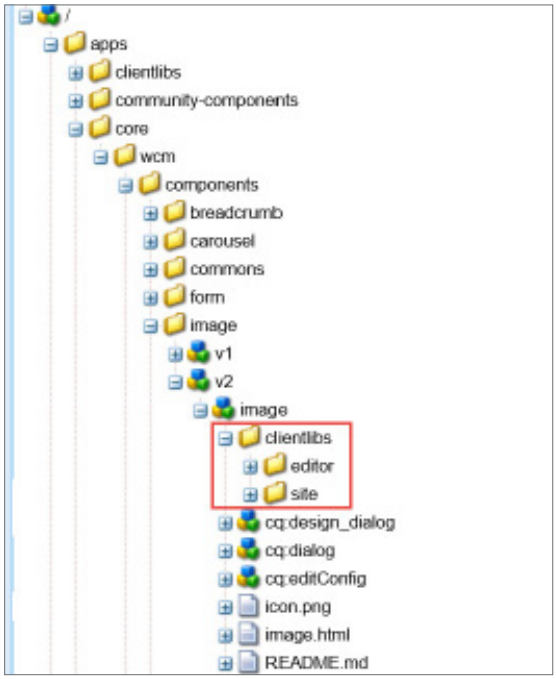
14. In the **we-train-proxy-components.zip** section, click **Install**. The **Install Package** dialog box opens.
15. Ignore the **Advanced Settings**, and click **Install**. The package is installed on your instance.
16. In CRXDE Lite, view the proxy components under:

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

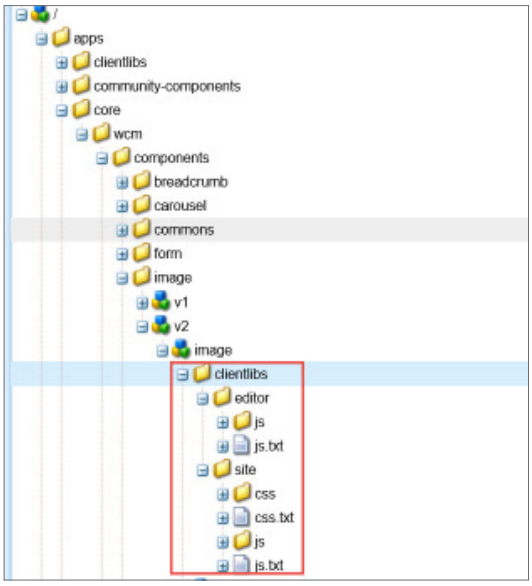


Client Libraries for Core Components

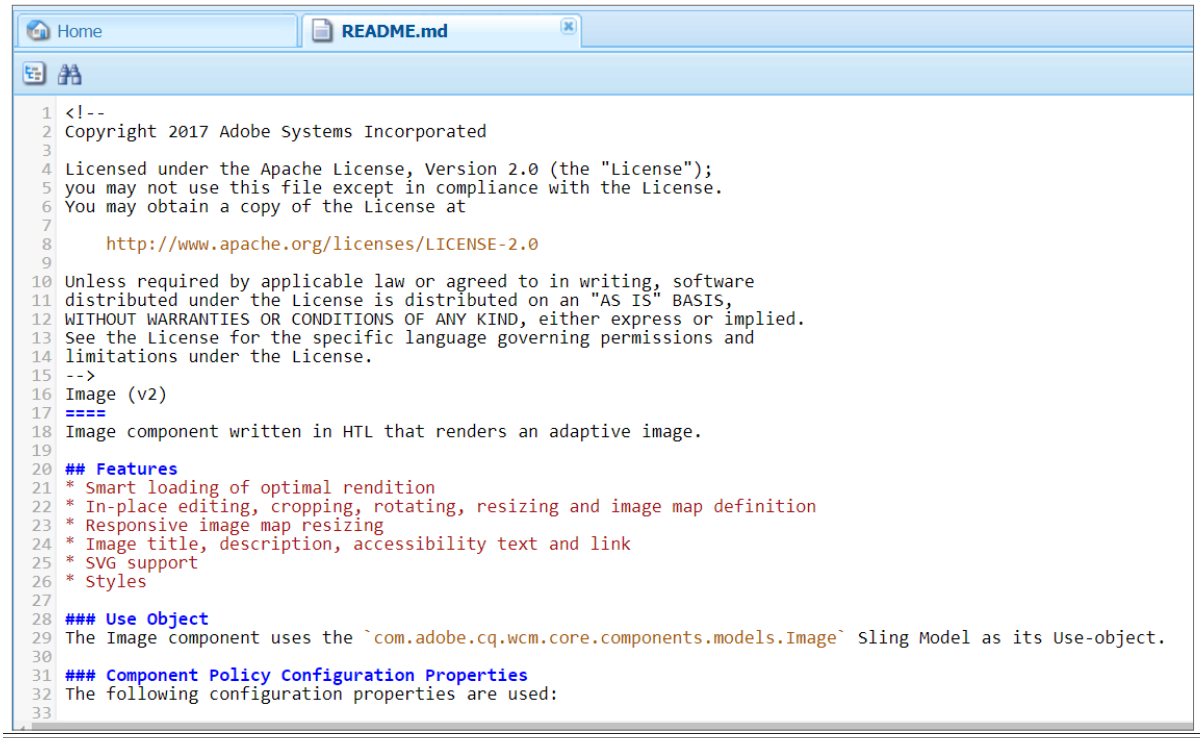
Several Core components have client libraries that must be included on the page for the component to function properly. For example, under `/apps/core/wcm/components/image/v2/image/clientlibs` notice, there are two client libraries—**editor** and **site**, as shown in the below screenshot:



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.

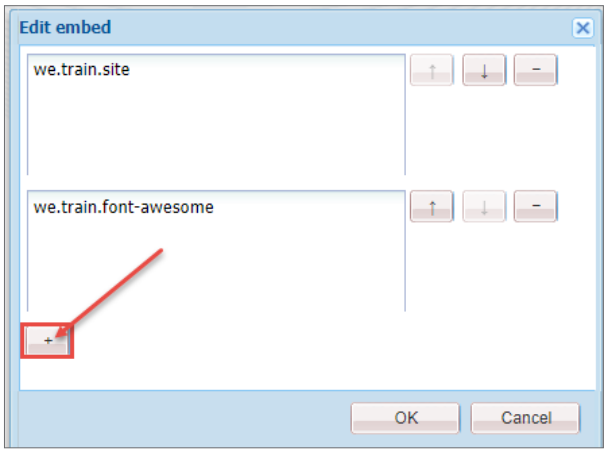


Note: Refer to the following link for more information on client libraries:
Update clientlib-base

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. 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

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

Content Policies

The content policies define the design properties of a component, such as the available components or dimensions. These policies are applicable to the template and the pages created with the template. Template authors can define and update the content policies in the template editor when creating the editable templates.

The property `cq:policy`, on the root node provides a relative reference to the content policy for the page's paragraph system:

`/conf/<your-folder>/settings/wcm/templates/<your-template>/policies/jcr:content/root`

The property `cq:policy`, on the component-explicit nodes under root, provides links to the policies for the individual components.

The actual policy definitions are stored under:

`/conf/<your-folder>/settings/wcm/policies/wcm/foundation/components`

Note: Storing policies under a general location helps the reusability of content policies across different templates.

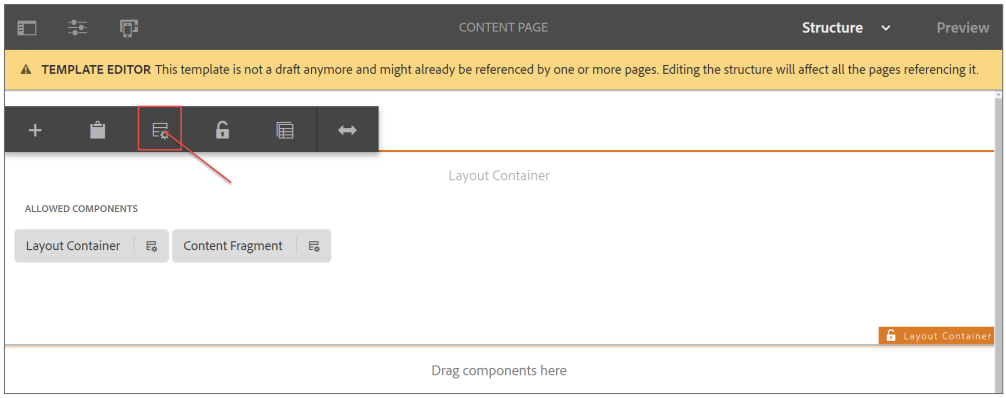
A common use case for content policies is when you want to use the same components across all templates. Content policies help set up allowed components for the **Layout Container** component on one template and then reuse that policy for all other templates that should use the same component set. If the policy ever needs to be updated with more components, you can modify the allowed components once and all templates will reflect the change.

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 in the page through the Layout Container's content policy.

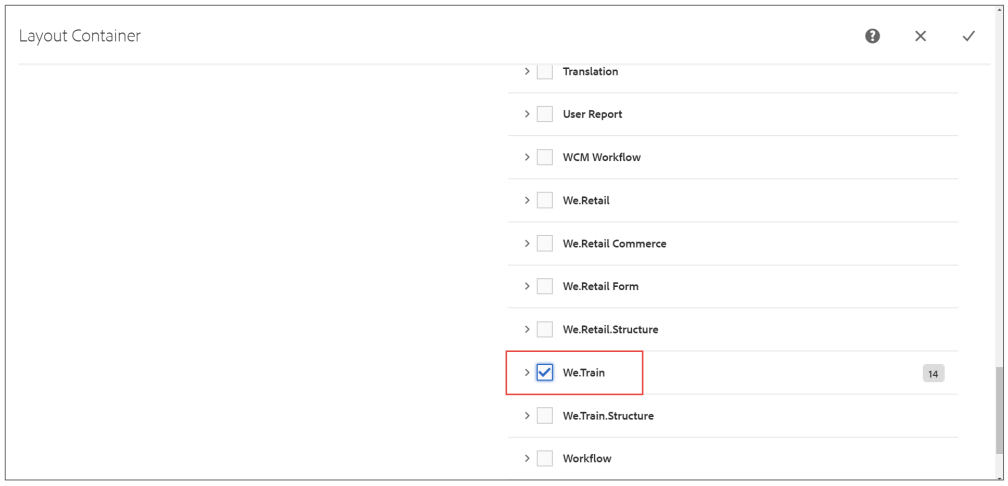
TO add proxy components to the template:

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, and select the **Content Page** template by clicking the **Select** icon (checkmark). The template is selected.
4. Click **Edit (*e*)** from the actions bar. The template opens in a browser tab.
5. Select the inner **Layout Container** and click the **Policy** icon from the component toolbar, as shown. The **Layout Container** wizard opens.

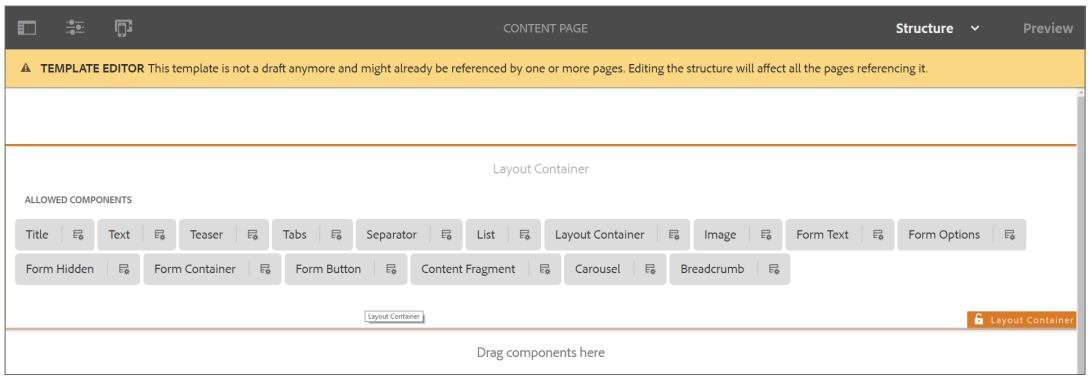


6. In the **Policy** section, ensure **We.Train Content Policy** is selected. You created this policy when you created the template earlier.

7. In the **Properties** section, scroll down and select the **We.Train** group checkbox, as shown:



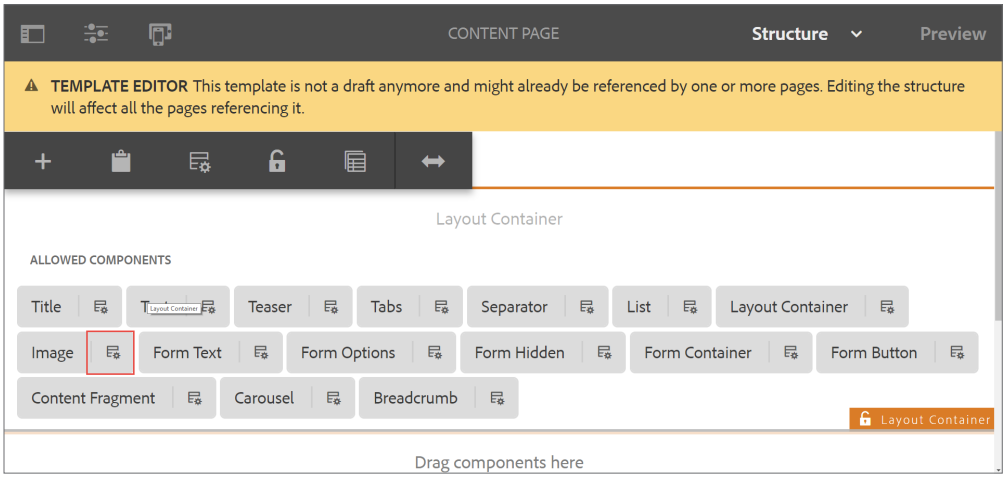
8. Click **Done** (checkmark icon in the upper right). This will add the newly created proxy components to the Layout Container from the We.Train group. Any new components with the **componentGroup=We.Train** property, will be added to the policy automatically.



Exercise 4: Add content policies to proxy components

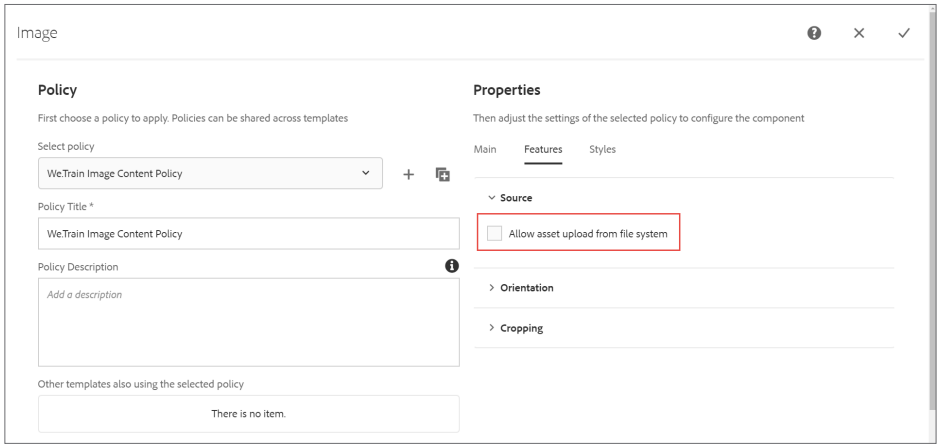
Now that you have proxy components for the Core components, you can make use of the built-in features of the proxy components. In this exercise, you will set up a content policy for the image component that will restrict uploading images from an author's local folder. Authors can only use images from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

1. Navigate to **Tools** > **Templates** > **We.Train** > **Content Page** in the AEM author instance.
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. In the **Policy** section, type **We.Train Image Content Policy** in the **Policy Title** field.
5. In the **Properties** section, ensure you are on the **Main** tab.
6. Under **Widths**, click **Add**, and enter **1024**.
7. In the **JPEG Quality** field, enter **100**.

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



9. Click **Done** (checkmark) to save changes. Later, when you test the image component, notice how you can only drag and drop images from the DAM onto the image component and cannot upload images from your local file system.

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

Note: The link to the AEM GEMS session is:
https://helpx.adobe.com/experience-manager/kt/eseminars/gems/AEM-Core-Components.html

Note: The link to the WKND tutorial is:
https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/getting-started.html

Note: To keep up-to-date on the latest changes to the Core components, watch/follow the Core components repository on GitHub:
https://github.com/adobe/aem-core-wcm-components

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.

Pre configuring 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 can be enabled or disabled with a simple click.

As the Core components can be pre-configured 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 to test the authoring capabilities.

Exercise 5: 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.

This exercise includes the following tasks:

1. Create a Page
2. Add Core components

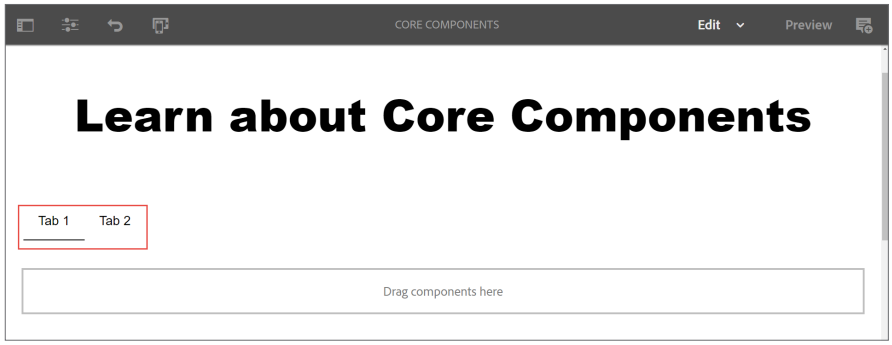
Task 1: Create a page

1. In your AEM author instance, 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 Properties** wizard opens.
4. In the **Title** field, type **Core Components**, and click **Create**. The **Success** dialog box is displayed.
5. Click **Open** to open the page.

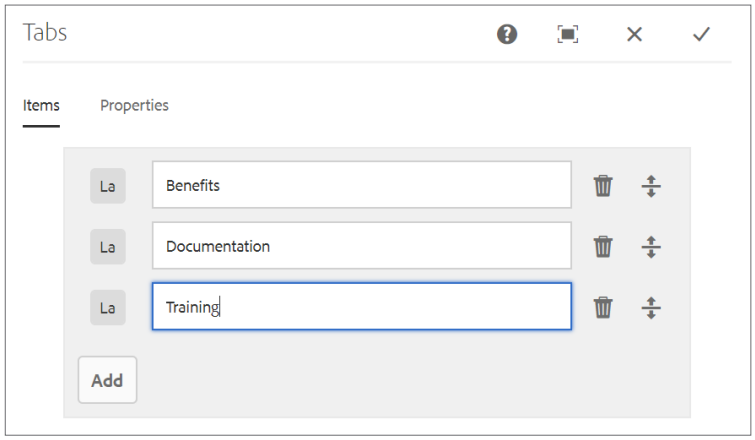
Task 2: Add core components

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 in the left panel. The list of available components is displayed.
3. Drag the **Title** component onto the **Drag components here** area.
4. Double-click the **Title** component you added to edit it. The **Title dialog** box opens.
5. In the **Title** field, type **Learn about Core Components**, and click the **Done** (checkmark) icon in the upper right. The title is updated.

6. From the Components browser in the left panel, drag the **Tabs** component onto the **Drag components here** area. Two tabs, Tab 1 and Tab 2, are added, as shown:

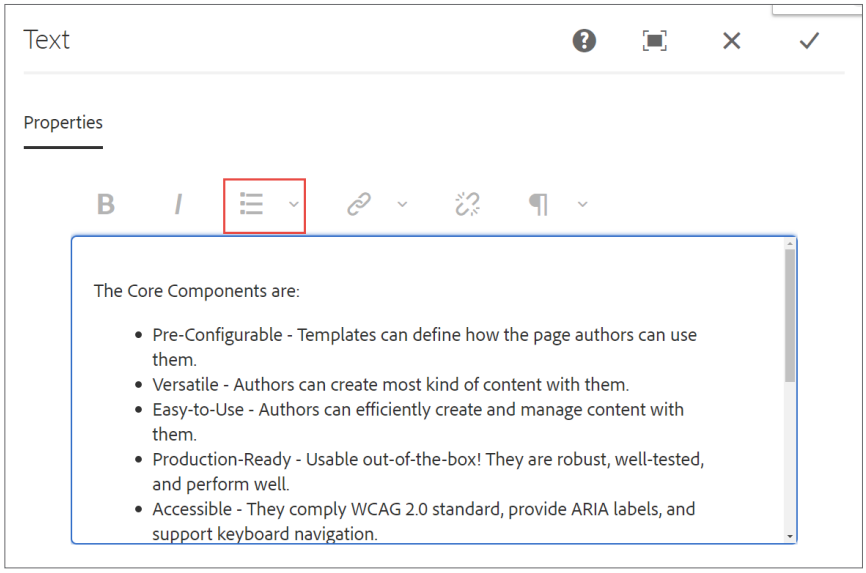


7. Double-click the tabs component. The **Tabs** dialog box opens.
8. On the default **Items** tab, rename **Tab 1** as **Benefits**.
9. Rename **Tab 2** as **Documentation**.
10. Click **Add**. The **Insert New Component** dialog box opens.
11. Click **Layout Container**. A new tab is added.
12. Name the new tab, **Training**. The **Tabs** dialog box should look, as shown:



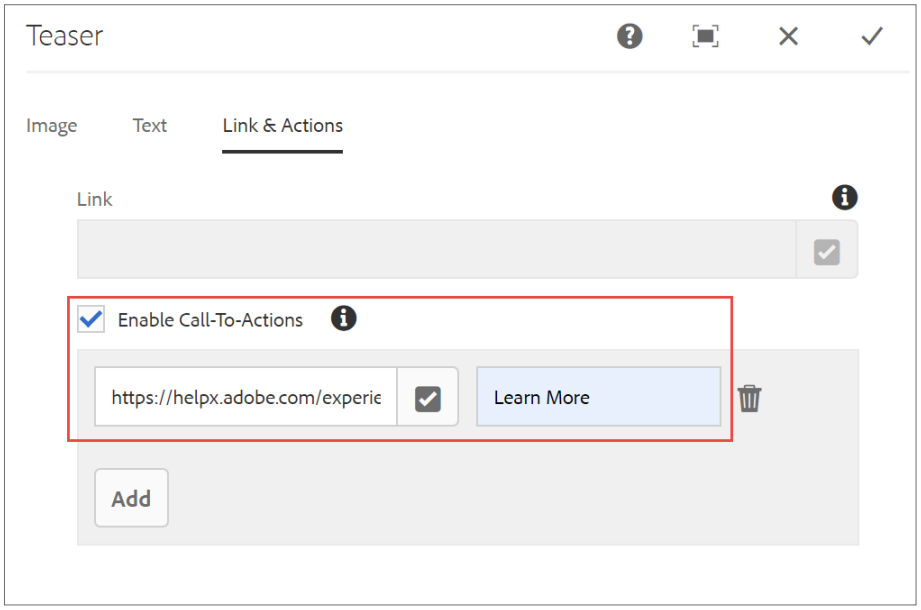
13. Click **Done** (checkmark in the upper right) to save the changes. The three tabs are added to the page.
14. On the **Benefits** tab, click the **Drag components here** area.
15. Click the **Insert component** icon (the + icon) and add a **Text** component.

16. Navigate to the **Core Components An Overview** exercise file provided to you and copy the content from **Benefits.txt**.
17. On the **AEM Components** page, double-click the **Text** component. The **Text** dialog box opens.
18. Paste the content and apply the bullet list format to the text, as shown:

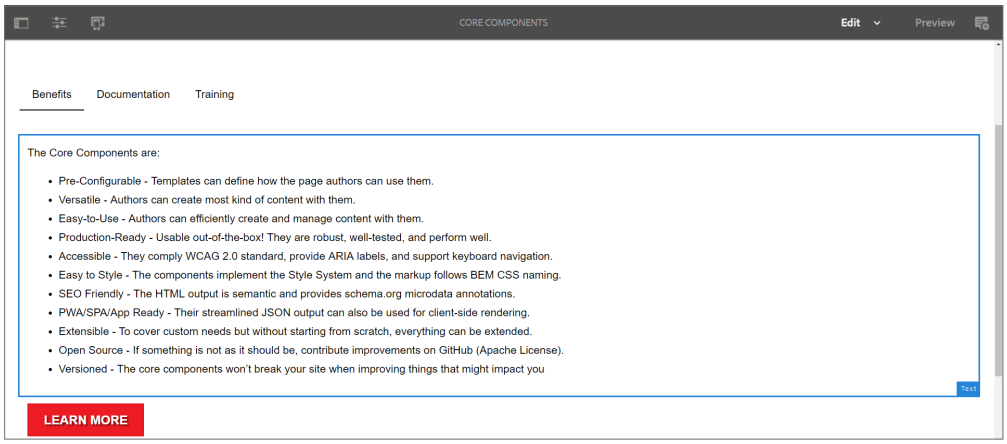


19. Click **Done** to save the changes. The text is added to the **Benefits** tab.
20. Click the **Drag components here** area below the text you added in the previous step.
21. Click the **Insert component** icon (the + icon), and add a **Teaser** component.
22. Click the **Teaser** component, and click the **Configure** icon (the wrench icon). The **Teaser** dialog box opens.
23. On the **Link & Actions** tab, select the **Enable Call-to-Actions** checkbox.
24. In the **Link** field, type **https://helpx.adobe.com/experience-manager/core-components/using/introduction.html**.

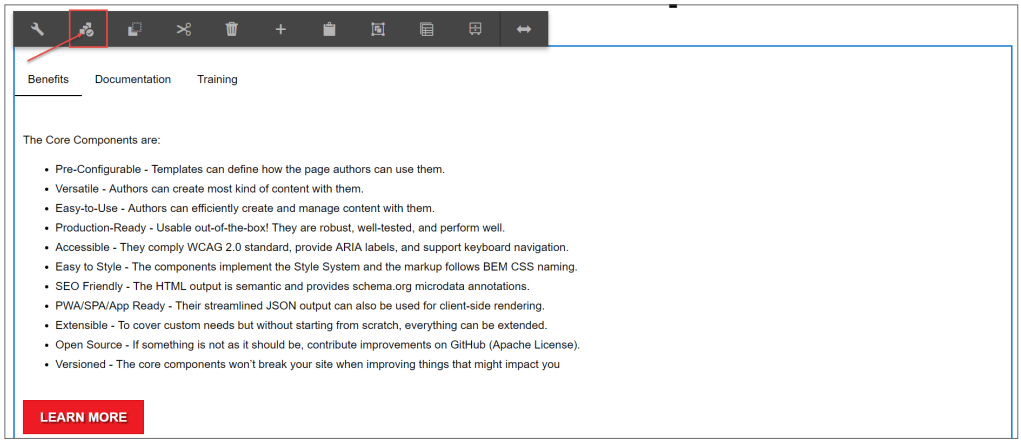
25. In the **Text** field, type **Learn More**. The **Teaser** dialog should look, as shown:



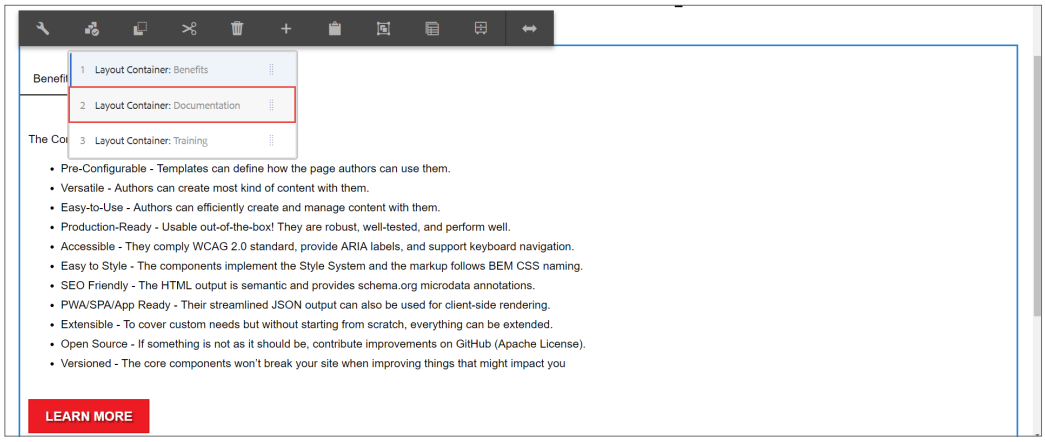
26. Click **Done** (checkmark icon) to save the changes. The **Learn More** button is added to the page. The Benefits tab should look, as shown:



27. To add components and content on the **Documentation** tab, click the **Tabs** component and click **Select Panel**, as shown:



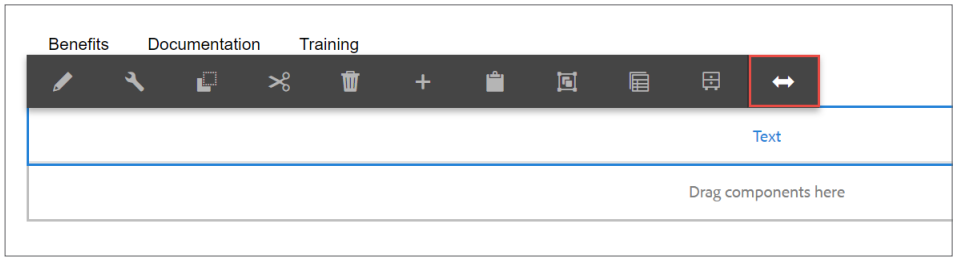
28. Click **Layout Container: Documentation**, as shown. The components and content you insert now will be added under Documentation.



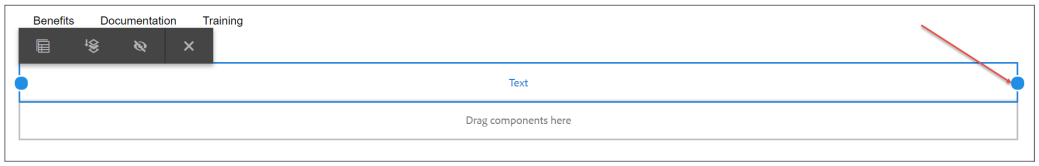
29. On the **Documentation** tab, click the **Drag components here** area.

30. Click the **Insert component** icon (the + icon), and add a **Text** component.

31. Click the **Text** component, and click **Layout**, as shown. The layout is now editable.



32. Drag the blue circle and adjust the layout to 6/12 columns, as shown:



33. Click **Close**. The component should look, as shown:

34. Navigate to the **Core Components An Overview** exercise file provided to you and copy the content from **Documentation.txt**.

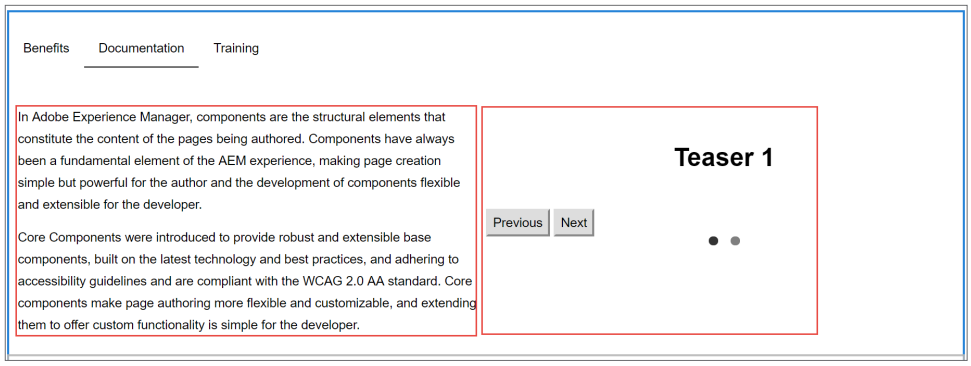
35. Click the **Text** component in AEM, and click the **Edit icon** (pencil icon) to add content.

36. Paste the content you copied in step 34 and click **Done**. The text is added under the **Documentation** tab.

37. Add a **Carousel** component under the Text component in the **Documentation** tab.

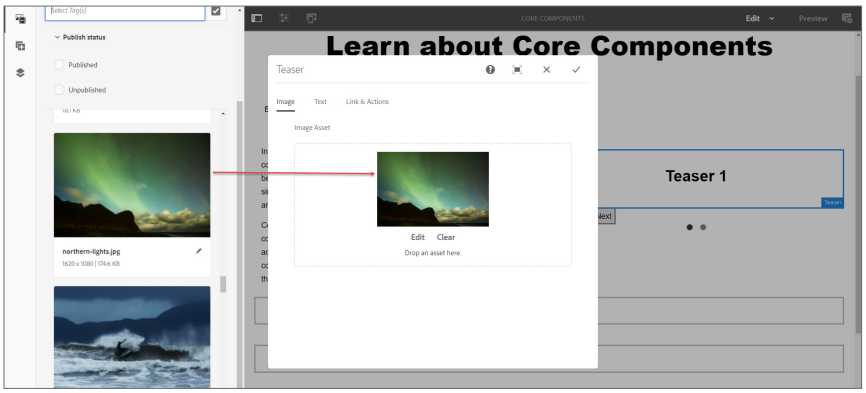
38. Click the **Carousel** component, and click **Layout**. The layout is now editable.

39. Drag the blue circle and adjust the layout to 6/12 columns. The two components are now placed next to each other, as shown:

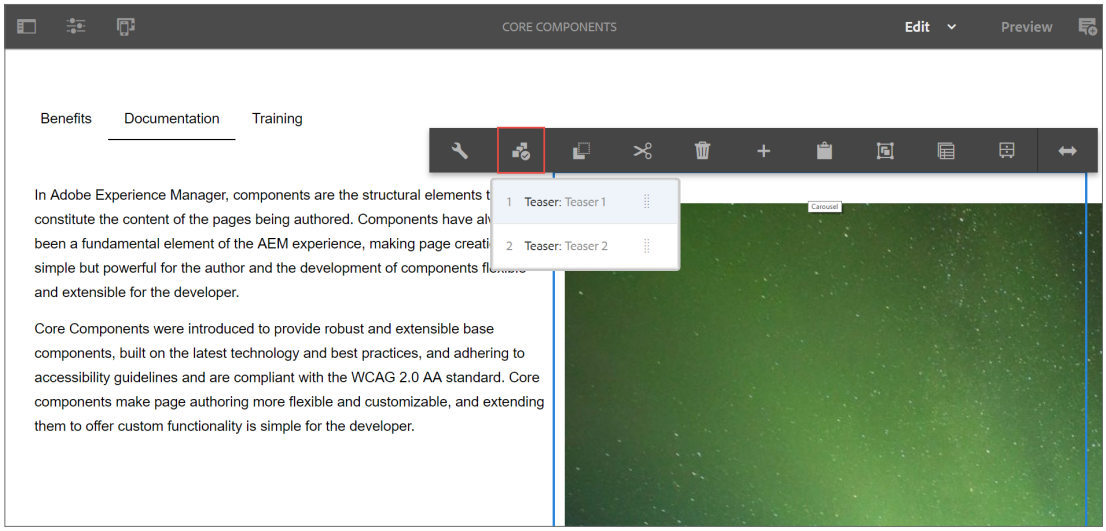


40. Select the **Teaser 1** dialog box and click **Configure**. The **Teaser** dialog box opens.

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** checkbox.
43. On the **Links & Actions** tab, select the **Enable Call-to-Actions** checkbox.
44. In the **Link** field, enter **https://helpx.adobe.com/experience-manager/core-components/using/authoring.html**
45. In the **Text** field, enter **Author Core Components**.
46. Click **Done**.
47. Select the **Carousel** component and click the **Select Panel** icon, as shown:

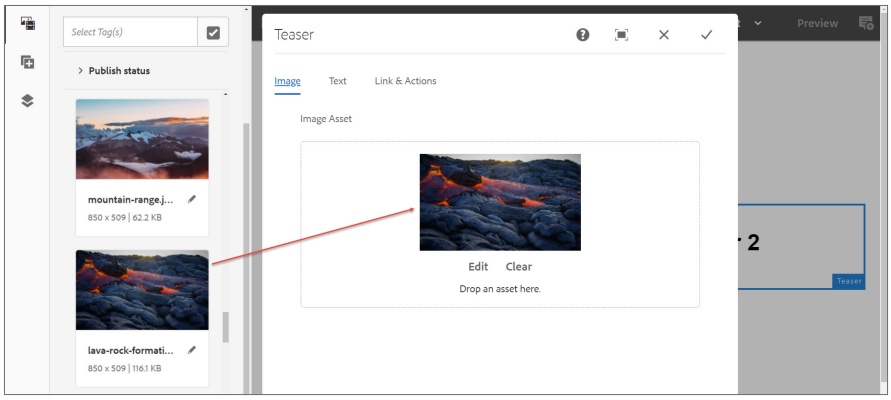


48. Click **Teaser: Teaser 2** to add content under Teaser 2.

49. Click the **Toggle Side Panel** icon to access the options in the left panel.

50. Select **Teaser 2** dialog 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** checkbox.

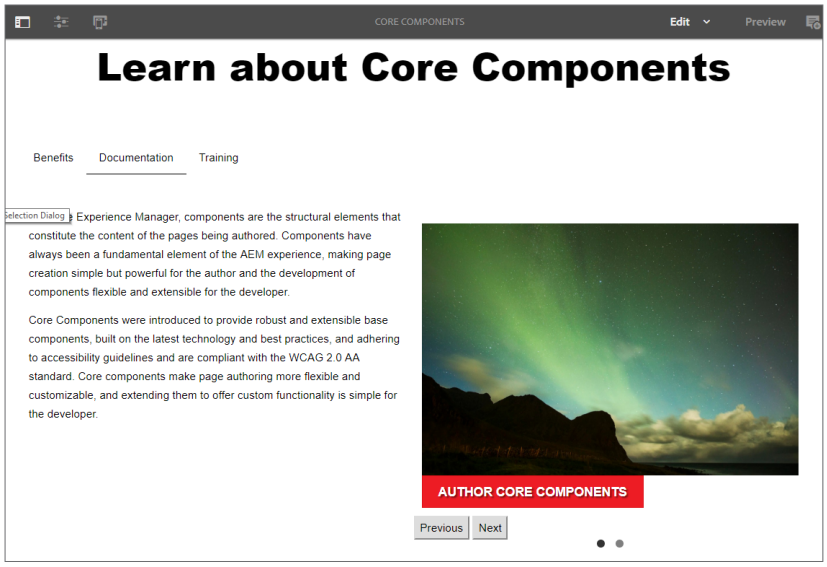
53. On the **Links & Actions** tab, select the **Enable Call-to-Actions** checkbox.

54. Click the **Done** icon (checkmark in the upper right).

55. In the **Link** field, type **https://helpx.adobe.com/experience-manager/core-components/using/developing.html**

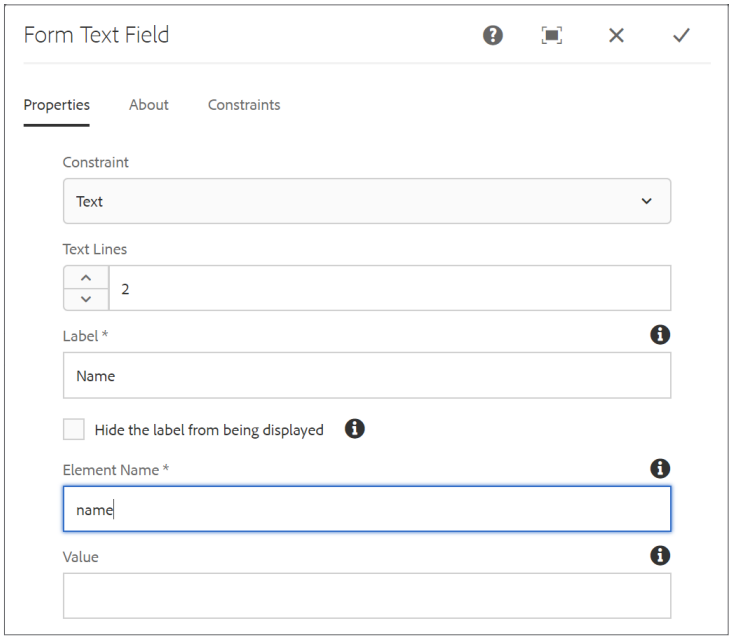
56. In the **Text** field, type **Develop Core Components**.

57. Click **Done** to save the changes. The **Documentation** tab should look, as shown:



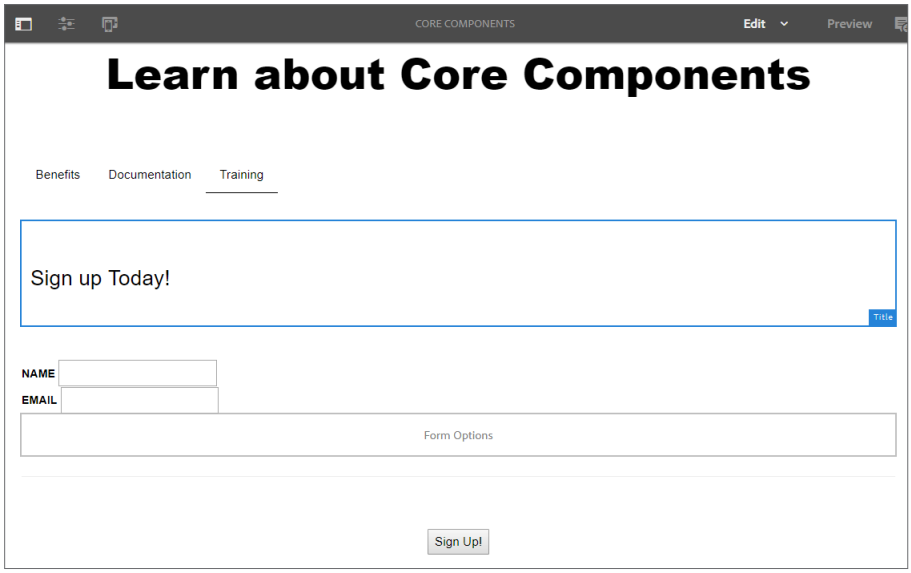
58. Select the **Tabs** component, click the **Select Panel** icon, and click **Layout Container: Training** to add components and content under the Training tab.

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** field, 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 **Close**.
66. Add a **Form Text** component, select it, and click **Configure**. The **Form Text Field** dialog box opens.
67. In the **Label** field, type **Name**.
68. In the **Element Name** field, 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** field, enter **Email**.
72. In the **Element Name** field, 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** field, enter **Audience**.
76. In the **Name** field, enter **audience**.
77. Click **Add** under **Option Entries**, and enter **Author** in the **Text** field.
78. Click **Add** and enter **Developer** in the **Text** field.
79. Click **Done** to save the changes.
80. Add a **Separator** component to the page. A line that separtes the two components are added.
81. Add a **Form Button** component below the Separator component, select it, and click **Configure**. The **Form Button** dialog box opens.
82. In the **Title** field, enter **Sign Up!**
83. Click **Done** to save the changes. The **Training** tab should look, as shown:



84. To view your completed work, click **Preview** in 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 a Experience Fragment called **Core Components**. You can view this fragment by adding an Experience Fragment component onto the page and dragging the Core Components fragment onto the page.

References

Use the following links for more information on:

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

Vital Child Nodes

The vital child nodes of a component are:

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

Authors should follow the following guidelines when working with structure and content components:

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

HTL Business Logic

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: Create a header component

In this exercise, you will create a header component and add it to the template.

1. In your AEM author instance, click **Adobe Experience Manager** from the header bar. The **Tools** console opens.
2. Click **CRXDE Lite**. The **CRXDE Lite** page opens.
3. Navigate to the **/apps/training/components/structure** folder.
4. Right-click the **structure** folder, and click **Create** > **Create Component**. The **Create Component** dialog box opens.
5. Enter the following details in the dialog box:
   a. **Label**: **header**
   b. **Title**: **We.Train Header**
   c. **Group**: **We.Train.Structure**
6. Click **Next**. The **Advanced Component Settings** are displayed. Leave these settings as is
7. Click **OK**. The component is created.
8. Click **Save All** from the actions bar.
9. Expand the header component and double-click **header.jsp**. The editor opens on the right.
10. Delete the existing content and click **Save All**.
11. Right-click **header.jsp, s**elect **Rename**, and change the file name to **header.html**.
12. Click **Save All** from the actions bar.
13. Double-click the **header.html** to open it in the editor on the right for editing.
14. Navigate to the **basic_header.html** file located in the **Working with Components in Adobe Experience Manager** exercise folder provided to you, copy the code, and paste the code in header.html.

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, Nav1, Nav2, and Nav3 are displayed. The header.html also has the Language Navigation and Search proxy components coded in. You can configure these.

```
<h3>Header Component</h3>

<!--/* Core Language Navigation Component */-->
<div class="modal-after">
    <sly data-sly-resource="${'./languagenavigation' @ resourceType='training/components/structure/languagenavigation'}"/>
</div>

<!--/* ------------Using HTL we can quickly create a list of links for the header------------ */-->
<ul class="nav navbar-nav navbar-center">
    <li class="nav navbar-nav navbar-left" data-sly-repeat="${currentPage.listChildren || ['nav1','nav2', 'nav3']}">
        <a href="${item.path || '#' @ extension='html'}">${item.title || item}</a>
    </li>
</ul>

<!--/* Core Search Component */-->
<div class="col-md-12">
    <sly data-sly-resource="${'./search' @ resourceType='training/components/structure/search'}"></sly>
</div>
```

Before you add the header component to the template, you must add a cq:dialog node to the header component.

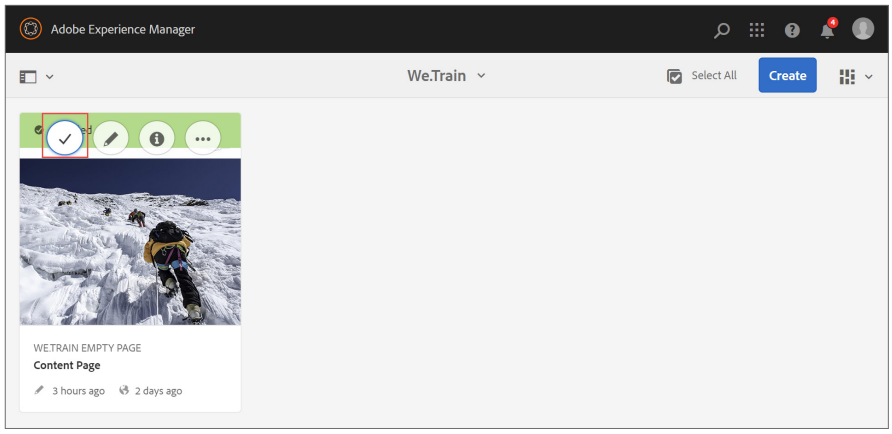
17. Right-click the **header** component, and click **Create** > **Create Node**. The **Create Node** dialog box opens.

18. Provide the following details:
    a. **Name**: **cq:dialog**
    b. **Type**: **nt:unstructured**

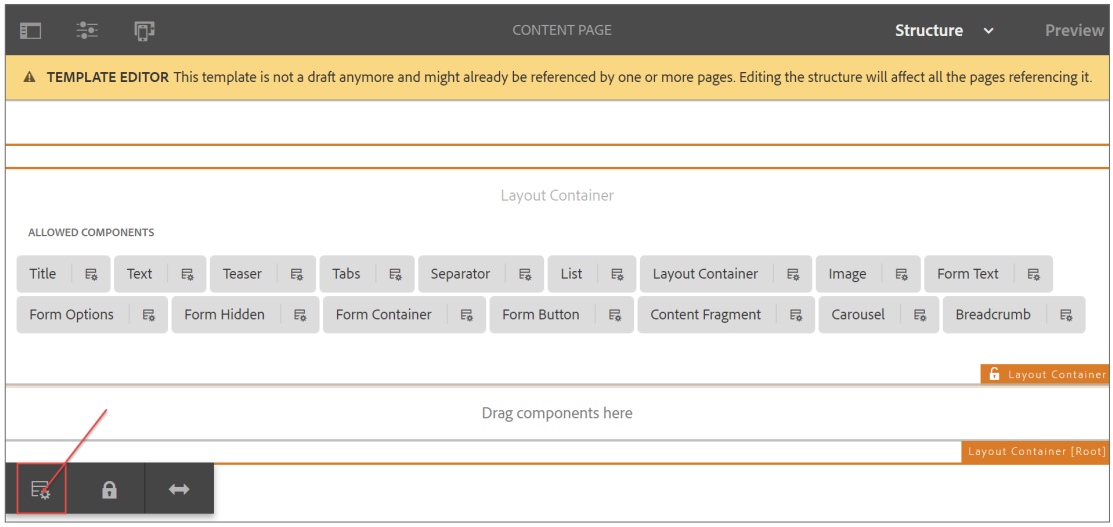
19. Click **OK**. The node is created.
20. Click **Save All**.
21. Click **CRXDE Lite** from the header bar to navigate back to AEM.

To add the header component to the template:

22. Click **Tools** > **Templates**. The **Templates** console opens.
23. Click the **We.Train** folder, and select the **Content Page** template by clicking the **Select** icon, as shown:



24. Click **Edit (*e*)** from the actions bar. The Template Editor opens.
25. Select the **Layout Container [Root]** and click the **Policy** icon from the component toolbar, as shown. The **Layout Container** policy wizard opens.



26. In the **Properties** section, on the **Allowed Components** tab, scroll down the components list, and select the **We.Train.Structure** checkbox. This group contains the header component.

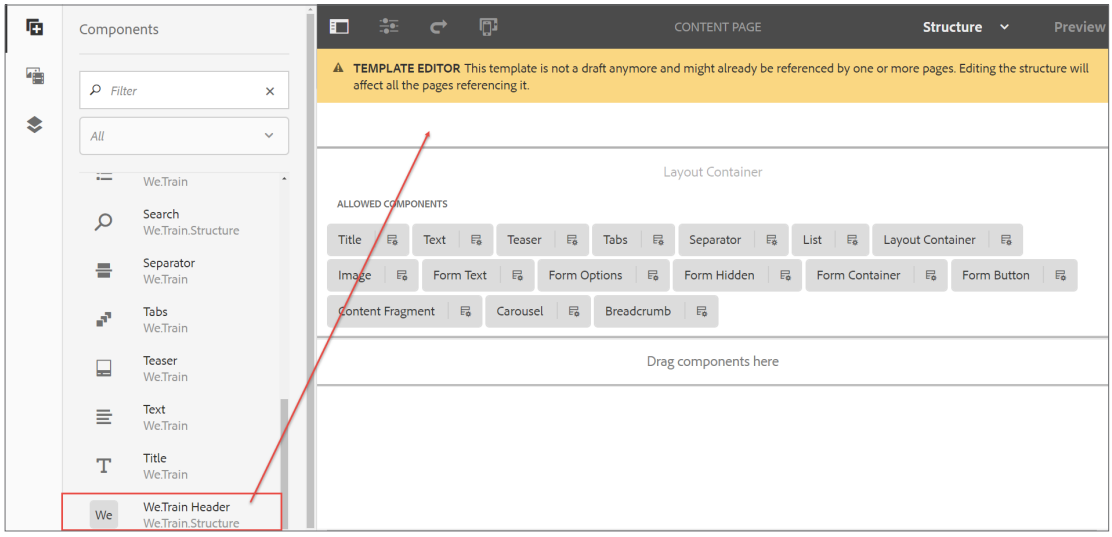
27. Click **Done** (checkmark in the upper right) to save the changes.

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.

---

28. Click the **Toggle Side Panel** icon, if the left panel is not visible, and click the **Components** icon. The **Components** browser opens.

29. 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.

30. Select the **Language Navigation** component, and click the **Policy** icon from the component toolbar.

31. In the **Policy** section, type **We.Train LangNav Content Policy** in the **Policy Title** field.

32. In the **Properties** section, type **/content/we-train** in the **Navigation Root** field.

33. Click **Done** (checkmark) to save the changes.

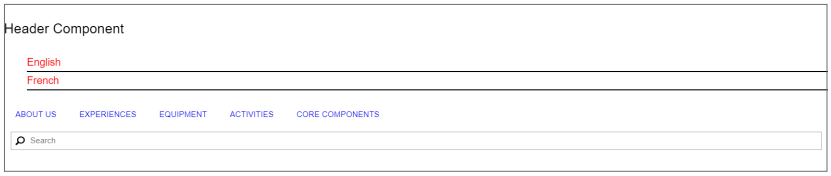
34. Select the **Search** component, and click the **Policy** icon.

35. In the **Policy** section, type **We.Train Search Content Policy** in the **Policy Title** field.
36. In the **Properties** section, type **/content/we-train** in the **Search Root** field.
37. 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/weretail/components/structure/header in CRXDE Lite.

---

38. 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:



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.

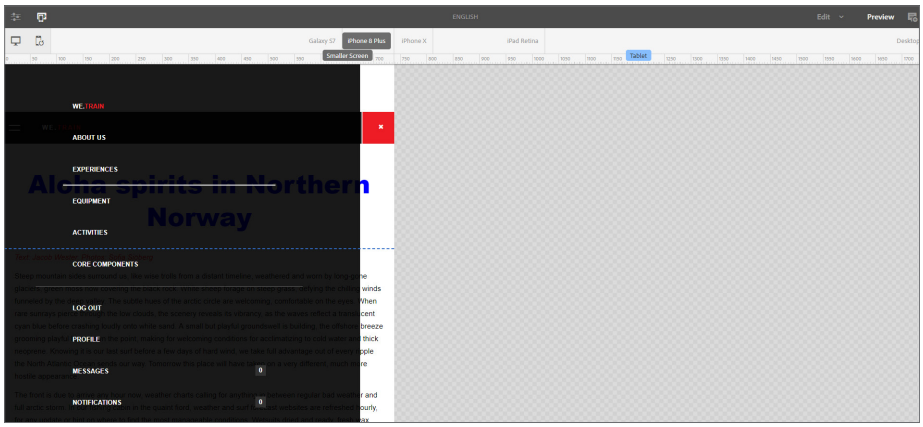
39. Click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.
40. Navigate to the **/apps/training/components/structure/header** node.
41. Double-click the **header.html** to open the script for editing.
42. Update the **header.html** script by replacing the existing code with new code from **basic_full_header.html** exercise file provided to you.
43. Click **Save All**.
44. Review the code you added. Notice in the following code snippet of header.html, there is an account navigation bar and a responsive site navigation bar. Most anchors are `href='#'` that help mock the page for the business logic. Notice the `${header.rootPage.path}` and `${!header.anonymous}` variables from the business logic. As the header variable is not defined, it is not evaluated.

45. Navigate to the **English** page of the We.Train site and refresh the page.

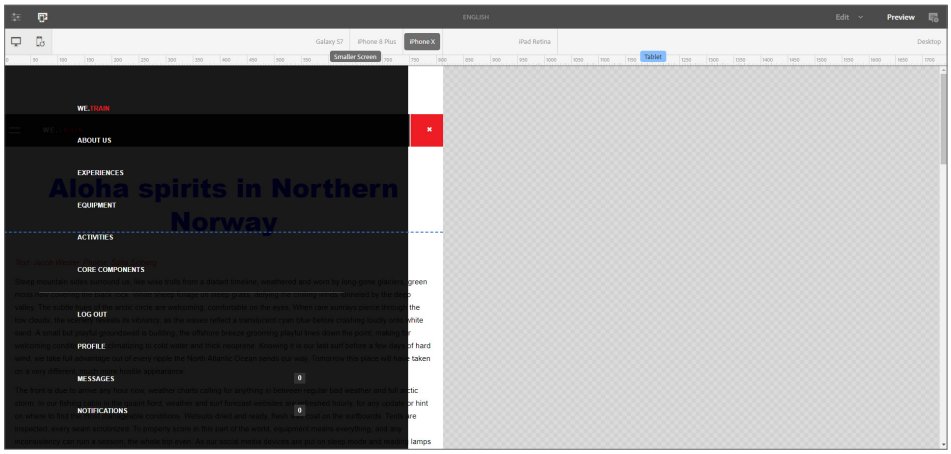
46. Click **Preview**, and in the **Preview** mode, click the **Emulator** icon.

47. Click Desktop, iPhone 8 Plus, and iPhone X at the top, and notice that the header is responsive.

iPhone 8 Plus



iPhone X



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 will cover 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** field, and click **OK**. The file is created.

To add code to the header.js file:

5. Double-click the **header.js** file to open the script for editing.
6. Use the code from **header.js** in the **Working with Components in Adobe Experience Manager** exercise folder provided to you.
7. Click **Save All**.
8. Review the code you added now. In the following code snippet:
   a. Line 4 helps the language root of the website to display the child pages in navigation.
   b. Line 12 helps log a message in the server-side Javascript.
   c. Line 14 helps filter the child pages only if **Hide in navigation** page property is selected.

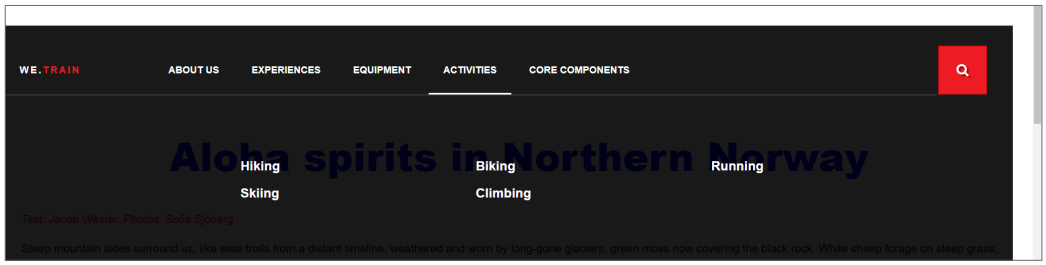
```
1  "use strict";
2  use(function() {
3      var items = [];
4      var root = currentPage.getAbsoluteParent(2);
5
6      //make sure that we always have a valid set of returned items
7      //if navigation root is null, use the currentPage as the navigation root
8      if(root == null){
9          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     var anon = (request.getRequestParameter("anonymous") != null) ?
21             request.getRequestParameter("anonymous").toString() : "false";
22
23     return {
24         items: items,
25         rootPage: root,
26         anonymous: anon
27     }
28 });
```

To update the code in the header.html page:

9. Double-click the **header.html** to open the script for editing.
10. Update the **header.html** script by replacing the existing code with new code from **js_header.html** in the **Working with Components in Adobe Experience Manager** exercise folder.
11. Click **Save All**.
12. Open **header.html** to view the code you added. Notice the following:
    a. Line 2: `data-sly-use.header="header.js"` helps define the header variable with a HTL Javascript Use-script.
    b. `${header.rootPage.path}` and `${!header.anonymous}` are now able to render based on the JavaScript.
    c. Line 53: `data-sly-repeat="${header.items}"`. Instead of using the child pages of the current page, the HTL statement requests the items object from the header.js script.

To view the changes on the English page:

13. 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. The English page opens.
14. Click **Preview** from the page toolbar, and hover the cursor over the **Activities** subpage, as shown. The header component has the same functionality as in the previous exercise.

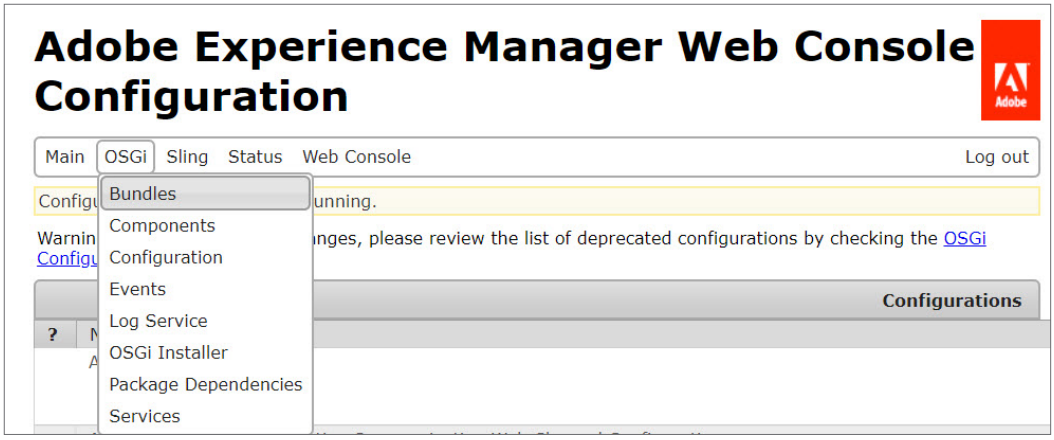


Exercise 3: Add a Sling Model business logic to the header component

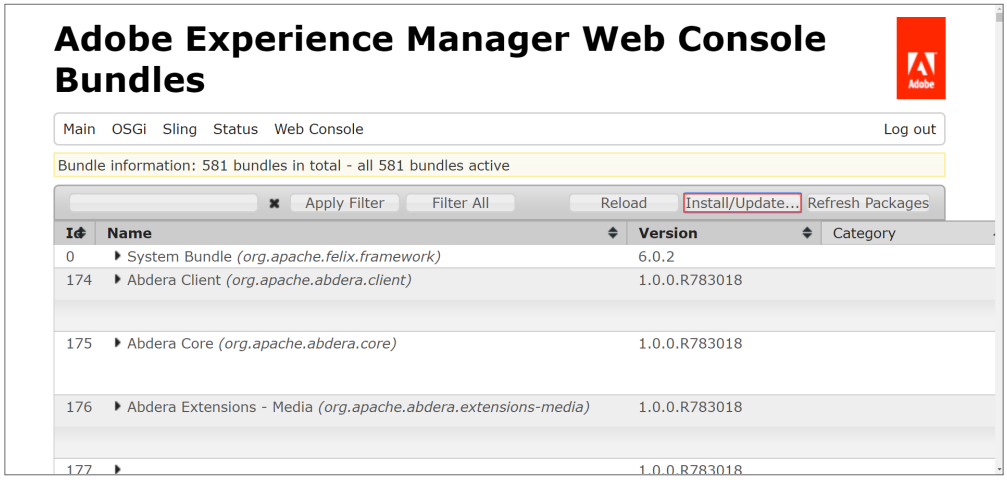
In this exercise, you will use a Sling Model for the business logic of the component. Using Sling Models enable heavier computation as well as extensibility for headless implementations. For convenience, you will upload the we.train.core-0.0.1-SNAPSHOT.jar file, which contains a Sling Model. This Sling Model implements the exact same business logic as the JavaScript file you created earlier.

To install the bundle:

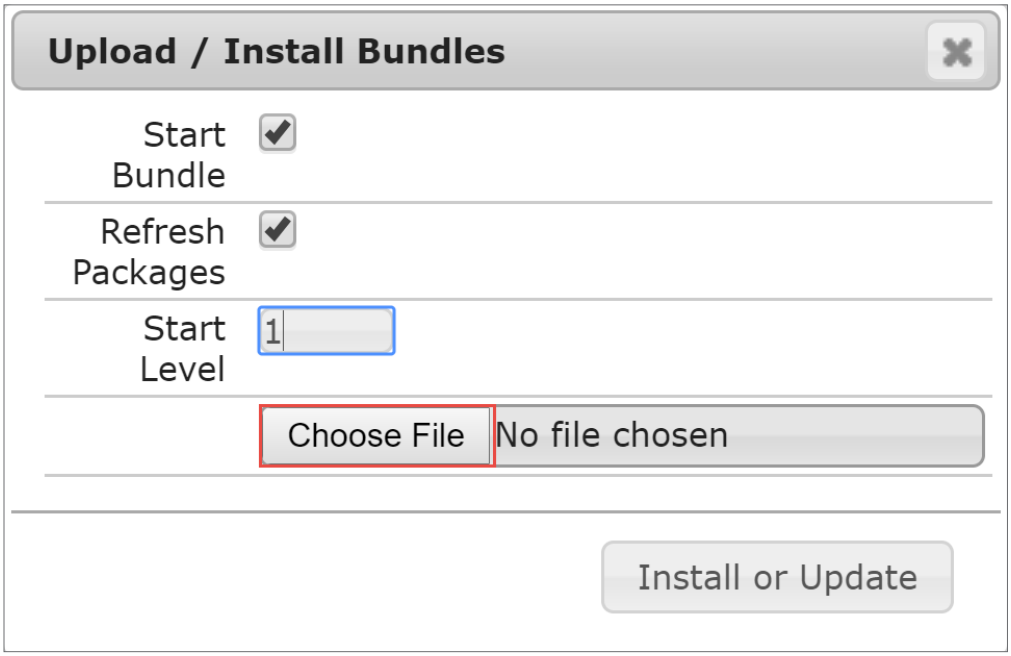
1. In your AEM author instance, click **Adobe Experience Manager** from the header bar, and click the **Tools** icon. The **Tools** console opens.
2. Click **Operations** > **Web Console**. The **Adobe Experience Manager Web Console Configuration** page opens.
3. Click **OSGi** from the header bar, and click **Bundles** from the drop-down menu, as shown. The **Adobe Experience Manager Web Console Bundles** page opens.



4. Click **Install/Update** from the actions bar, as shown. The **Upload / Install Bundles** dialog box opens.



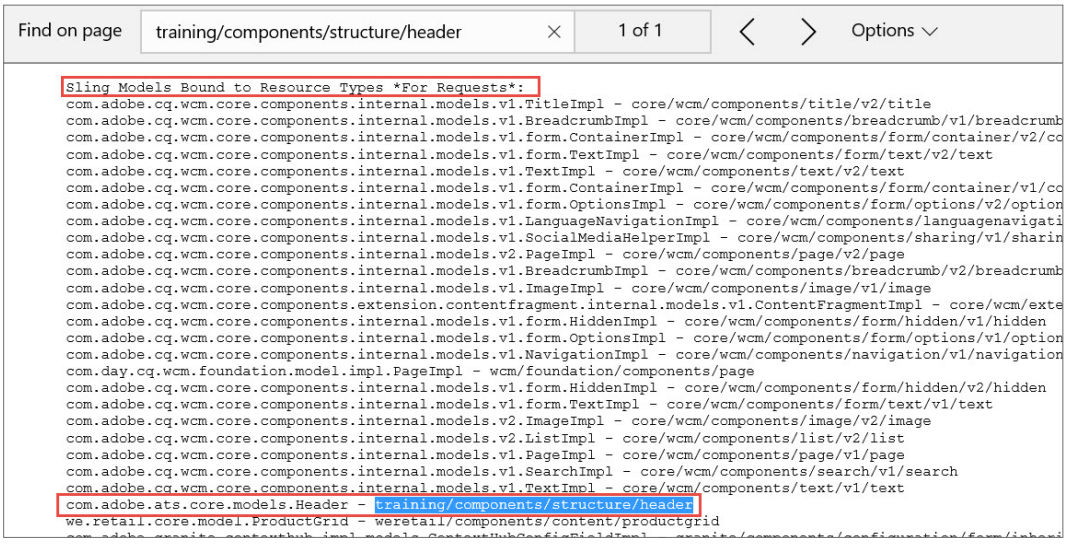
5. Select the **Start Bundle** and **Refresh Packages** checkboxes, enter **1** in the **Start Level** field, and click **Choose File**, as shown. The **Open** dialog box appears.



6. Navigate to the **Working with Components in Adobe Experience Manager** exercise folder on your file system and double-click the **we.train.core-0.0.1-SNAPSHOT.jar file**. The file is selected.
7. Click **Install or Update** in the **Upload / Install Bundles** dialog box. The **Adobe Experience Manager Web Console Bundles** page refreshes when the installation is complete.

To verify if the bundle is installed successfully on your instance:

8. 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.
9. 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, use **Ctrl+F** or **Command F**). This confirms the bundle is installed successfully.



To update the code on the header.html page:

10. Click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.
11. Navigate to the **/apps/training/components/structure/header** node.
12. Double-click the **header.html** to open the script for editing.
13. Update the **header.html** script by replacing the existing code with new code from **model_header.html** file provided to you.
14. Click **Save All**.

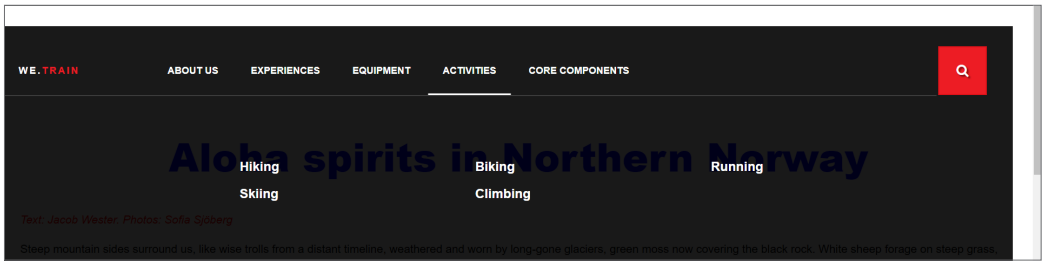
15. Review the code you added in **header.html**. Notice that the model_header.html script is the same as js_header.html, except you are requesting the Sling Model rather than the Javacript.

```
<!--/* -----------HTL with Sling Model business logic added----------- */-->
<sly data-sly-use.header="com.adobe.ats.core.models.Header" />
<div  class="navbar navbar-inverse navbar-fixed-top hidden-xs hidden-sm">
    <div class="container-fluid">
        <ul class="nav navbar-nav navbar-left">
           <li><a href="#" data-toggle="modal" data-target=".we-LanguageModal" title="${header.rootPath.name}" style="text-
transform: uppercase;">
                <i class="fa fa-globe" aria-hidden="true"></i>
                ${header.rootPage.name || 'en'}
            </a></li>
        </ul>
        <ul class="nav navbar-nav navbar-right">
            <li data-sly-test.anonymous="${header.anonymous}"><a href="${'#' @ extension = 'html'}">${ 'Login' @ i18n }</a></li>
            <sly data-sly-test="${!anonymous}">
                <li><a href="${'#' @ extension = 'html'}">${ 'Account' @ i18n }</a></li>
                <li><a href="${'#' @ extension = 'html'}">${ 'Profile' @ i18n }</a></li>
                <li><a href="${'#' @ extension = 'html'}"><i class="fa fa-envelope-o"></i> ${
                    'Messages' @ i18n }
                    <span class="badge" id="we-retail-message-count" data-siteurl="#">0</span></a></li>
                <li><a href="${'#' @ extension = 'html'}"><i class="fa fa-flag-o"></i> ${ 'Notifications' @ i18n }
                    <span class="badge" id="we-retail-notification-count" data-siteurl="#">0</span></a></li>
                <li><a href="/system/sling/logout.html?resource=${currentPage.path @ extension = 'html'}">${ 'Log out' @ i18n }</a></li>
            </sly>
        </ul>
    </div>
</div>

<div class="container">
    <nav class="navbar navbar-inverse navbar-absolute-top">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-toggle="collapse"
                    data-target="#we-example-navbar-collapse-inverse" aria-expanded="false">
                <span class="sr-only">${ 'Toggle navigation' @ i18n }</span>
                <span class="icon-bar"></span> <span class="icon-bar"></span>
```

To view the changes on the English page:

16. 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).

17. Click **Preview** from the page toolbar, and ensure you can see the subpages of the Activities page, as shown:



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 simply 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 the component

In this exercise, you will create a new component that will include a custom dialog for authoring input.

To create an edit dialog:

1. In the **CRXDE Lite** page (http://localhost:4502/crx/de.), navigate to the **/apps/training/components/structure** folder.
2. Right-click the **structure** folder, and click **Create** > **Create Component.** The **Create Component** dialog box opens.
3. Enter the following values in the dialog box:
   a. **Label**: **pagehero**
   b. **Title**: **Page Hero**
   c. **Group**: **We.Train.Structure**
4. Click **Next**.
5. Leave the Advanced Component Settings as is and click **OK**. The component is created.
6. Click **Save All** from the actions bar.
7. Expand the pagehero node in CRXDE Lite and double-click **pagehero.jsp**. The editor opens on the right.
8. Delete the existing content and click **Save All**.
9. Right-click **pagehero.jsp**, select **Rename**, and change the file name to **pagehero.html**.
10. Click **Save All**.

To add code to the pagehero.html page:

11. Double-click the **pagehero.html** to open the script for editing.

12. Update the **pagehero.html** script by replacing the existing content with new code from **purehtl_pagehero.html**.

Note: purehtl_pagehero.html has business logic directly in the HTL. Using this method, a front-end developer can create the entire component. Sometimes, content may need to be rendered outside of an AEM site. Using a Sling Model with your component can easily accomplish this. If you like to see the pagehero component with a Sling model, use model_pagehero.html rather than purehtl_pagehero.html. You need to also ensure
we.train.core-0.0.1-SNAPSHOT.jar is installed as it contains the Hero.java model (Java class is included in the Exercise files).

13. Review the code you just included in **pagehero.html**. Notice that the HTL script uses the global variables properties and resource, which helps return values persisted by the dialog box. As you do not have a dialog box yet, the values will not return anything.

```
1.  <!--/*
2.  The first two lines determine if an image has been uploaded via the dialog
3.  First preference is an asset dragged and dropped from the Side Panel.
4.
5.  Asset uploaded through the dialog box is uploaded to ~/page/../pagehero/
    file node
6.  An Asset from the DAM (drag and drop) is saved as a property called fileRefer-
    ence
7.  */-->
8.  <sly data-sly-test.imageUpload="${'{0}{1}' @ format=[resource.path, '/file']}"/>
9.  <sly data-sly-test.imagePath="${properties.fileReference || imageUpload}"/>
10.
11.   <!--/* If imagePath is empty, then a placeholder is shown. */-->
12.   <div class="we-HeroImage width-full${!wcmmode.disabled ? ' cq-dd-image' : ''}"
13.      style="${'background-image:url({0});' @ format = image-
    Path, context='styleString' }">
14.
15.     <!--/* Content that will be displayed if it's set in the dialog */-->
16.     <div class="container">
17.       <div class="we-HeroImage-wrapper">
18.         <p>${hero.heading}</p>
19.         <strong class="we-HeroImage-title h1">${properties.jcr:title}</strong>
20.
21.         <p data-sly-test="${properties.buttonLabel && properties.buttonLinkTo}">
22.           <a class="btn btn-primary" href="${properties.buttonLinkTo @ exten-
    sion = 'html'}" role="button">
23.           ${properties.buttonLabel}
24.           <i class="fa fa-arrow-right" aria-hidden="true"></i>
25.           </a>
26.         </p>
27.       </div>
28.     </div>
29.   </div>
```

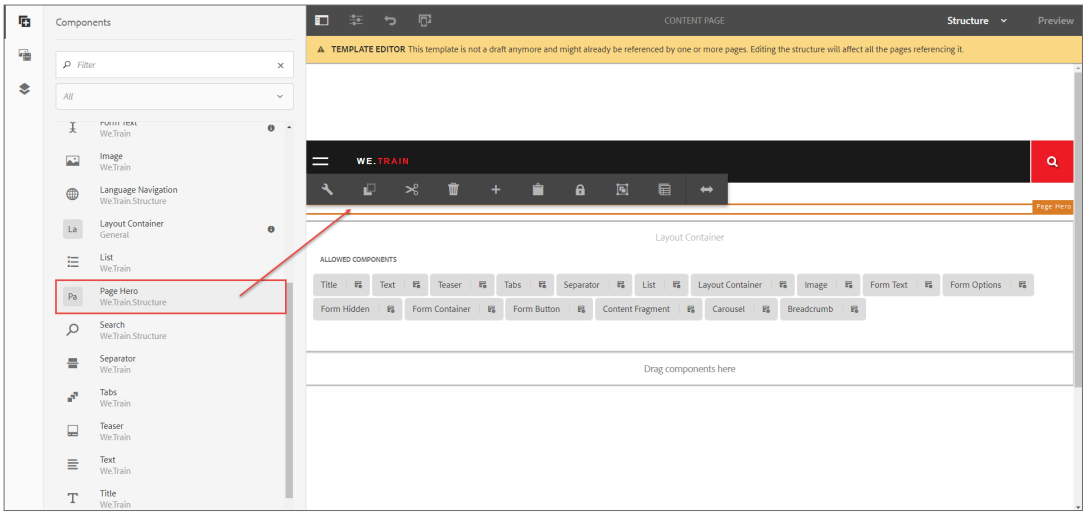
14. Click **Save All**.

Before, you add the pagehero component to the template, you need to add a cq:dialog node to the pagehero component.

15. Select the **pagehero** component, and click **Create** > **Create Node**. The **Create Node** dialog box opens.
16. Update the following values:
    a. **Name**: **cq:dialog**
    b. **Type**: **nt:unstructured**
17. Click **OK** and save the changes.
18. Click **CRXDE Lite** from the header bar to navigate back to AEM.

To add the pagehero component to the template:

19. Click **Tools** > **Templates**. The **Template** console opens.
20. Click the **We.Train** folder to open it, and select the **Content Page** template by clicking the **Select** icon (checkmark).
21. Click **Edit (*e*)** from the actions bar. The **Template Editor** opens.
22. Click the **Toggle Side Panel** icon, if not already selected, and click the **Components** icon from the left panel. Notice that the Page Hero component is available in the Components browser.
23. Drag the **Page Hero Component** from the left panel above the **Layout Container** below the **Header** component, as shown. The **Page Hero** component is added to the template.

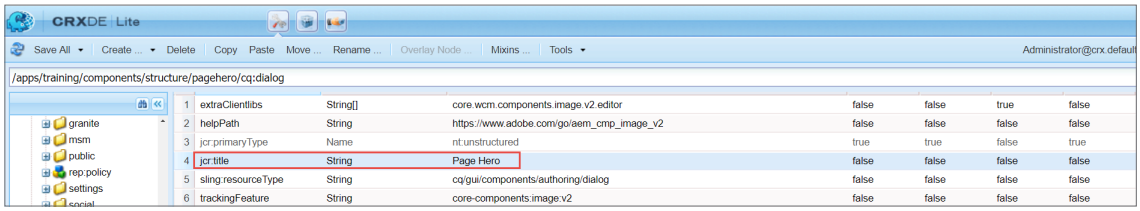


Now that you have added the Page Hero component to the template, you can start to build and test the dialog box.

24. Go to 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 can copy the Core Image dialog box, and then modify it according to your needs.

25. Navigate to the **/apps/training/components/structure/pagehero** node.
26. Delete the **cq:dialog** node below the pagehero node.
27. Click **Save All**.
28. Navigate to the **/apps/core/wcm/components/image/v2/image** node.
29. Right-click the **cq:dialog** node, and click **Copy**. The node is copied.
30. Navigate to the **/apps/training/components/structure** folder.
31. Right-click the **pagehero** component, and click **Paste**. The cq:dialog node of Core Image is added to the pagehero component.
32. Click **Save All**.
33. Select the **cq:dialog** node, and on the **Properties** tab, double-click **jcr:title**, and update the value as **Page Hero**, as shown:



34. Right-click the **extraClientlibs** property, and click **Delete** to delete it.
35. Similarly, delete the **helpPath** property.
36. Click **Save All**.
37. Navigate to the **/apps/training/components/structure/pagehero/cq:dialog/content/items/tabs/items/metadata/items/columns** node.
38. Right-click the **columns** node, and click **Delete** to delete it.
39. Click **Save All**.
40. Navigate to the **/apps/training/components/structure/pagehero/cq:dialog/content/items/tabs/items/metadata** node.
41. Right-click the metadata node, and click **Rename** to rename it.
42. Rename it as **hero**.
43. Click **Save All**.
44. Select the **hero** node, and in the **Properties** tab, double-click **jcr:title**.
45. In the **Value** field, enter **Hero**.

46. Click **Save All**.

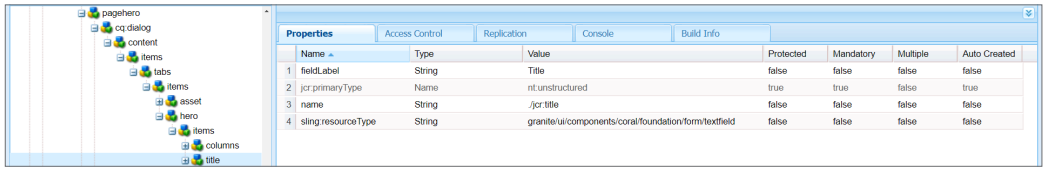
47. Right-click the **items** node that is below the **hero** node, and click **Create** > **Create Node**. The **Create Node** dialog box opens.

48. Update the following details:
    a. **Name**: **title**
    b. **Type**: **nt:unstructured**

49. Click **OK** > **Save All** to save the changes.

50. 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



51. Click **Save All**.

Similar to the **title** node, you will create **heading**, **buttonLabel**, and **buttonLink** form fields below the **/hero/items** node. Instead of creating the formfields, you will copy/paste the **title** node and update the required values accordingly.

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

53. Ensure you have selected the **/hero/items** node, right-click items, and click **Paste**. The **Copy of title** node is created.

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

55. 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

56. Click **Save All**.

57. Perform steps 51 and 53, and create **buttonLabel** and **buttonLink** nodes below the **/hero/items** node.

58. 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

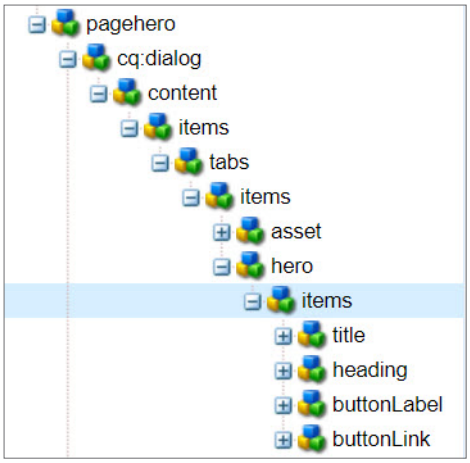
59. Click **Save All**.

60. 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

62. Click **Save All**.

Your form fields structure should look similar to the one given in the below screenshot:



To enable the dialog box on the page, you must unlock the Page Hero component in the template.

63. Navigate back to the **Content Page** template. If it is not open, click **CRXDE Lite** from the header bar, and click **Tools** > **Templates**. The **Templates** console opens.

64. Navigate to the **We.Train** folder, select the **Content Page** template, and click **Edit (*e*)** from the actions bar. The **Template Editor** opens.

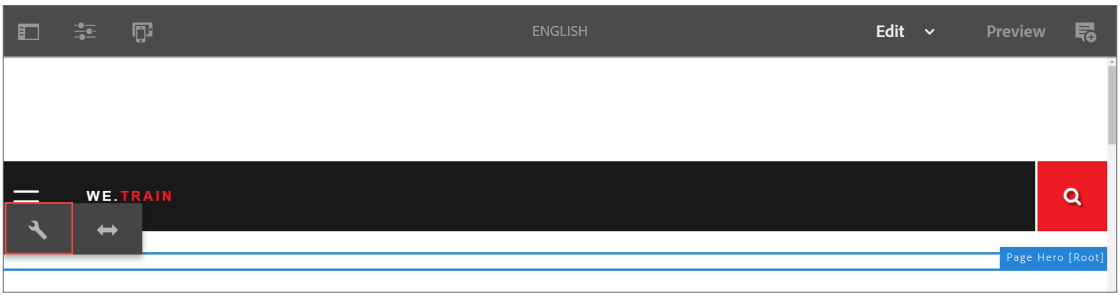
65. Select the **Page Hero** component, at the top of the page, and click the **Unlock structure component** icon from the component toolbar. The **Unlock** dialog box opens.

66. Click **Unlock**.

67. Now that the Hero component is added to the template, you can test the component on a page. Navigate to the English page of the We.Train site and edit it. Observe that the Hero component

is now available on the page. If you hover the cursor over the component, a blue box appears, indicating that the component is editable.

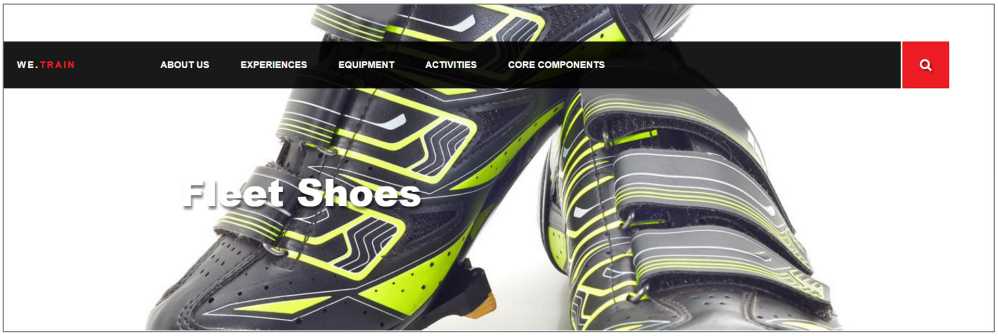
68. Hover over the **Page Hero[Root]** component and click the **Configure** (wrench icon) icon, as shown. The **Page Hero** dialog box opens.



69. On the **Asset** tab, drop an asset or browse for a file to upload an image.

70. Click the **Hero** tab, and add a title in the **Title** field.

71. Click **Done** (checkmark) to save the changes. If you see an image with a title, you have successfully created a dialog box with your component.



Exercise 5: Add an editconfig node to the component

In this exercise, you will add an cq:editConfig node to the Page Hero component. Instead of recreating an existing node structure, you will copy the Core Image component, add it to the Page Hero component, and update the required values accordingly.

To add an editConfig node:

1. 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/structure** folder.
5. Right-click the **pagehero** component, and click **Paste** to paste the node you copied in step 3. The cq:editConfig node is added.
6. Right-click the **cq:inplaceEditing** node, which is below the **cq:editConfig** node, and click **Delete**.
7. Click **Save All**.
8. Navigate to the **/apps/training/components/structure/pagehero/cq:editConfig/cq:dropTargets/image** node, select the **parameters** node, and add the following property:

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

9. Click **Save All**.

To test if the cq:editConfig is added to the Page 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, and click the **Assets** icon. The **Assets** browser opens.
12. Drag an image from the **Assets** browser onto the **Page Hero** component. The image is added to the component. This indicates the success of this exercise. You used cq:editConfig to add the ability of your Page Hero component to accept content through the drag-and-drop feature.

Exercise 6: Add a component client library to the component

Now that you have configured the hero component as per your needs, you want to apply a design so that the hero image in the header looks more appealing. In this exercise, you will create a component client library that will have a specific design for the hero component.

To add a component clientlib:

1. Click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.
2. Navigate to the **/apps/training/component/structure** folder.
3. Right-click the **pagehero** component, and click **Create** > **Create Node.** The **Create Node** dialog box opens.
4. Provide the following details:
   a. **Name**: **clientlibs**
   b. **Type**: **cq:ClientLibraryFolder**
5. Click **OK** > **Save All**.
6. Select the **clientlibs** node and add the following properties:

Name	Type	Value
allowProxy	Boolean	true
categories	String[]	we.train.pagehero

7. Click **Save All**.
8. Right-click the **clientlibs** folder, and click **Create** > **Create Folder**. The **Create Folder** dialog box opens.
9. Enter **css** in the **Name** field, and click **OK**.
10. Click **Save All**.
11. Right-click the css folder, and click **Create** > **Create File**. The **Create File** dialog box opens.
12. Enter **pagehero.css** in the **Name** field, and click **OK**.
13. Click **Save All**.

To add code to the pagehero.css file:

14. Double-click **pagehero.css** to open the script for editing.

15. Navigate to the exercise file provided to you, and use the code in **pagehero.css** file to update the pagehero.css script in CRXDE Lite.
16. Click **Save All**.
17. Right-click the **clientlibs** folder, and click **Create** > **Create File**. The **Create File** dialog box opens.
18. Enter **css.txt** in the **Name** field, and click **OK**.
19. Click **Save All**.
20. Double-click **css.txt** to open the file for editing.
21. Add `css/pagehero.css` to the css.txt file.

To include the component client library at the top of the page, you need to embed it to the clientlibs-base.

22. Navigate to the **/apps/training/clientlibs** folder, and select **clientlib-base** folder.
23. On the **Properties** tab, double-click the embed property to edit its value. The **Edit embed** dialog box opens.
24. Click +, add **we.train.pagehero** in the field, and click **OK**. This will add the css for the pagehero component at the top of the page.
25. Click **Save All**.

To observe the changes from the clientlib on the Page Hero component:

26. Navigate to the English page of the We.Train site. (http://localhost:4502/editor.html/content/we-train/en.html)
27. Click the **Page Information** icon from the page toolbar, and click **View as Published** from the drop-down menu. The page opens in a new tab.
28. Right-click on the page, and click **View page source** from the list. The source code opens in a new tab of the browser.
29. Click the clientlib-base.css line, as shown.

```
<!DOCTYPE HTML>
<html lang="en">
    <head>
    <meta charset="UTF-8"/>
    <title>English</title>


    <meta name="template" content="content-page"/>




<link rel="stylesheet" href="/etc.clientlibs/training/clientlibs/clientlib-base.css" type="text/css">
```

30. Search for **pagehero** in the CSS file. The client library should now include the CSS from the pagehero component. You will also see the we-heroImage class names relating to the Hero component. You have successfully added CSS to your component using a client library and verified that it is delivered along with the page.

```
.pagehero{
        position: relative;
    top: -170px;
    margin-bottom: -170px;
    overflow: visible;
}
.we-HeroImage-title,
.we-HeroImage p {
  font-family: inherit;
  font-weight: 700;
  line-height: 1.1;
  color: inherit;
  margin-top: 24px;
  margin-bottom: 12px;

}
.we-HeroImage-title {
  font-size: 48px;
  font-weight: 900;
  line-height: 1.29166667;
  display: block;
  margin: 0 auto 20px;
  text-shadow: 2px 4px 3px rgba(0, 0, 0, 0.5);
}
.we-HeroImage p {
  font-size: 24px;
  font-weight: 400;
  /* line-height: 2.58333333; */
  margin: 0 auto;
}
.we-HeroImage {
/*       */
  margin-bottom: 40px;
  color: #ffffff;
  background-color: #eeeeee;
  min-height: 460px;
  padding: 40px 0;
  background: #555555 50% no-repeat;
  background-size: cover;
  position: relative;
  text-align: center;
```

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 7: 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.

1. Ensure the **CRXDE Lite** tab 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 Component**. The **Create Component** dialog box opens.
4. Provide the following details:
   a. **Label: footer**
   b. **Title: We.Train Footer**
   c. **Group**: **We.Train.Structure**
5. Click **Next**, and click **OK**. The component is created.
6. Click **Save All**.
7. Expand the **footer** component and double-click **footer.jsp**. The editor opens on the right.
8. Delete the existing content and click **Save All**.
9. Right-click **footer.jsp**, select **Rename**, and change the name to **footer.html**.
10. Click **Save All** from the actions bar.

To add code to the footer.html page:

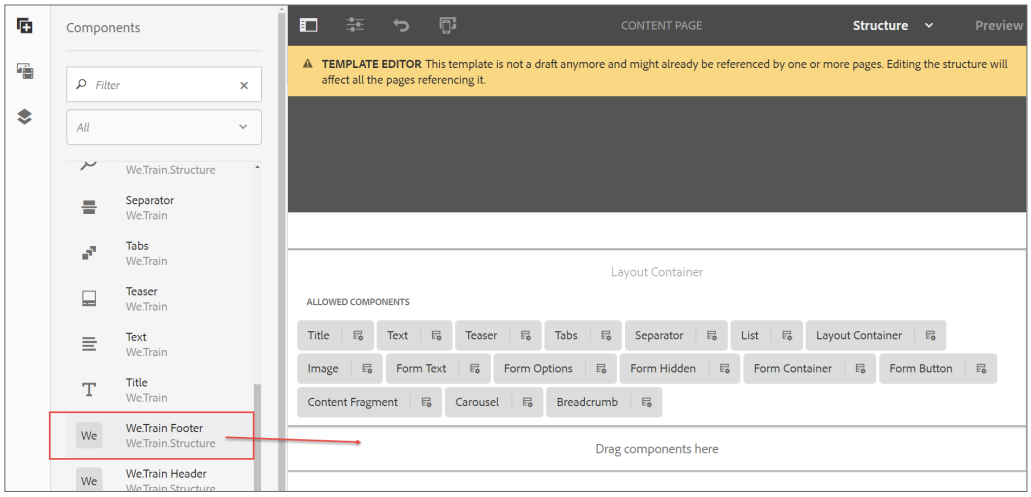
11. Double-click the **footer.html** to open the script for editing.
12. Replace the existing content with code from the **model_footer.html** exercise file provided to you. The footer component uses a Sling Model called **Footer.java**. The bundle, we.train.core-0.0.1-SNAPSHOT.jar you installed previously 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.

You must add a cq:dialog node to add the footer component to the template:

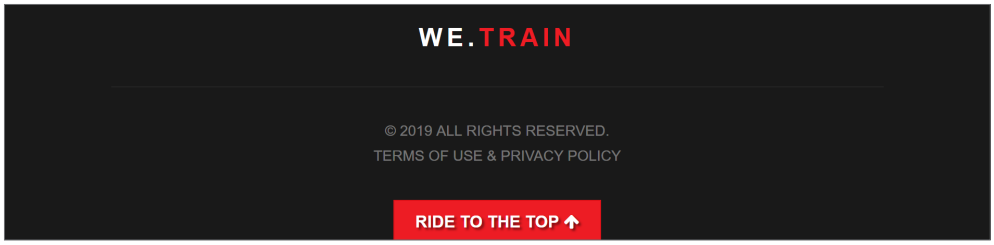
13. Select the **footer** component, and click **Create** > **Create Node**. The **Create Node** dialog box opens.
14. Provide the following details:
    a. **Name**: **cq:dialog**
    b. **Type**: **nt:unstructured**
15. Click **OK** and click **Save All**.

To add the footer component to the template:

16. In your AEM author instance, click **Adobe Experience Manager** from the header bar, click the **Tools** icon, and click **Templates**. The **Templates console** opens.
17. Click the **We.Train** folder, select the **Content Page** template, and click **Edit (*e*)** from the actions bar.
18. Click the **Toggle Side Panel** icon from the page toolbar, and click the **Components** icon. The **Components** browser opens. Notice that the We.Train Footer is available.
19. Drag the **We.Train Footer** component from the **Components** browser onto the **Drag components here** placeholder, as shown. The component is added.



20. Refresh the English page and notice how the footer is rendering.



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:

21. Ensure the **English** page is open.
22. Select the **Drag components here** area, click the **Insert component** icon (+ icon), and add a **List** component.
23. Select the **List** component, and click the **Configure** (wrench) icon from the component toolbar. The **List** dialog box opens. Observe the dialog box and the options it provides.
24. Click X (Cancel) to close the dialog box.

To create a design dialog:

25. In CRXDE Lite, navigate to the **/apps/training/components/structure** folder.
26. Right-click the **footer** node, and click **Create** > **Create Node** The **Create Node** dialog box opens.
27. Provide the following details:
    a. **Name**: **cq:design_dialog**
    b. **Type: nt:unstructured**.
28. Click **OK**. The node is created.
29. Click **Save All**.
30. Select the **cq:design_dialog** node and add the following properties:

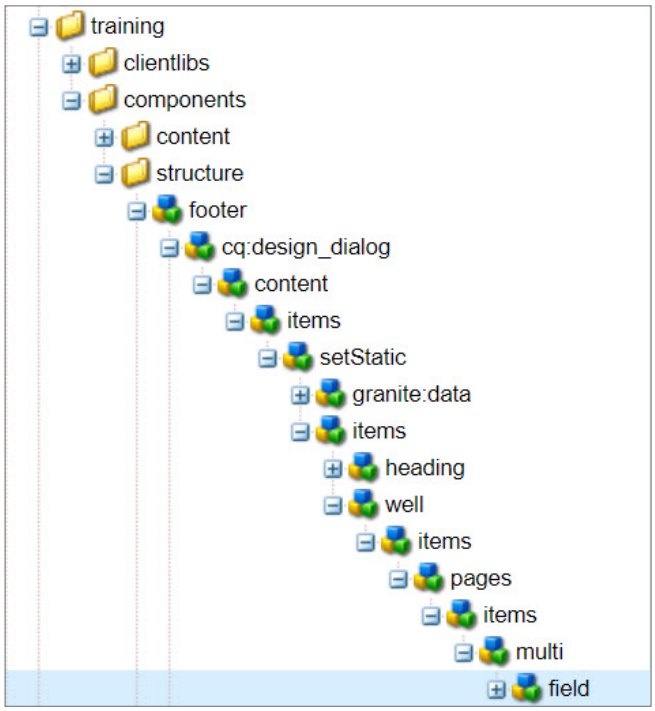
Name	Type	Value
sling:resourceType	String	cq/gui/components/authoring/dialog
jcr:title	String	Footer Policy

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

To remove all the extra fields from the dialog and customize it to your footer component.

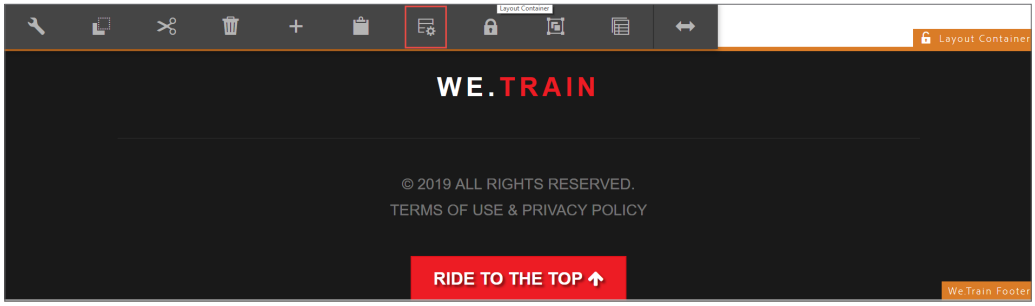
35. Navigate to the **/apps/training/components/structure/footer/cq:design_dialog/content/items/tabs/items** node, right-click the **itemSettings** node, and click **Delete**. The node is deleted.
36. Click **Save All**.
37. Navigate to the **/apps/training/components/structure/footer/cq:design_dialog/content/items/tabs/items/listSettings/items/columns/items/column/items** node.
38. Right-click the **setStatic** node, and click **Copy**.
39. Navigate to the /**apps/training/components/structure/footer/cq:design_dialog/content/items** node and paste the node you copied in the previous step.
40. Click **Save All**.
41. Navigate to the **/apps/training/components/structure/footer/cq:design_dialog/content/items/setStatic** node.
42. On the **Properties** tab, right-click the **granite:class** property, and click **Delete**. The property is deleted from the node.
43. Click **Save All**.
44. Navigate to the /**apps/training/components/structure/footer/cq:design_dialog/content/items** node.
45. Right-click the **tabs** node, and click **Delete**.

46. Navigate to the **/apps/training/components/structure/footer/cq:design_dialog/content/items/setStatic/items/well/items/pages/items/multi/field** node. Notice that the property name, ./pages holds the fixed list. Your folder structure should look similar to the one shown in the below screenshot:



Now that the design dialog is built, test it on the page.

47. 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.

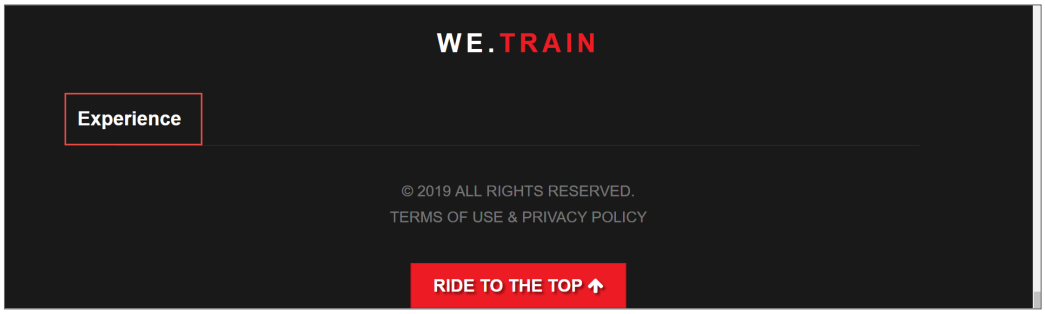


48. In the **Policy** section, in the **Policy Title** field, type **We.Train Footer Policy**.

49. In the **Properties** section, under **Options for Fixed List**, click **Add**, and type **/content/we-retail/us/en/experience**.

50. Click **Done** (checkmark icon).

51. 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.



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

53. 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, will affect all pages in the site using that policy.

References

Use the following link for more information on:

- Structure of the component:

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
- Add a selector to the custom component
- Create localization information
- Extend the core component
- Extend the AEM navigation UI

Features of Components

Some common component features are:

- **Component Placeholder**: Helps validate if the component is on the page and is available only in the page Edit mode. The placeholder is not available in the page Preview mode or on the publish instance. 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 the English and translated texts. You can also export dictionaries to XLIFF format for translating, and then import the translations back into the dictionaries. It is also possible to add the i18n dictionaries to a translation project from 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, 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 Component**. The **Create Component** dialog box opens.
4. Enter the following details in the dialog box:
   a. **Label**: **stockplex**
   b. **Title**: **StockPlex**
   c. **Description**: We.Train Complex Stock Component
   d. **Group**: **We.Train**
5. Click **Next**.
6. Keep the **Advanced Component Settings** as is and click **OK**. The component is created.
7. Click **Save All** from the actions bar.
8. Expand the stockplex component and double-click **stockplex.jsp**. The editor opens on the right.
9. Delete the existing content and click **Save All**.
10. Right-click **stockplex.jsp**, select **Rename**, and change the file name to **stockplex.html**.
11. Click **Save All**.

12. Double-click **stockplex.html** to open the script for editing.
13. Replace the existing code with code from **placeholder_stockplex.html** in the **Creating Custom Components in Adobe Experience Manager** exercise folder.

```
1.  <div class="cmp-stockplex"
2.      data-sly-use.template="core/wcm/components/commons/v1/templates.html"
3.      data-sly-test.symbol="${properties.symbol}" >
4.  </div>
5.
6.  <!-- If there is no stock symbol added to the dialog, create a compo-
    nent placeholder -->
7.  <sly data-sly-call="${template.placehold-
    er @ isEmpty=!symbol, classAppend='cmp-stockplex'}"></sly>
```

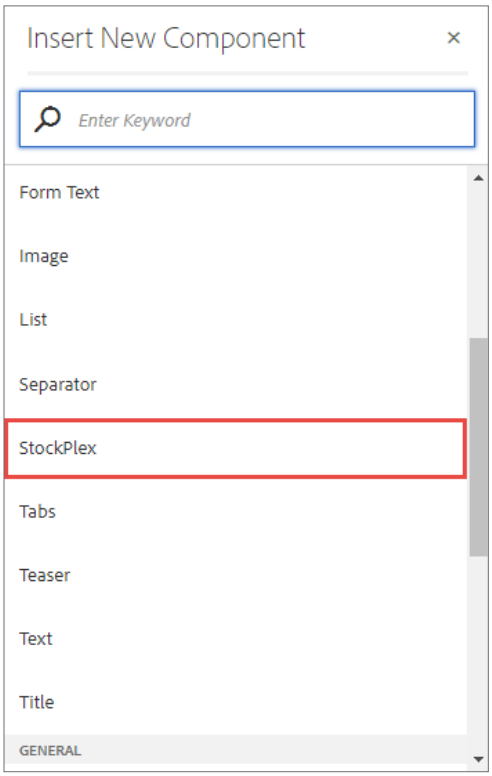
14. Click **Save All**.

You must create a cq:dialog node to enable the component delete functionality and to provide a means for the author to enter content.

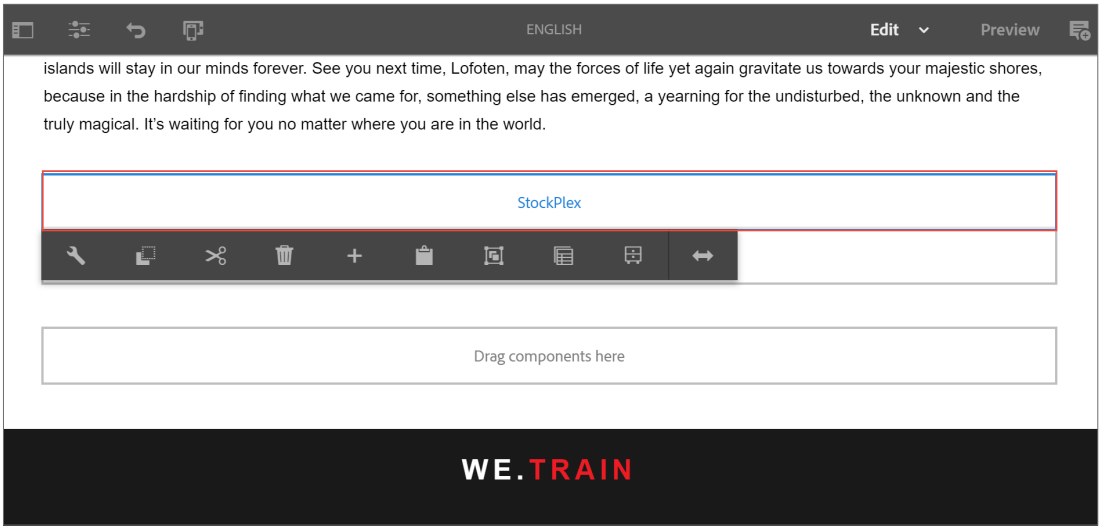
15. Right-click the **stockplex** component, and click **Create** > **Create Node**. The **Create Node** dialog box opens.
16. Update the following details:
    a. **Name**: **cq:dialog**
    b. **Type**: **nt:unstructured**
17. Click **OK**. The node is created.
18. Click **Save All**.
19. Open The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

20. At the bottom of the page, click the **Drag components here** placeholder, and click the **Insert component** icon from the component toolbar. The **Insert New Component** dialog box opens.

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

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.

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. Update the **stockplex.html** script by replacing the existing code with new code from **dialog_stockplex.html** located in the **Creating Custom Components in Adobe Experience Manager** exercise folder.

5. Open the **stockplex.html** to 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. Now, create a dialog to allow entry of that data by the author..

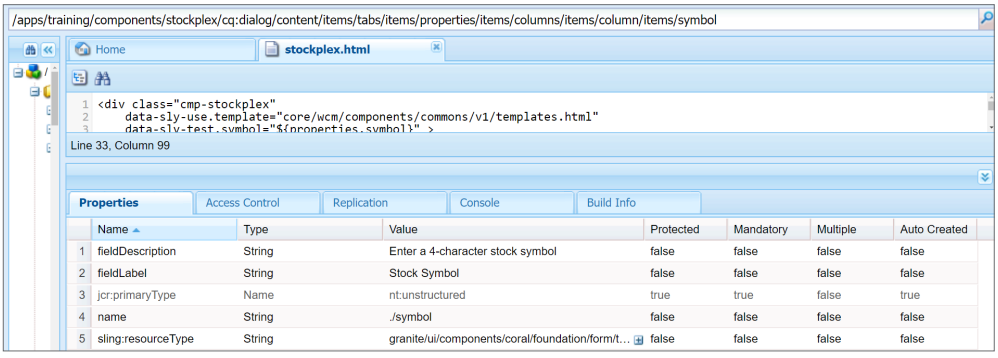
```
1.  <div class="cmp-stockplex"
2.    data-sly-use.template="core/wcm/components/commons/v1/templates.html"
3.    data-sly-test.symbol="${properties.symbol}" >
4.    <div class="cmp-stockplex__symbol">${properties.symbol}</div>
5.    <div class="cmp-stockplex__currentPrice">Current Value: 300</div>
6.     <div data-sly-test.summary="${properties.summary}">
7.       <h3>Summary: ${properties.summary}</h3>
8.     </div>
9.     <div class="cmp-stockplex__button">
10.      <a href="#">
11.       <button>Placeholder</button>
12.      </a>
13.    </div>
14.     <div class="cmp-stockplex__details" data-sly-test.summary="${properties.
    showStockDetails}">
15.      <ul class="column1">
16.        <li>Request Date: November 13, 2018</li>
17.        <li>Open Price: 300</li>
18.        <li>Range High: 303</li>
19.      </ul>
20.      <ul class="column2">
21.        <li>Range Low: 299</li>
22.        <li>Close: 303</li>
23.        <li>Volume: 30000</li>
24.      </ul>
25.    </div>
26. </div>
27. <!-- If there is no stock symbol added to the dialog, create a compo-
    nent placeholder -->
28. <sly data-sly-call="${template.placehold-
    er @ isEmpty=!symbol, classAppend='cmp-stockplex'}"></sly>
```

6. Click **Save All**.

To create a dialog:

7. Select the **cq:dialog** node that is below the **stockplex** component.
8. Right-click the **cq:dialog** node and click **Delete**. The node is deleted.
9. Click **Save All** from the actions bar.
10. Navigate to the **/apps/core/wcm/components/title/v2/title/cq:dialog** node, right-click the **cq:dialog** node, and click **Copy**.
11. Navigate to the **/apps/training/components/content** folder, right-click the **stockplex** node, and click **Paste**.
12. 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.
13. Click **Save All** from the actions bar.
14. Navigate to the **/apps/training/components/content/stockplex/cq:dialog/content/items/tabs/items/properties/items/columns/items/column/items** node.
15. Right-click the **types** node and click **Delete** to delete it.
16. Click **Save All** from the actions bar.
17. Right-click the **defaulttypes** node and click **Delete** to delete it.
18. Navigate to the **/apps/training/components/content/stockplex/cq:dialog/content/items/tabs/items/properties/items/columns/items/column/items/title** node.
19. Right-click the **title** node, click **Rename,** and rename it as **symbol**.
20. Update the following properties for the **symbol** node, as shown:

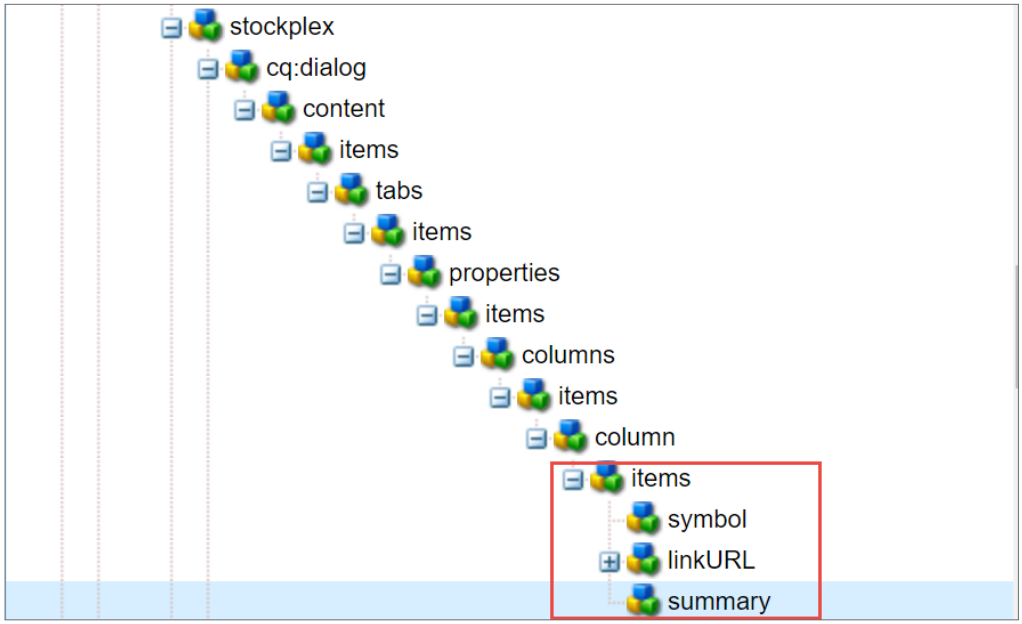
Name	Type	Value
name	String	./symbol
fieldLabel	String	Stock Symbol
fieldDescription	String	Enter a 4-character stock symbol



21. Right-click the **symbol** node, and click **Copy**.

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

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



24. Update the following properties of the **summary** node:

Name	Type	Value
name	String	./summary
fieldLabel	String	Summary of Stock
fieldDescription	String	Enter a summary description of the stock

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

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

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

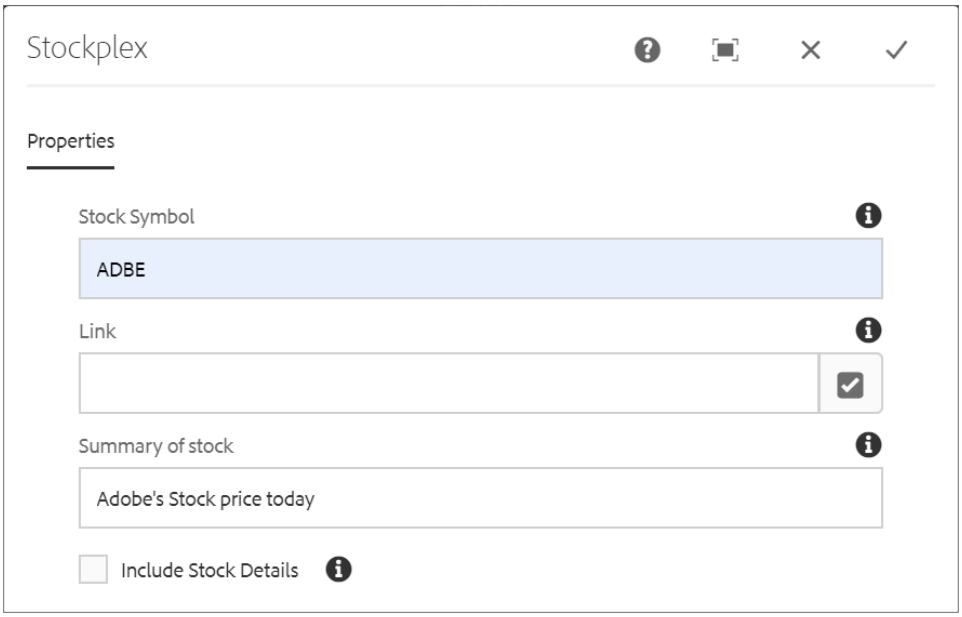
28. Update the following properties of the **stockdetails** node:

Name	Type	Value
name	String	./showStockDetails
fieldDescription	String	Include requestDate, upDown, openPrice, range high/low, volume, and others
sling:resourceType	String	granite/ui/components/coral/foundation/form/checkbox

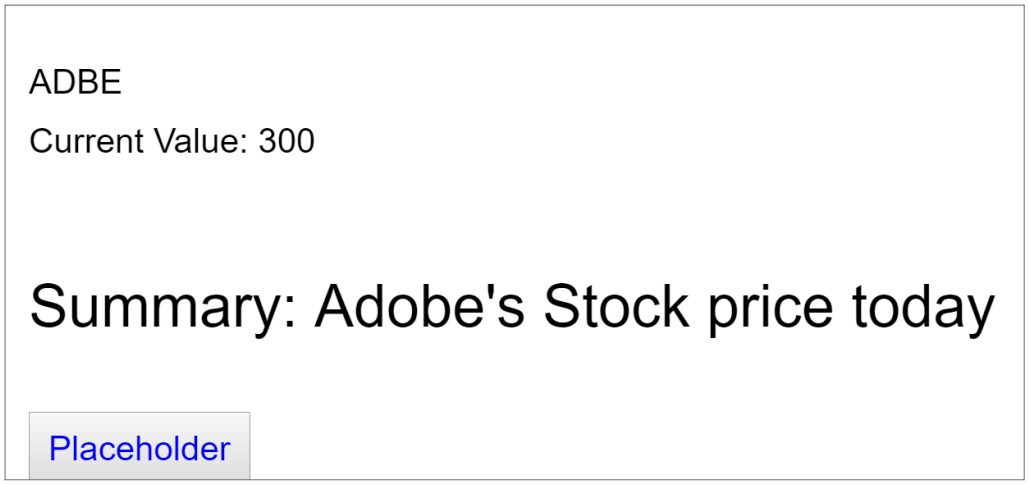
29. Add the following new properties to the stockdetails node:

Name	Type	Value
text	String	Include Stock Details
value	Boolean	true

30. Select the **fieldLabel** property, right-click it, and click **Delete** from the list.
31. Click **Save All** from the actions bar.
32. Navigate to the tab where the English page is open or click http://localhost:4502/editor.html/content/we-train/en.html
33. Select the **stockplex** component you added in a previous task, and click the **Configure** (wrench) icon. The **Stockplex** dialog box opens.
34. Provide the following values, as shown:
    a. **Stock Symbol**: **ADBE**
    b. **Summary of Stock**: **Adobe's Stock price today**

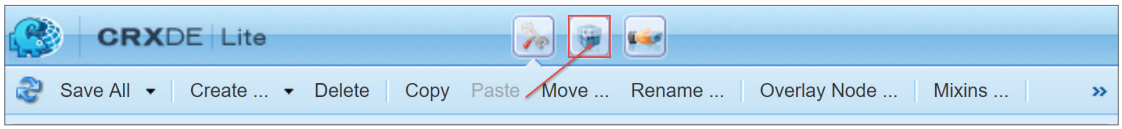


35. Click **Done** (checkmark). You can view the content rendered on the stockplex component in the English page.

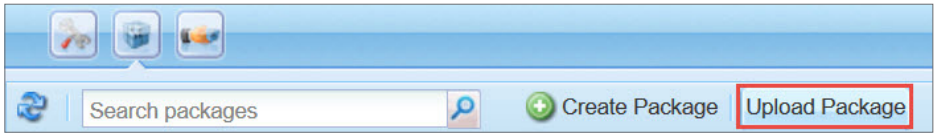


Now that you have a basic dialog with three form fields, you need to create validation of the stock symbol. As the symbol description suggests, the user must enter a four letter symbol. You can accomplish this by including client-side JavaScript with a client library.

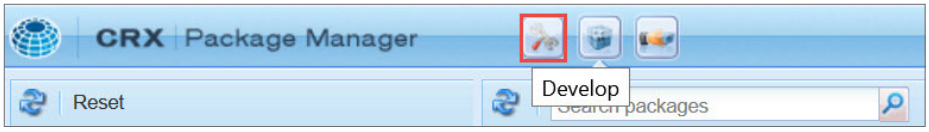
36. Navigate back to CRXDE Lite page or click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.
37. Click the **Package** icon from the header bar, as shown. The **CRX Package Manage**r page opens.



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

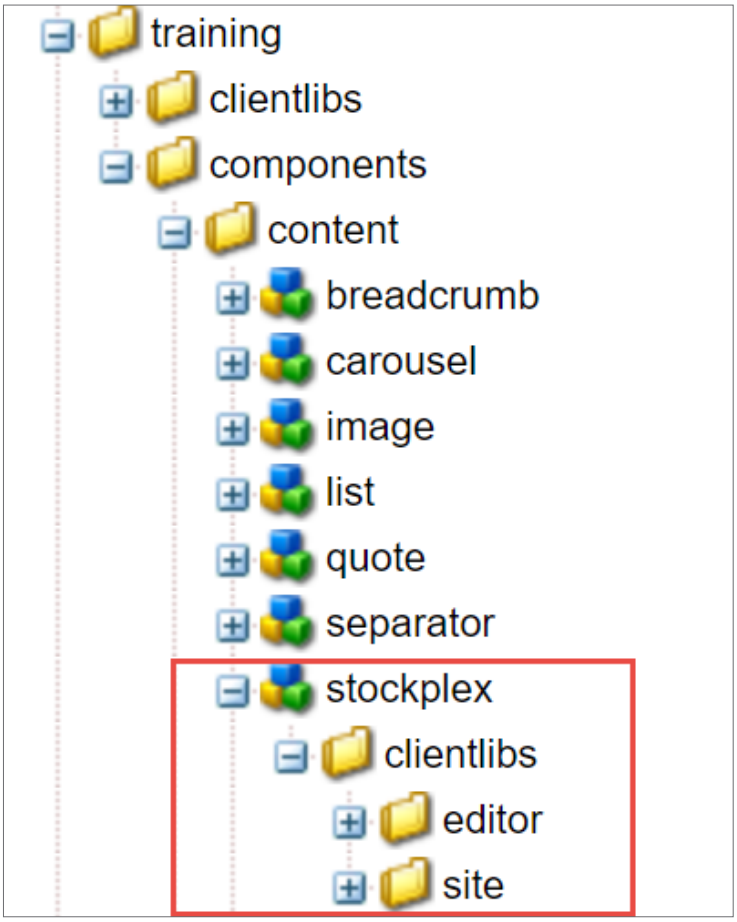


39. Click **Browse** in the dialog box. The **Open** dialog box opens.
40. Navigate to the exercise folder provided to you, and upload the **stockplex-clientlibs-all.zip** package.
41. Click **OK** in the **Upload Package** dialog box. The package is uploaded.
42. Click **Install** from the actions bar in the **stockplex-clientlibs-all.zip** package section. The **Install Package** dialog box opens.
43. Click **Install**. The package is installed.
44. Click the **Develop** icon from the header bar, as shown. The **CRXDE Lite** page opens.



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

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

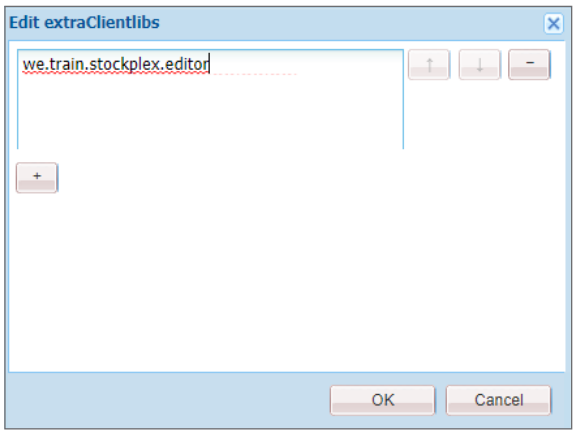


To use the stockplex editor client library, you must add it to the dialog box. If you want to define styling and behavior for your component, you can create a dedicated client library that defines your custom CSS/LESS and JS.

To have your client library loaded solely for your component dialog (that is, it will not be loaded for another component), you must set the property extraClientLibs ** of your dialog to the category name of the client library you just 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.

46. Navigate to the **/apps/training/components/content/stockplex/cq:dialog** node.
47. In **Properties** section, double-click in the **extraClientlibs**. The **Edit extraClientlibs** dialog box opens.

48. Update the value to **we.train.stockplex.editor**, as shown, and click **OK**. The existing value is replaced with the new value.

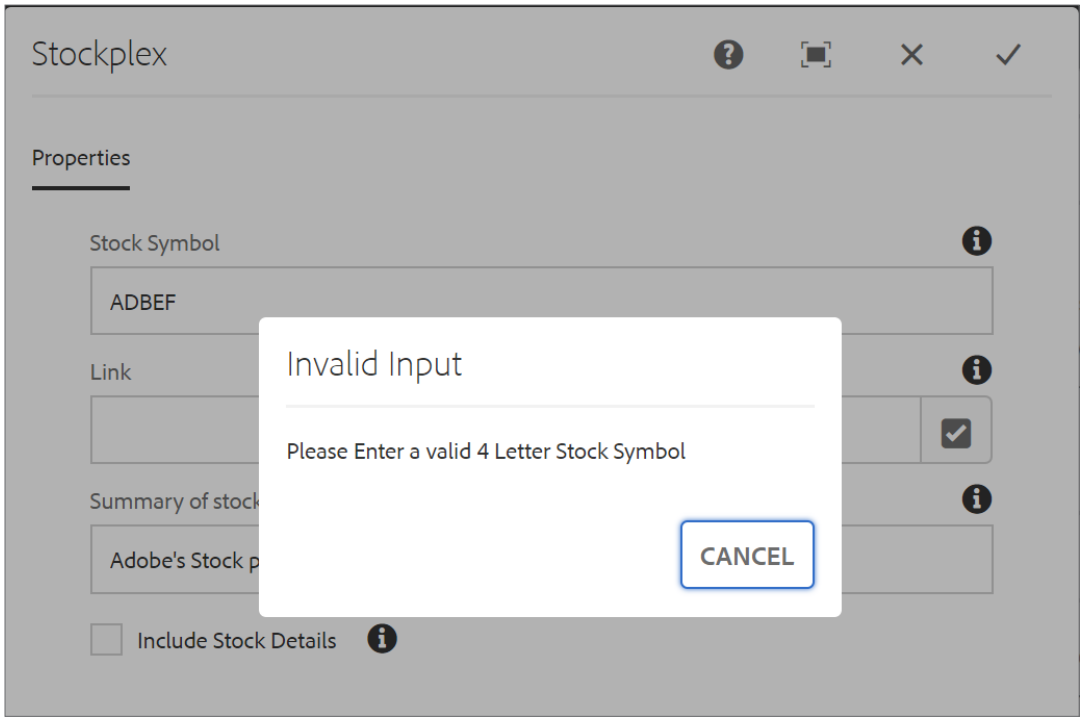


49. Click **Save All** from the actions bar.

50. Navigate to the tab where the **English** page is open.

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

52. In the **Stock Symbol** field, type more than four characters, and then click the **Done** (checkmark) icon. 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.



53. Click **Cancel** to close the dialog box, and then click the **Cancel** (X) icon in the Stockplex dialog box to discard your changes.

Exercise 3: Add a base style to the custom component

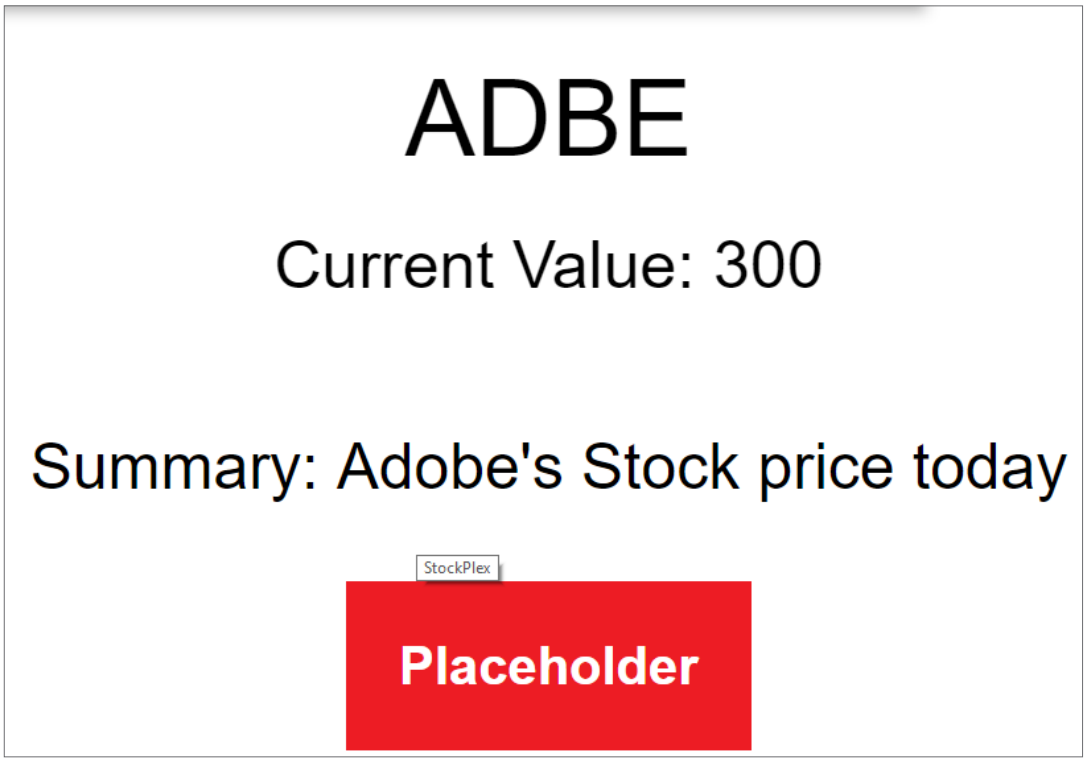
In the previous exercise, you installed the client libraries for the stockplex component. In this exercise, you will add a design to the stockplex component.

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. Select the stockplex node and double-click the **stockplex.html** to open the script for editing.
5. Update the **stockplex.html** script by replacing the existing code with the new code in **clientlibs_stockplex.html** located in the **Creating Custom Components in Adobe Experience Manager** exercise folder.
6. Click **Save All**.

7. 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 pre-created 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:

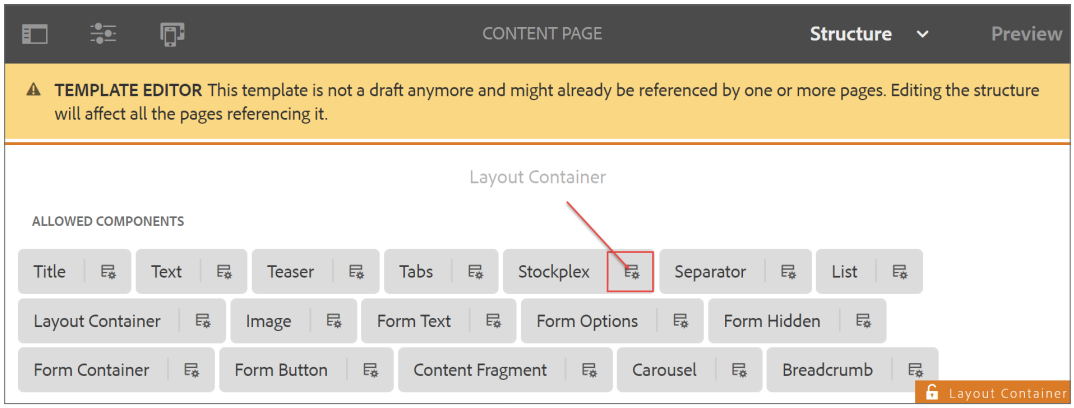
2. Navigate to the **/apps/training/components/content/stockplex/clientlibs/site/css/stockplex.css** node.
3. Double-click **stockplex.css** and notice **cmp-stockplex--italic**, **cmp-stockplex--bold**, and **cmp-stockplex--red** in the editor. These classes are not used in the base stockplex code and must be added to 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.

4. 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.
5. Navigate to the **/apps/training/components/content/stockplex** node, and click **Paste** to paste the node. The node is added to the stockplex component.
6. Click **Save All** from the actions bar.
7. Select the **cq:design_dialog** node, and on the **Properties** tab, update the **jcr:title** property to **Stockplex**.
8. Click **Save All** from the actions bar.
9. 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.
10. Click **Save All** from the actions bar.

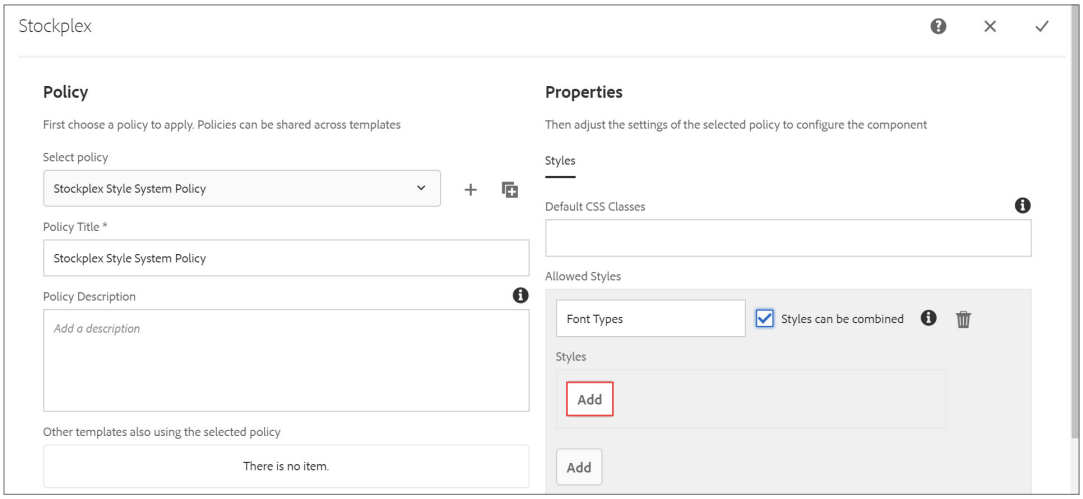
Now, as a template author, you will add the style system to the stockplex content policy.

11. Click **CRXDE Lite** from the header bar. The **Navigation** page opens.
12. Click the **Tools** (hammer) icon. The **Tools** console opens.
13. Ensure you are in the **General** section and click the **Templates** tile. The **Templates** console opens.
14. 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.
15. In the **Layout Container** component, click the **Policy** icon next to the **Stockplex** component, as shown. The **Stockplex** policy wizard opens.



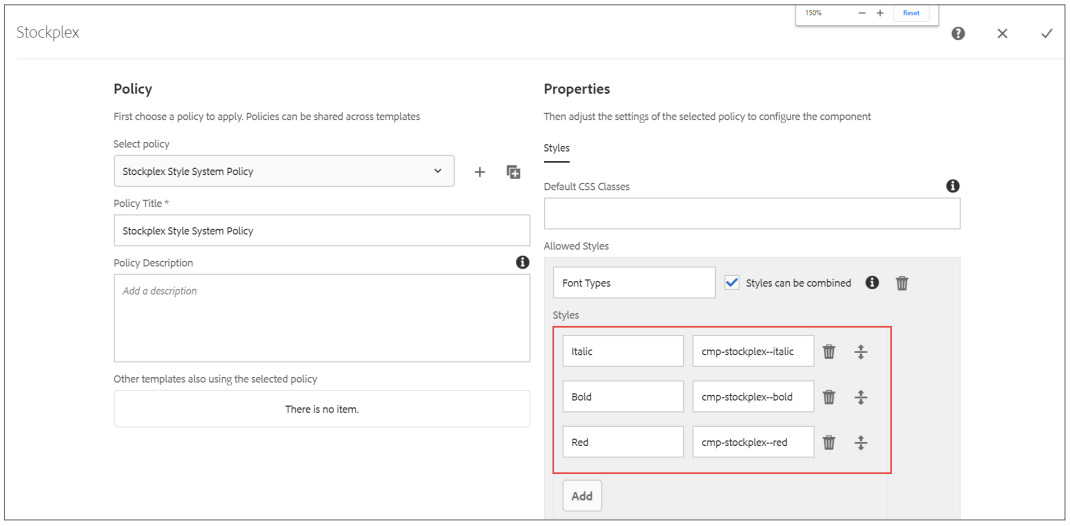
16. In the **Policy** section, type **Stockplex Style System Policy** in the **Policy Title** field.
17. In the **Properties** section, under the **Styles** tab, click the **Add** button below the **Allowed Styles** field.

18. In the **Group Name** field, type **Font Types**, and select the **Styles can be combined** checkbox.
19. Click the **Add** button that is below the **Styles** field, as shown. A new style field is created.



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

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

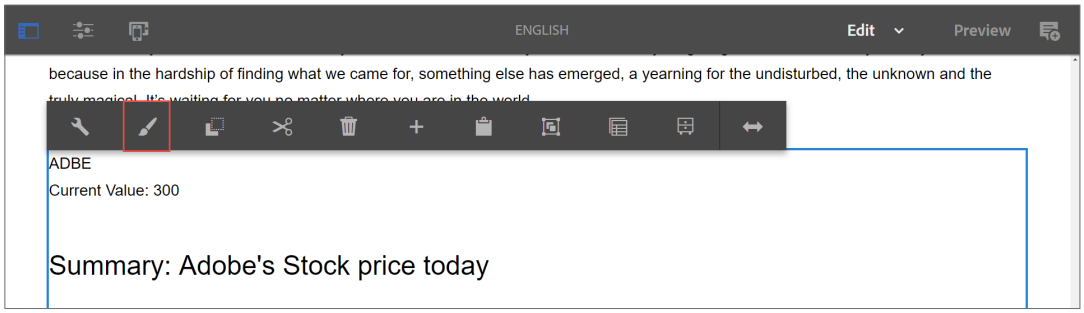


24. Click the **Done** (checkmark) icon in the Stockplex wizard. The **Content Page** template editor opens.

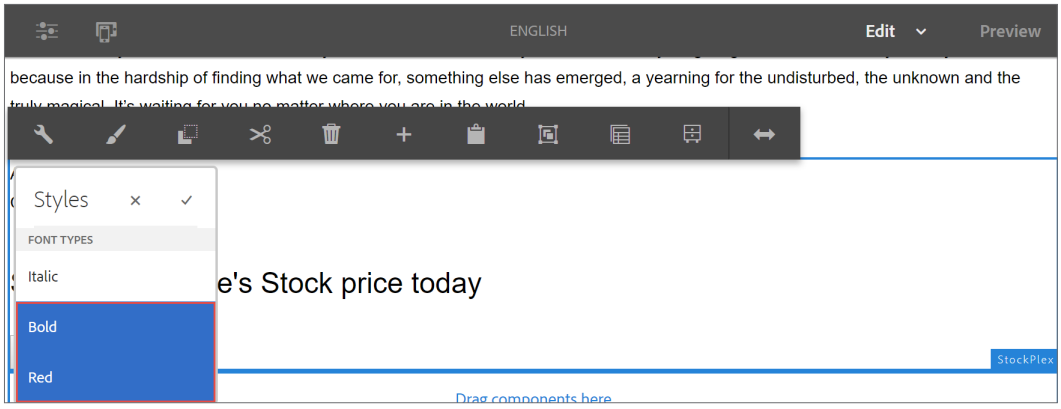
Now that you have the CSS developed by the designer, and the style system is added by the template editor, authors can use the style system.

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

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



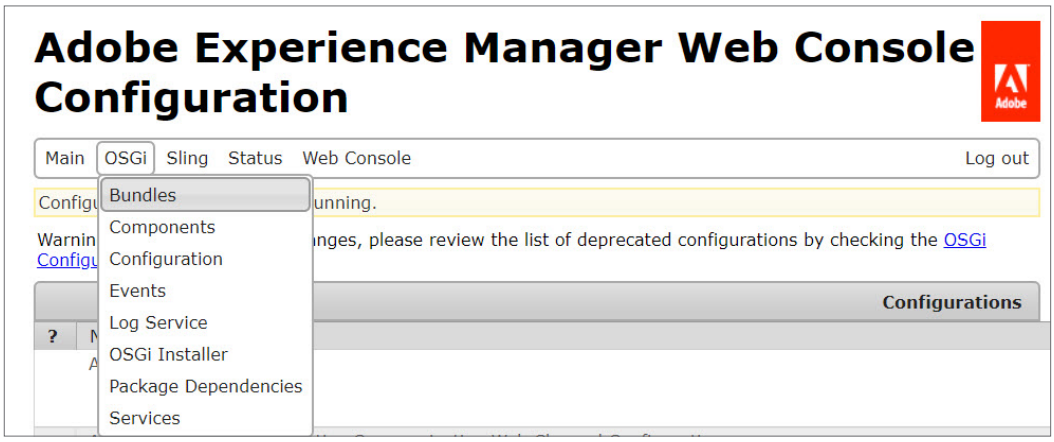
27. Click the **Styles** (paintbrush) icon from the actions bar, and click **Italic**, **Bold**, or **Red**, and observe the changes. You have now modified your component's script to use the CSS styles from the component's client library. After modifying, you noticed changes in the appearance of the component. If you are unable to view any changes to the styles in the stockplex component, refresh the English page.



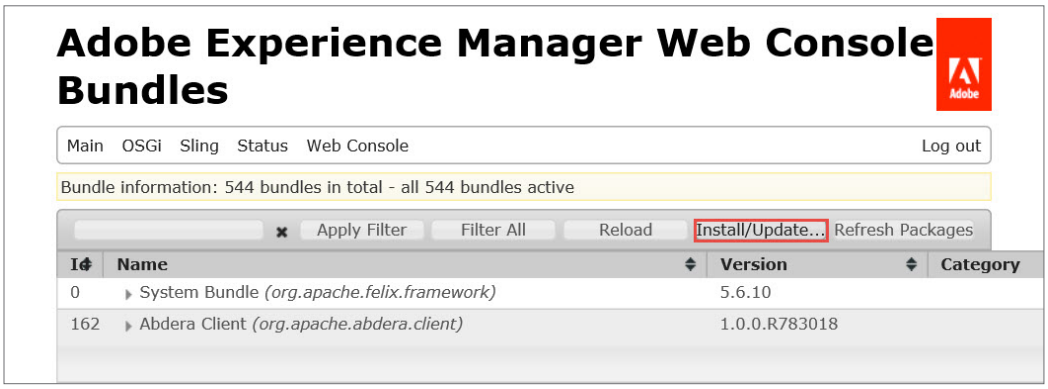
Exercise 5: Add a Sling Model as the business logic

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.

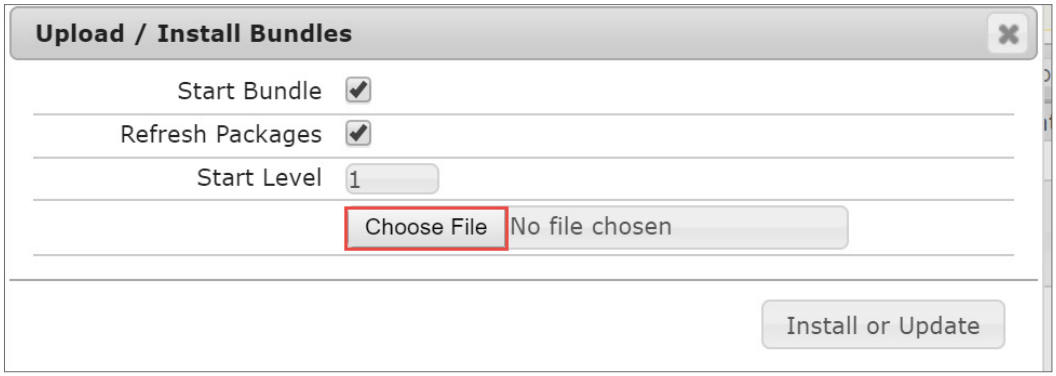
1. The Sling Model you will install is called Stockplex.java and is compiled into an OSGi bundle. Observe the raw java code in your exercise files.
2. In your AEM author instance, click **Adobe Experience Manager** from the header bar, and click the **Tools** icon. The **Tools** console opens.
3. Click **Operations** > **Web Console**. The **Adobe Experience Manager Web Console Configuration** page opens.
4. Click **OSGi** from the header bar, and click **Bundles** from the drop-down menu, as shown. The **Adobe Experience Manager Web Console Bundles** page opens.



5. Click **Install/Update** from the actions bar, as shown. The **Upload / Install Bundles** dialog box opens.



6. Select the **Start Bundle** and **Refresh Packages** checkboxes, enter **1** in the **Start Level** field, and click **Choose File**, as shown. The **Open** dialog box appears.



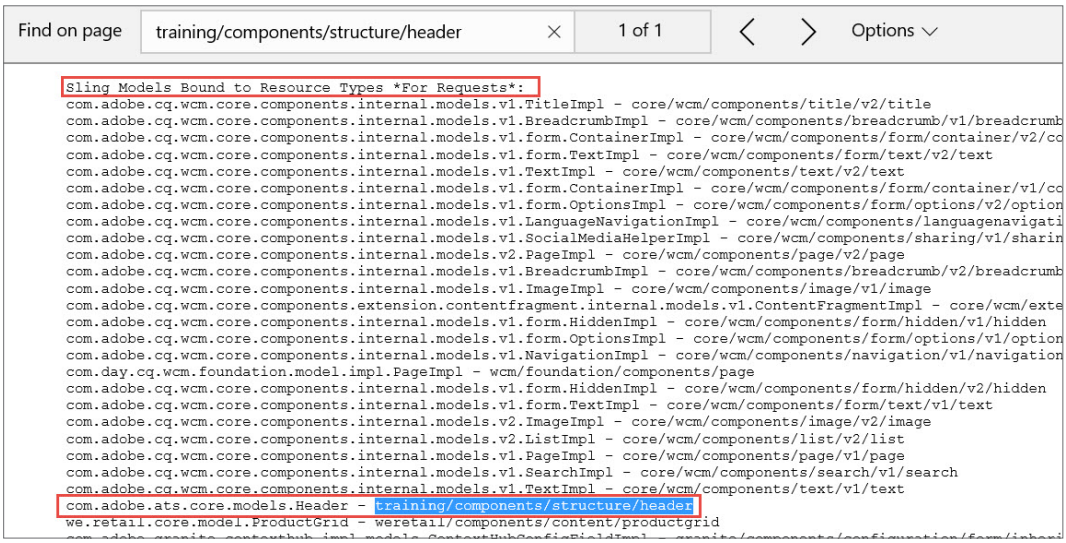
7. Navigate to the **Creating Custom Components in Adobe Experience Manager** exercise folder on your file system and double-click the **training.core-0.0.1-SNAPSHOT.jar file**. The file is selected.

8. Click **Install or Update** in the **Upload / Install Bundles** dialog box. The **Adobe Experience Manager Web Console Bundles** page refreshes when the installation is complete.

To verify if the bundle is installed successfully on your instance:

9. 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.

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.ats.core.models.Header - training/components/structure/header** is under **Sling Models Bound to Resource Types *For Requests** (if you cannot find this, use Ctrl + F (Windows) or Command + F (Mac). This confirms the bundle is installed successfully.



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

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

12. Navigate to the **/apps/training/components/content** node.

13. Select the **stockplex** node, and double-click **stockplex.html** to open the script for editing.

14. Update the **stockplex.html** script by replacing the existing code with the new code in **slingmodel_stockplex.html** from the **exercise files** provided to you.

15. Click **Save All** from the actions bar. Review the code you added. Notice that the `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.

```
<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}:</span>
            <br />
```

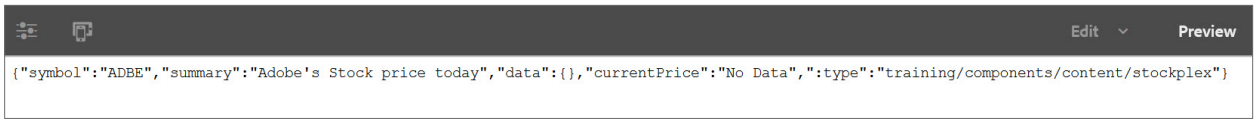
```
30.             <span class="cmp-stockplex__details-data">${stockplex.data[dataKey]}</span>
31.           </li>
32.         </ul>
33.       </div>
34.     </div>
35. </div>
36.
37. <!-- If there is no stock symbol added to the dialog, create a component placehold-
    er -->
38. <sly data-sly-call="${template.placeholder @ isEmpty=!symbol, classAppend='cmp-stock-
    plex'}"></sly>
```

To observe the JSON output:

16. 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.
17. 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:

18. 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.



19. Additionally, you can view the stockplex JSON apart of the entire page's JSON output. 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: Add live data to the Stockplex Component

In this exercise, you will add a data source to the Stockplex component. You must add back-end business logic to successfully import data into JCR and then be able to view that on the stockplex component.

1. Install **stockplex-backend.zip** package from the exercise files provided to you. The package creates:

- An OSGi config for ADBE
- An OSGi config for MSFT
- A service user (training-user)
- A OSGi config for a Service User Mapper

2. Install the **training.core-0.0.1-SNAPSHOT.jar** OSGi bundle from the exercise files. This installs a bundle with StockImportScheduler and Stockplex.java.
3. In CRXDE lite, navigate to **/apps/training/trainingproject/config**. You should see two nodes. One specific for Adobe stock information and another for Microsoft. These nodes run every 30 seconds. That is, every 30 seconds the content is imported. To achieve this the StockImportScheduler uses service user (line 171). To create the service user, a third configuration called ServiceUserMapper is required. This configuration maps service user to your project.
4. In the user admin page (http://localhost:4502/useradmin), provide permissions to training user.
   a. Search for the **training-user** that was automatically installed.
   b. Double-click **training-user** and provide complete access to the user by selecting all the boxes at the top—Read, Modify, Create, Delete, Read ACL, Edit ACL, and Replicate.
   c. Click **Save**.
5. Wait for a few seconds, navigate back to CRXDE Lite, and refresh **/content**. You should see two nodes under /content/stocks for Adobe and Microsoft. All the stock information is added now.
6. To use the back-end data, open the **We.Train** > **English** page.
7. Add the Stockplex component to the page.
8. Update the Symbol field to ADBE in the dialog. Observe the real stock data for ADBE.

Exercise 6: Add a selector script to a component

Let us look back at Sling URL decomposition, resource resolution, and how Sling finds the rendering script. In this exercise, you will add print.html script to the page node, and observe how Sling will first attempt to find a script to match the selector.

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/structure** folder.
3. Right-click the **page** node and click **Create** > **Create File**. The **Create File** dialog box opens.
4. Type **print.html** in the **Name** field, and click **OK**. The file is created.
5. Click **Save All** from the actions bar.
6. Double-click **print.html**. The editor opens on the right.
7. Navigate to the exercise files provided to you, copy the code from **print.html**, and paste it in the CRXDE Lite editor.
8. Click **Save All** from the actions bar.
9. Open the **print.html** to view the code that you added. Observe the following code snippet:

```
1. WE.<strong>TRAIN</strong>: ${currentPage.title} Page Print
2. <hr />
3.
4. <!--/* Include all content components for printing */-->
5. <div  data-sly-resource="${'root/responsivegrid' @ resourceType='wcm/founda-
   tion/components/responsivegrid'}"></div>
```

You must now update the footer component to call the print script with a selector.

10. Navigate to the **/apps/training/components/structure/footer/footer.html** file.
11. Double-click **footer.html**. The editor opens on the right.
12. Replace the code in footer.html with code from **print_footer.html** located in the exercise files provided to you.
13. Click **Save All** from the actions bar.

14. Open the **footer.html** to view the code that you added. Observe the following code snippet:

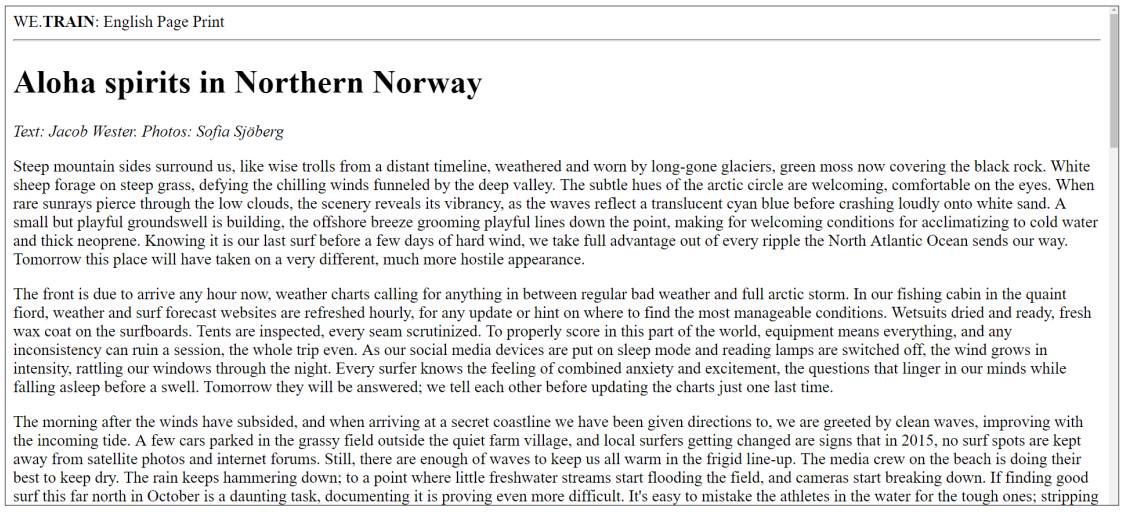
```
<!--/* Inserts a print selector into the request so that the page/print.
html script can be run. */-->
<a href="${currentPage.Path @ selectors='print', extension='html'}" class="btn btn-pri-
mary btn-print">
  ${ 'Print' @ i18n } <i class="fa fa-print" aria-hidden="true"></i>
</a>

<sly data-sly-use.footer="com.adobe.ats.core.models.Footer">
  <footer class="we-Footer">
    <div class="container">

      <div class="row">

        <div class="col-md-4">
          <div class="we-Logo we-Logo--big">
            we.<strong>Train</strong>
          </div>
        </div>

        <div class="row col-md-8 col-xs-12 we-footer-links">
          <sly data-sly-list.item="${footer.items}">
            <div class="col-lg-2 col-md-2 col-xs-3">
              <div class="we-Footer-nav">
                <h2 class="h4">
                  <a href="${ item.path @ extension = 'html'}">${item.title || item.
name}</a>
                </h2>
              </div>
            </div>
          </sly>
        </div>

      </div>

```

```
33.         <div class="row we-Footer-section we-Footer-section--sub">
34.           <div class="we-Footer-section-item">
35.             <span class="text-uppercase text-muted">${ 'Â© {0} All rights reserved.' @ i18n, format=[footer.currentYear], context="html"}</span>
36.           </div>
37.           <div class="we-Footer-section-item">
38.             <a href="#" class="text-uppercase text-muted">${ 'Terms of use & privacy policy' @ i18n }</a>
39.           </div>
40.         </div>
41.
42.         <div class="row">
43.           <div class="col-md-12">
44.             <div class="text-center">
45.               <a href="#top" class="btn btn-primary">
46.               ${ 'Ride to the top' @ i18n }
47.               <i class="fa fa-arrow-up" aria-hidden="true"></i>
48.               </a>
49.             </div>
50.           </div>
51.         </div>
52.
53.       </div>
54.     </footer>
55. </sly>
```

To observe the changes:

15. Navigate to the **English** page of the We.Train site. Notice how the **Print** button is added to the bottom of the page before the footer component.

16. Click **Preview** from the page toolbar, and click the **Print** button. The page opens on the same tab, as shown. You provided an alternate, print rendering of your page by adding a Sling selector script named print.html.


WE.**TRAIN**: English Page Print

Aloha spirits in Northern Norway

Text: Jacob Wester. Photos: Sofia Sjöberg

Steep mountain sides surround us, like wise trolls from a distant timeline, weathered and worn by long-gone glaciers, green moss now covering the black rock. White sheep forage on steep grass, defying the chilling winds funneled by the deep valley. The subtle hues of the arctic circle are welcoming, comfortable on the eyes. When rare sunrays pierce through the low clouds, the scenery reveals its vibrancy, as the waves reflect a translucent cyan blue before crashing loudly onto white sand. A small but playful groundswell is building, the offshore breeze grooming playful lines down the point, making for welcoming conditions for acclimatizing to cold water and thick neoprene. Knowing it is our last surf before a few days of hard wind, we take full advantage out of every ripple the North Atlantic Ocean sends our way. Tomorrow this place will have taken on a very different, much more hostile appearance.

The front is due to arrive any hour now, weather charts calling for anything in between regular bad weather and full arctic storm. In our fishing cabin in the quaint fiord, weather and surf forecast websites are refreshed hourly, for any update or hint on where to find the most manageable conditions. Wetsuits dried and ready, fresh wax coat on the surfboards. Tents are inspected, every seam scrutinized. To properly score in this part of the world, equipment means everything, and any inconsistency can ruin a session, the whole trip even. As our social media devices are put on sleep mode and reading lamps are switched off, the wind grows in intensity, rattling our windows through the night. Every surfer knows the feeling of combined anxiety and excitement, the questions that linger in our minds while falling asleep before a swell. Tomorrow they will be answered; we tell each other before updating the charts just one last time.

The morning after the winds have subsided, and when arriving at a secret coastline we have been given directions to, we are greeted by clean waves, improving with the incoming tide. A few cars parked in the grassy field outside the quiet farm village, and local surfers getting changed are signs that in 2015, no surf spots are kept away from satellite photos and internet forums. Still, there are enough of waves to keep us all warm in the frigid line-up. The media crew on the beach is doing their best to keep dry. The rain keeps hammering down; to a point where little freshwater streams start flooding the field, and cameras start breaking down. If finding good surf this far north in October is a daunting task, documenting it is proving even more difficult. It's easy to mistake the athletes in the water for the tough ones; stripping

Exercise 7: Create the localization information

In this exercise, you will create localized content. In addition, you will look at HTL code that specifies the internationalization process.

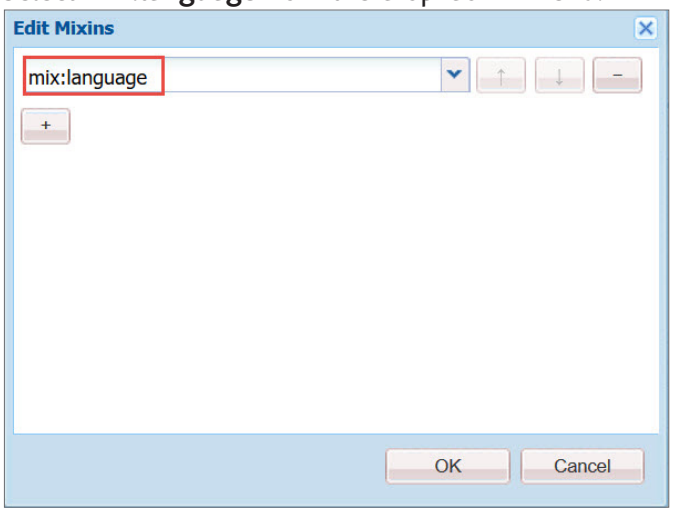
1. In CRXDE Lite, navigate to the **/apps/training/components/structure/footer** node.
2. Double-click the **footer.html** to open the script for editing. In the code snippet, observe the i18n content:

```
1.  <!--/* Inserts a print selector into the request so that the page/print.
    html script can be run. */-->
2.  <a href="${currentPage.Path @ selectors='print', extension='html'}" class="btn btn-pri-
    mary btn-print">
3.    ${ 'Print' @ i18n } <i class="fa fa-print" aria-hidden="true"></i>
4.  </a>
5.
6.  <sly data-sly-use.footer="com.adobe.ats.core.models.Footer">
7.    <footer class="we-Footer">
8.      <div class="container">
9.
10.       <div class="row">
11.
12.         <div class="col-md-4">
13.           <div class="we-Logo we-Logo--big">
14.             we.<strong>Train</strong>
15.           </div>
16.         </div>
17.
18.         <div class="row col-md-8 col-xs-12 we-footer-links">
19.           <sly data-sly-list.item="${footer.items}">
20.             <div class="col-lg-2 col-md-2 col-xs-3">
21.               <div class="we-Footer-nav">
```

```
22.                     <h2 class="h4">
23.                       <a href="${ item.path @ extension = 'html'}">${item.title || item.
    name}</a>
24.                     </h2>
25.                   </div>
26.                 </div>
27.               </sly>
28.             </div>
29.
30.
31.         </div>
32.
33.         <div class="row we-Footer-section we-Footer-section--sub">
34.           <div class="we-Footer-section-item">
35.             <span class="text-uppercase text-muted">${ '© {0} All rights re-
    served.' @ i18n, format=[footer.currentYear], context="html"}</span>
36.           </div>
37.           <div class="we-Footer-section-item">
38.             <a href="#" class="text-uppercase text-muted">${ 'Terms of use & privacy poli-
    cy' @ i18n }</a>
39.           </div>
40.         </div>
41.
42.         <div class="row">
43.           <div class="col-md-12">
44.             <div class="text-center">
45.               <a href="#top" class="btn btn-primary">
46.               ${ 'Ride to the top' @ i18n }
47.               <i class="fa fa-arrow-up" aria-hidden="true"></i>
48.               </a>
49.             </div>
50.           </div>
51.         </div>
52.
53.       </div>
54.     </footer>
```

To internationalize the copyright statement on the footer of a website.

3. Navigate to the **/apps/training** folder, and create a new folder by clicking **Create** > **Create Folder**. The **Create Folder** dialog box opens.
4. Enter **i18n** in the **Name** field, and click **OK**. The folder is created.
5. Click **Save All** from the actions bar.
6. Right-click the **i18n** folder, and click **Create** > **Create File**. A new file is created.
7. Name it **fr.json** and click **OK**.
8. Click **Save All** from the actions bar.
9. Select the **fr.json** file, and click **Mixins** from the actions bar. The **Edit Mixins** dialog box opens.
10. Click the + icon in the **Edit Mixins** dialog box.
11. Select **mix:language** from the drop-down menu:



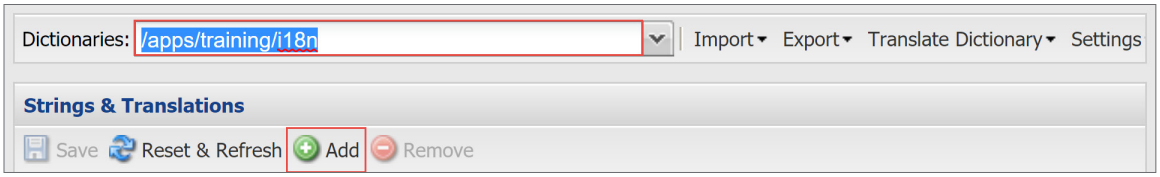
12. Click **OK**.
13. Click **Save All** from the actions bar.
14. Select the **fr.json** node and add the following property:

Name	Type	Value
jcr:language	String	fr

15. Click **Save All**.

16. Navigate to the exercise folder provided to you, copy the URL from the **AEM Translator Tool URL.txt** file, and paste it in a browser. The **Strings & Translations** page opens.

17. 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.



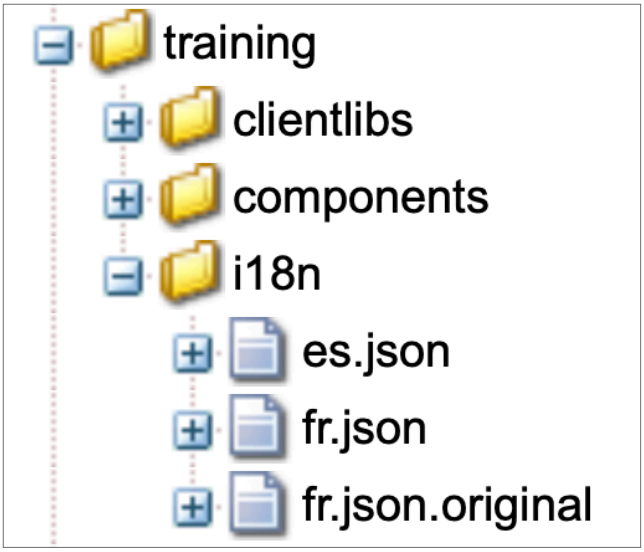
18. Enter the following information in the appropriate fields:

Field Name	Value
String	© {0} All rights reserved.
ES	© {0} Todos los derechos reservados.
FR	© {0} Tous droits réservés.

19. Click **OK**.

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

21. 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 was renamed and retained. It can be deleted.

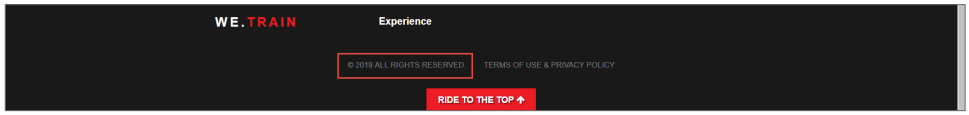


22. Examine the contents and properties of the es.json and fr.json files. Both are nodes of type nt:file and carry the mixin type of mix:language. Each file also has a jcr:language property. The files contain the localization string data.
You have now added two elements to your component to localize it. First, you added an HTL expression with the localization option to your component script. Second, you created an i18

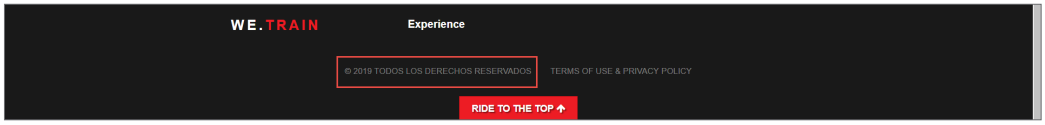
node structure in the component, which provides Sling and HTL, the necessary data to localize strings used in your component.

To test the localized content:

23. Navigate to the tab where the **English** page is open, or click http://localhost:4502/editor.html/content/we-train/en.html. The **English** page of the We.Train site opens. Notice, the footer of the page, as shown:



24. Navigate to the **Sites** tab, or click http://localhost:4502/sites.html/content. The **Sites** console opens.
25. Navigate to the **We.Train** site, and open the **French** page by selecting the thumbnail and clicking **Edit *(e)***. The page opens in a new tab.
26. Scroll down and notice the footer. The copyright message is translated to French, as shown:



Optional exercise: Add complete translation support for We.Train French, Spanish, and Italian

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. The Exercise files contain complete translations for the supported languages in We.Train. Update es.json, fr.json, and it.json files to accommodate all i18n keys in the components.
3. Create an Italian (it) language branch and a Spanish (es) language branch and see if the content is translated. Check the French page as well.

Optional exercise: Manage supported languages

AEM supports nine languages in the translator tool. You can customize the translator tool to show more/fewer languages depending on the requirements. To customize this console:

1. Create a new /etc/languages node
2. Add a multi-value property called **languages**. You can use this to set the languages supported in the translator tool.
3. Customize the translator tool to only support Spanish (es) and French (fr).
   If you need help with the customization, refer AEM: Using Translator to Manage Dictionaries.

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.

The two methods used for customizations are:

- Overlays
- Sling Resource Merger

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 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 looks at /libs.

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

Sling Resource Merger

The Sling Resource Merger is applicable only to the touch UI and enables you to customize consoles and page authoring. With Sling Resource Merger, you can overlay nodes without replicating all of their parent nodes.

The Sling Resource Merger provides services to access and merge resources. It merges the overlays of resources by using resource resolver search paths and difference mechanisms. A customized Sling vocabulary is used to manage the overrides of nodes and their properties. With this, the overlay resources/properties (defined in /apps) are merged with the original resources/properties (from /libs).

The Sling Resource Merger is used to ensure that all changes are made in /apps, and avoids the need to make any changes in /libs. After the structure is created, you can add your own properties to the nodes under /apps. The content of /apps has a higher priority than that of /libs. The properties defined in /apps indicate how the content merged from /libs are to be used.

The following table describes the differences between Overlays and Sling Resource Merger:

Overlays	Sling Resource Merger
Works based on search paths (/libs + /apps)	Works based on Resource Type Hierarchy
Must copy the whole subtree	Extends within an almost empty subtree
All properties are duplicated	Only required properties are overlaid
When upgrades are done to the /libs folder, changes need to be manually recreated under /apps.	As properties are not copied, only the structure, upgrades are automatically reflected in /apps

After you copy a system file from /libs to /apps to overlay it with custom code, your custom code will not pick up any modifications to the system component/file/script/dialog box, which results from the application of a hotfix, feature pack, or upgrade. A careful examination of the release notes of any upgrade, feature pack, or hotfix should be a step in your upgrade plan. This way, you will be able to evaluate changes and incorporate them into your application.

Exercise 9: Extend a core component

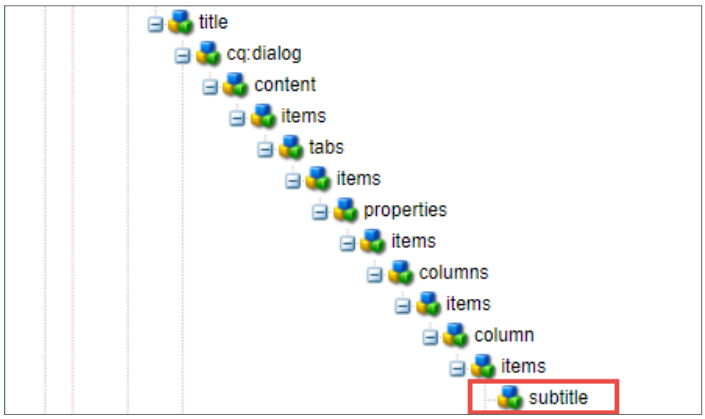
In this exercise, you will extend a core component without losing the functionality that it provides and ensuring it works for future core component upgrades.

To extend the Title component with the Sling Resource Merger:

1. In **CRXDE Lite**, navigate to the **/apps/training/components/content** folder. Notice that a **title** proxy component is available.
2. Navigate to the **/apps/core/wcm/components/title/v2/title** node, select the **cq:dialog** node, and click **Copy** from the actions bar.
3. Navigate to the **/apps/training/components/content/title** node, and click **Paste**. The **cq:dialog** node is added to the title proxy component.
4. Click **Save All** from the actions bar.
5. Navigate to the **/apps/training/components/content/title/cq:dialog/content/items/tabs/items/properties/items/columns/items/column/items** node, select the **types** node, and click **Delete** from the actions bar.
6. Click **Save All** from the actions bar.
7. Similarly, select the **defaulttypes** node that is below the **items** node, and click **Delete** from the actions bar.
8. Click **Save All** from the actions bar.

Note: You must delete the form fields that you want to inherit from the v2 title component. As a result, the custom formfields you create next will be merged with the v2 title dialog.

9. Select the **title** node, click **Rename** from the actions bar, and rename it to **subtitle**, as shown:



10. Update the following properties of the **subtitle** node:

Name	Type	Value
fieldLabel	String	Subtitle
fieldDescription	String	Enter your custom subtitle
name	String	./subtitle

11. Click **Save All**.
12. Open the **English** page of We.Train site and click the **Toggle Side Panel** icon from the page toolbar. The panel opens.
13. Click the **Components** icon. The **Component browser** opens.
14. Drag the **Title** component from the browser below the **Stockplex** component. The **Title** component is added to the page.
15. Select the **Title** component and click the **Configure** (wrench) icon from the component toolbar. The **Title** dialog box opens.
16. Notice that the **Subtitle** field is now added to the dialog box.
17. Update the following details:
    a. **Title**: **My new title**
    b. **Type / Size**: **H1**
    c. **Subtitle**: **My custom subtitle**
18. Click the **Done** (checkmark) icon The **English** page opens.

Notice how at this point, you do not see the subtitle on the page. You must create a local title.html script to reference the new subtitle property.

19. In **CRXDE Lite**, navigate to the **/apps/training/components/content** folder, select the **title** node, and click **Create** > **Create File**. The **Create File** dialog box opens.
20. Enter **title.html** in the **Name** field, and click **OK**. The file is created below the **title** node.
21. Click **Save All**.

22. Double-click the **title.html** field. The editor opens on the right.

23. Copy the code from **title.html** exercise file provided to you, and paste it in title.html.

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

25. Open the **title.html** to view the code that you added. Notice that the script is the exact same script as the core title component, except there is a new line of code that adds the subtitle to the component:

```
1.  <h1 data-sly-use.title="com.adobe.cq.wcm.core.components.models.Title"
2.      data-sly-use.template="core/wcm/components/commons/v1/templates.html"
3.      data-sly-element="${title.type}"
4.      data-sly-test.text="${title.text}">${title.text}</h1>
5.
6.      <!--/* Subtitle added to HTL */-->
7.      <h2 data-sly-test="${properties.subtitle}">${properties.subtitle}</h2>
8.
9.  <sly data-sly-call="${template.placeholder @ isEmpty=!text}"></sly>
```

26. Open the **English** page and observe that the new subtitle is now appearing. You have successfully extended the component. You added a new feature, a subtitle, to the core Title component by using the Sling resource merger to modify its dialog, and a minimal script override to modify its output.

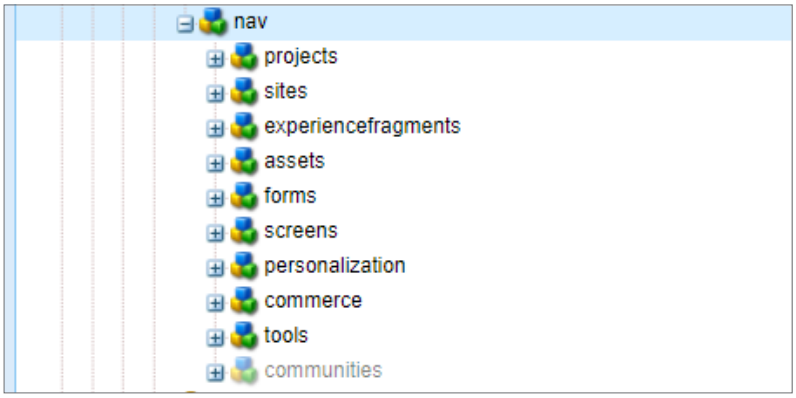
Exercise 10: Extend the navigation UI

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 UI** opens. Notice the Sites navigation button.

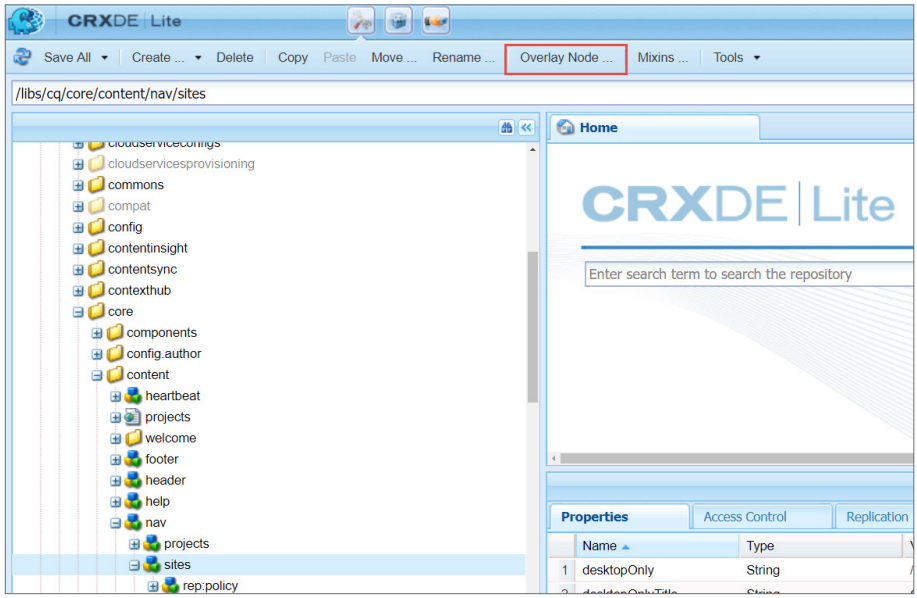
To modify the navigation button:

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.



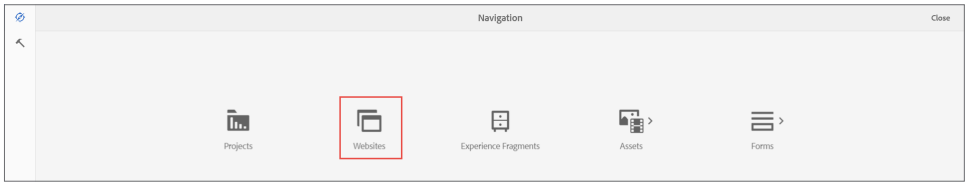
4. Ensure the **Overlay Location** field has **/apps/** as its value, select the **Match Node Types** checkbox, and click **OK**.
5. Click **Save All** from the actions bar.
6. Navigate to the **/apps** folder, and notice that the **/cq/core/content/nav/sites** structure is created. If you cannot view 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/** 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.



Optional Exercise: Hide a navigation button

To hide a navigation button:

1. In **CRXDE Lite**, navigate to the **/apps/cq/core/content/nav** node, click **Create** > **Create Node**. The **Create Node** dialog box opens.
2. Type **communities** in the **Name** field, and ensure the **Type** is **nt:unstructured**.
3. Click **OK**. The node is created.
4. Click **Save All** from the actions bar.
5. Add the following property to the **communities** node:

Name	Type	Value
sling:hideResource	Boolean	true

6. Click **Save All** from the actions bar.
7. Open the tab where the **Navigation** page is open. Refresh the page and notice that the **Communities** option is hidden from **Navigation**.

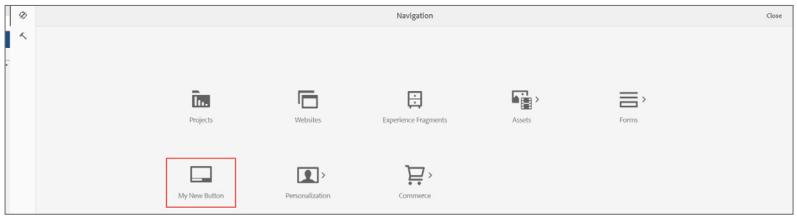
Optional Exercise: Add a new navigation button

To add a new navigation button:

1. In CRXDE Lite, navigate to the **/libs/cq/core/content/nav** node, select the **screens** node, and click **Copy** from the actions bar.
2. Navigate to the **/apps/cq/core/content/nav** node, and click **Paste** from the actions bar. The screens node is added.
3. Click **Save All** from the actions bar.
4. Select the **screens** node, and rename it as **mybutton**.
5. Click **Save All** from the actions bar.
6. Update the following properties of the **mybutton** node:

Name	Type	Value
jcr:title	String	My New Button
id	String	cq-mybutton
href	String	/sites.html

7. Click **Save All** from the actions bar.
8. Open the tab where the **Navigation** page is open. Refresh the page and notice that a new console, **My New Button**, is created, as shown:



9. Click **My New Button**. Notice that the **Sites** console opens.

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:

- Explain how to prepare AEM for production
- Explain the environment setup of AEM
- Complete the empty page template type
- Create production templates
- Create a code content package
- Explain how to debug errors by using the Developer mode
- Explain how to debug errors by using parameters

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, sync as 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.

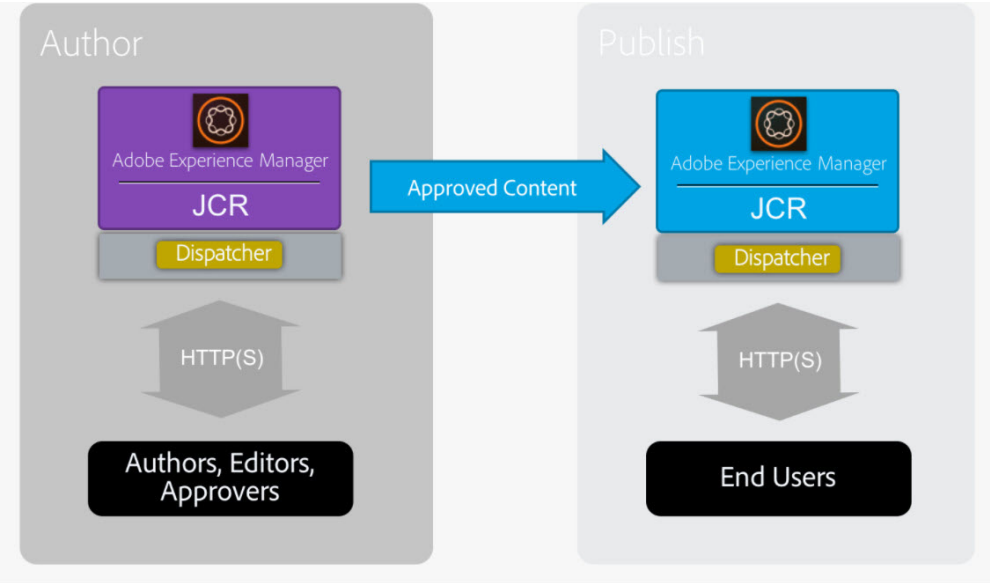
AEM Environment

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

Instances

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

The author instance is used by content authors, editors, and approvers to create and review content. When the content is approved, it is published to the publish instance from where it is accessed by end users.



Dispatcher

The Dispatcher is the caching and/or load balancing tool used by AEM. It handles caching and URL filtering and is installed on the web server. You can increase the security of your AEM instance by using the Dispatcher in conjunction with an enterprise-class web server.

There are two basic approaches to web publishing:

- Using static web servers, such as Apache or IIS. This approach is simple and fast.
- Using Content Management Servers, which provide dynamic, real-time, intelligent content but require much more computation time and other resources.

The Dispatcher creates an environment that is both fast and dynamic. It works as a part of a static HTML server, such as Apache and:

- Store as much of the site content as possible in the form of a static website
- Access the layout engine as little as possible

The static content is handled with the same speed and ease as on a static web server. In addition, you can use the administration and security tools available for your static web server(s).

The dynamic content is generated as needed without slowing down the system.

The Dispatcher contains the mechanisms to generate and update static HTML based on the content of the dynamic site. You can specify in detail which documents are stored as static files and which are always generated dynamically.

Replication Agents

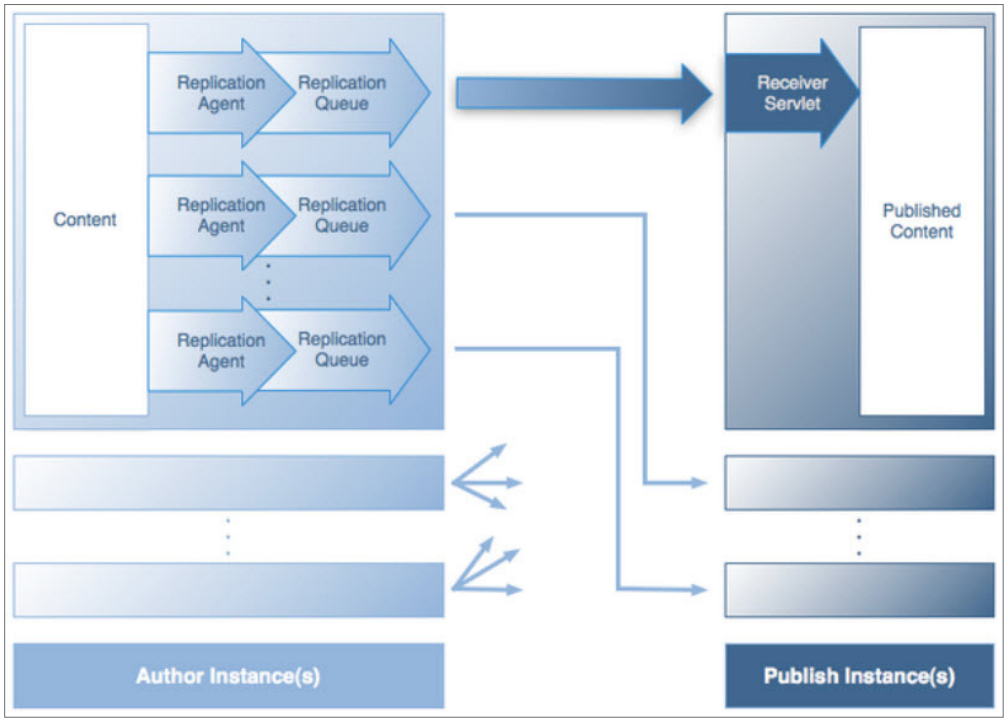
Replication agents help:

- Publish (activate) content from an author environment to a publish environment
- Remove content from the Dispatcher cache explicitly
- Return user input, such as form input, from the publish environment to the author environment

Replicating from the Author Instance to the Publish Instance

Replication to a publish instance or a Dispatcher involves:

1. The author publishes (activates) the content. This can be initiated by a manual request or by preconfigured triggers.
2. The request is passed to the appropriate default replication agent. An environment can have several default agents.
3. The replication agent packages the content and places it in the replication queue.
4. The content is transported from the queue to the publish environment by using the configured protocol—usually HTTP.
5. A servlet in the publish environment receives the request and publishes the received content. The default servlet is http://localhost:4503/bin/receive.



Performance Guidelines

When working with AEM:

- TarMK with File Datastore is the recommended architecture for most customers:
  - The minimum topology is one author instance, two publish instances, and two Dispatchers.
  - The binary-less replication is turned on if the File Datastore is shared.
- MongoMK with File Datastore is the recommended architecture for horizontal scalability of the author tier:
  - The minimum topology is three author instances, three MongoDB instances, two publish instances, and two Dispatchers.
  - The binary-less replication is turned on if the File Datastore is shared.
- Nodestore should be stored on the local disk and not on a Network Attached Storage (NAS).
- Amazon S3 is the recommended datastore for a total volume of assets above 5 TB:
  - The Amazon S3 datastore is shared between the author tier and the publish tier.
  - The binary-less replication must be turned on.
  - Datastore Garbage Collection requires a first run on all author nodes and publish nodes and then a second run on author nodes.
- The custom index should be created in addition to the out-of-the box index based on most common searches:
  - Lucene indexes should be used for custom indexes.
- Workflow customization can substantially improve the performance. For example, you can remove the video step in the Update Asset workflow and disable the listeners that are not used.

Security Checklist

You must follow this checklist when working with AEM security:

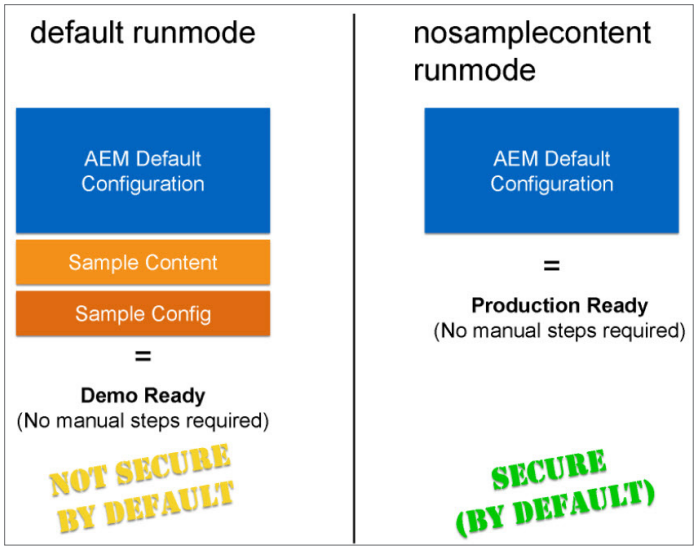
- Run AEM in the production ready mode.
- Enable HTTPS for the transport layer security.
- Install security hotfixes.
- Change the default passwords for the AEM and OSGi console admin accounts.
- Implement the custom error handler.
- Complete the Dispatcher security checklist.

AEM in Production Ready Mode

AEM includes a **nosamplecontent** runmode to automate the process to prepare an AEM instance for deployment in a production environment.

The new run mode will automatically configure the instance to adhere to the security best practices described in the security checklist and remove all sample We.Retail applications and configurations in the process.

The following screenshot depicts the AEM runmode:



To run AEM in production-ready mode, you must add **nosamplecontent** through the -r runmode switch to your existing startup arguments:

```
java -jar aem-quickstart.jar -r nosamplecontent
```

When running AEM in production-ready mode:

- The CRXDE Support bundle (com.adobe.granite.crxde-support) is disabled by default in production-ready mode. It can be installed any time from the Adobe public Maven repository.
- The Apache Sling Simple WebDAV Access to repositories (org.apache.sling.jcr.webdav) bundle will only be available on author instances.
- Newly created users need to change their password on the first login. This does not apply to the admin user.
- Generate debug info is disabled for the Apache Java Script Handler.
- Mapped content and Generate debug info are disabled for the Apache Sling JSP Script Handler.
- The Day CQ WCM Filter is set to edit on author and disabled on publish instances.
- The Adobe Granite HTML Library Manager is configured with the following settings:
  a. Minify: enabled
  b. Debug: disabled

c. Gzip: enabled
d. Timing: disabled

- The Apache Sling GET Servlet is set to support secure configurations by default. The following table lists the configurations:

Configuration	Author	Publish
TXT rendition	disabled	disabled
HTML rendition	disabled	disabled
JSON rendition	enabled	enabled
XML rendition	disabled	disabled
json.maximumresults	1000	100
Auto Index	disabled	disabled

Exercise 1: 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. Click **Adobe Experience Manager** from the header bar, and click the **Tools** icon.
2. Click **General** > **Templates**. The **Templates** console opens.
3. 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.
4. Select the **Layout Container [Root]** and click the **Policy** icon from the component toolbar. The **Layout Container** wizard opens.
5. In the **Policy** section, notice the structure components you added previously to the policy.
6. In the **Properties** section, under **Allowed Components**, 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.

7. If not already open, open CRXDE Lite (http://localhost:4502/crx/de).
8. Navigate to the **/conf/we-train/settings/wcm/templates/content-page/policies/jcr:content** node.
9. Select the **root** node, and click **Copy** from the actions bar. The root node you are copying has the policy for the root layout container. This ensures the approved structure components are available for a template author.
10. Navigate to the **/conf/we-train/settings/wcm/template-types/empty-page/policies/jcr:content** node, and click **Paste** from the actions bar.
11. Click **Save All** from the actions bar.

12. Navigate to the **/conf/we-train/settings/wcm/template-types/empty-page/policies/jcr:content/root** node, select the **responsivegrid** node, and click **Delete** from the actions bar. This will help template authors determine the components to be used on a page. If you need to auto-add any other configurations or components to a new template, you can copy those nodes/properties to the template-type.

Exercise 2: 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.

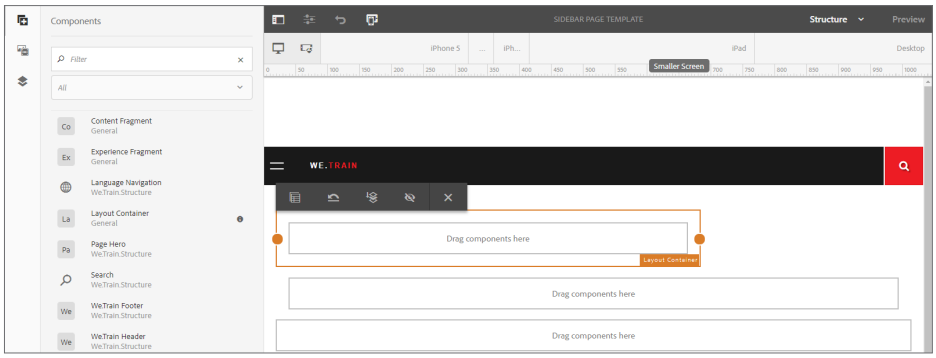
You will create the following two new templates:

- Sidebar Page template
- Stockplex template

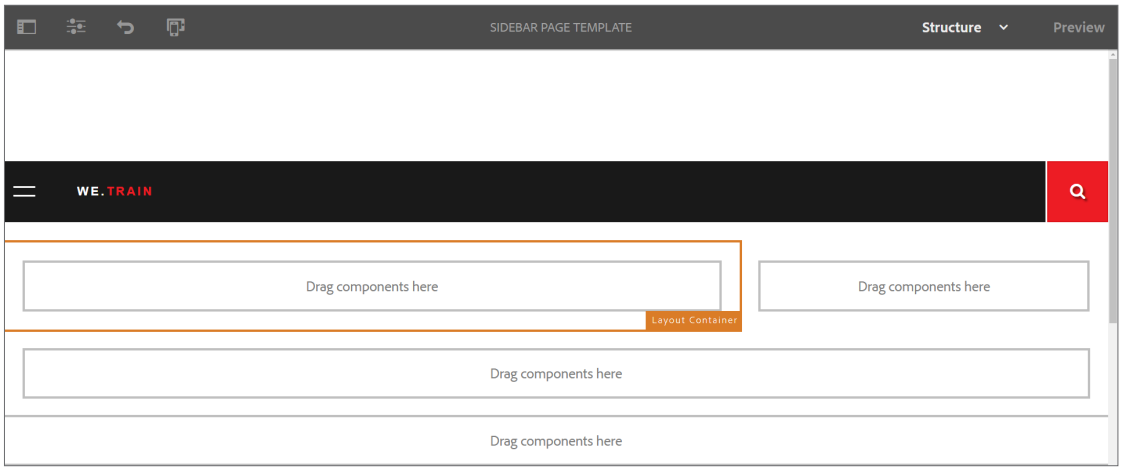
To create a Sidebar Page template:

1. Click **Adobe Experience Manager** from the header bar, and click the **Tools** icon.
2. Ensure you are in the **General** section, and click **Templates**. The **Templates** console opens.
3. Click the **We.Train** folder to open it, and 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(2/2)** wizard opens.
5. Enter **Sidebar Page Template** in the **Template Title** field, 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 browser** opens. Ensure the **All** option is selected on the drop-down menu in the Components browser.
10. Drag the **We.Train Header** component from the components browser to the **Drag components here** area placeholder. The component is added.
11. Drag the **Layout Container** component from the components browser 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 browser to the **Drag components here** area placeholder The component is added.
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.

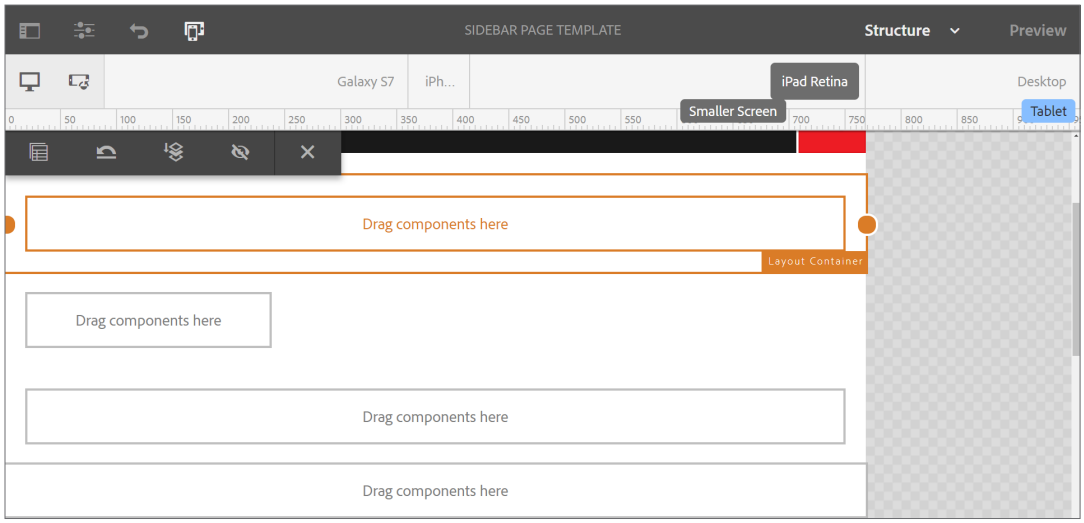


15. Select the middle **Layout Container**, click the **Layout** icon, and drag the right orange handler up to 1/3rd of the grid system. The **Layout Container** resizes accordingly.
16. Verify the last layout container will remain as it is (12/12). You will have a 2/3, 1/3, and 12/12 layout, as shown:



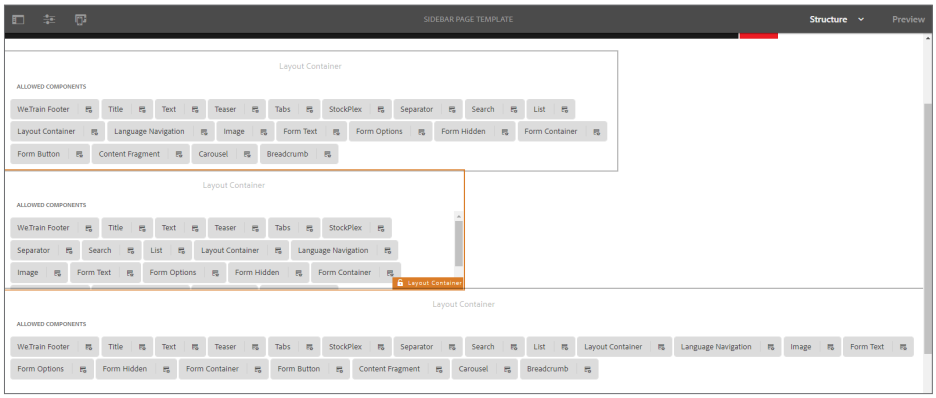
17. Click the **Emulator** icon from the toolbar, and click the **iPad Retina** device from the group. The template rearranges itself to the device size.

18. Select the first **Layout Container**, click the **Layout** icon, and adjust 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. Select the middle **Layout Container**, click the **Layout** icon, and click the **Hide component** (eye) icon. The **Layout Container** is hidden from the grid.
23. Click the **Emulator** icon from the toolbar, and click the **Desktop** device from the group. The template rearranges itself to the device size.
24. 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.
    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, select the **We.Train Content Policy** from the **Select policy** drop-down menu.
        ii. Click the **Done** (checkmark) icon. You will be taken back to the template.

Your template should look similar to the one given in the below screenshot:

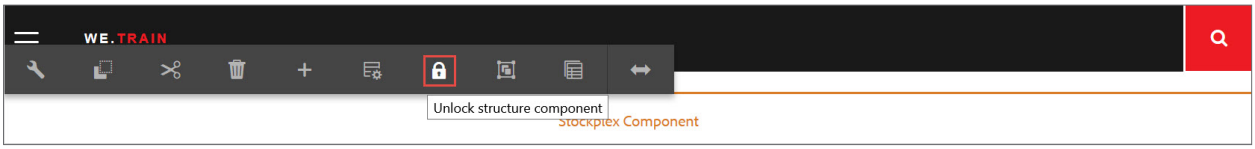


25. Navigate to the tab where the **Templates** console is open.
26. Select the **Sidebar Page Template**, and click **Enable** from the actions bar. The **Enable** dialog box opens.
27. Click **Enable**. The template is enabled.
28. With the **Sidebar Page Template** already selected, click **Publish**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

To create a Stockplex template:

30. Navigate to **Templates** > **We.Train** folder, and 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(2/2)** wizard opens.
32. Enter **Stockplex Template** in the **Template Title** field, 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 browser** opens.
37. Ensure the **All** option is selected on the drop-down menu in the Components browser.
38. Drag the **We.Train Header** component from the Components browser to the **Drag components here** area placeholder. The **Header** component is added.

39. Drag the **Stockplex** component from the Components browser and drop it below the **Header** component. The **Stockplex** component is added.
40. Drag the **Layout Container** component from the Components browser and drop it below the **Stockplex** component. The **Layout Container** component is added.
41. Drag the **We.Train Footer** component from the Components browser and drop it below the **Layout Container** component. The footer component is added. Your page should look similar to the one given in the below screenshot:
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.

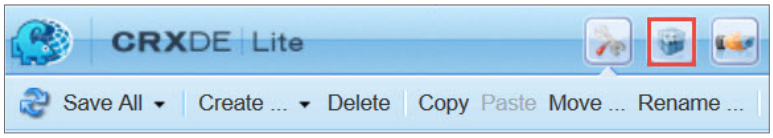


44. Select the **Layout Container** component and click the **Policy** icon from the component toolbar. The **Layout Container** wizard opens.
45. In the **Policy** section, select the **We.Train Content Policy** from the **Select policy** drop-down menu.
46. Click the **Done** (checkmark) icon. The 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 checkboxes are selected, and click **Publish**. The **INFO The page has been published** success message appears.

Exercise 3: Create a code content package

In this exercise, you will include all your work and configurations in your environment into an exportable package.

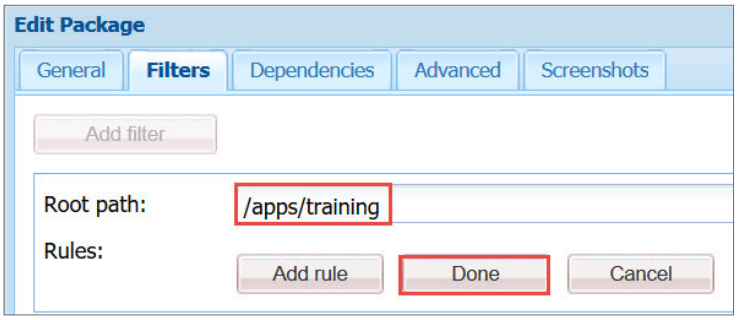
1. Ensure you are in CRXDE Lite or click http://localhost:4502/crx/de. The **CRXDE Lite** page opens.
2. Click the **Package** icon from the header bar, as shown. The **CRX Package Manager** page opens.



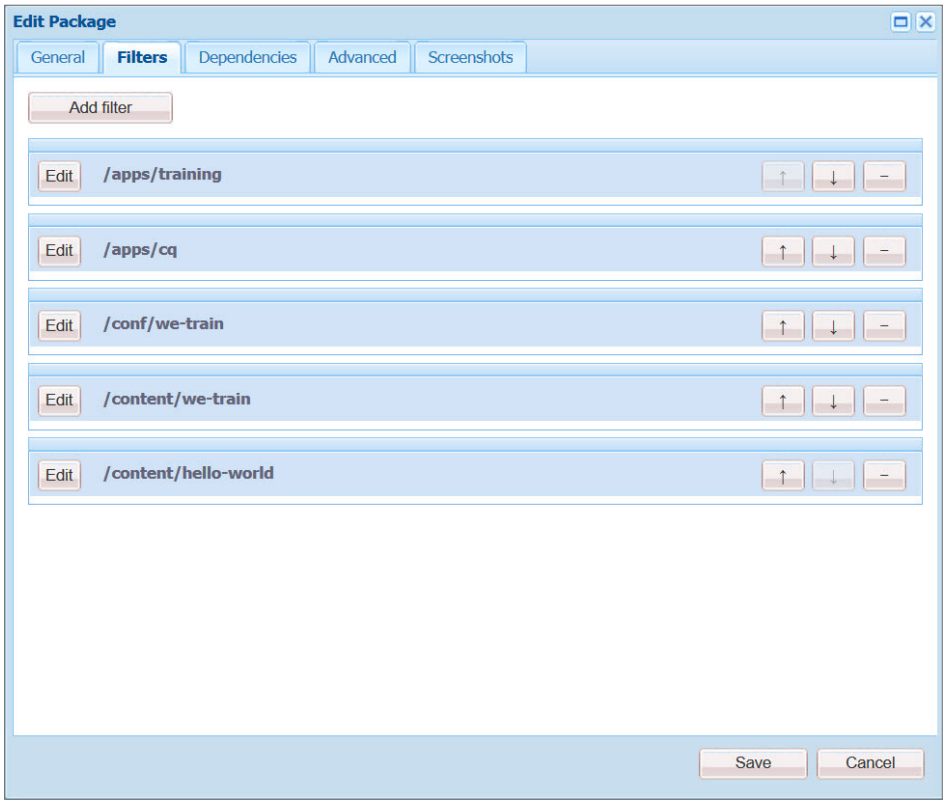
3. Click **Create Package** from the actions bar. The **New Package** dialog box opens.
4. Enter the following details:

Field	Value
Package Name	<YourName>DevWebsites
Version	v1.0
Group	training

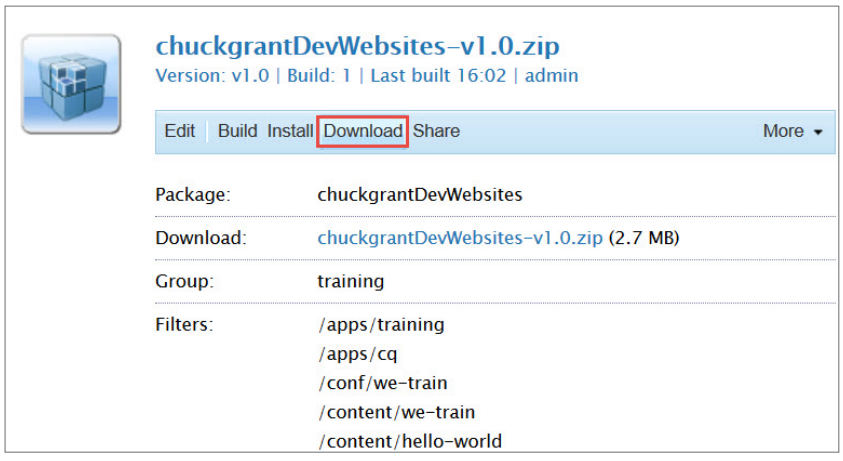
5. Click **OK**. The **CRX Package Manager** page opens.
6. Click **Edit** in the newly created package. The **Edit Package** dialog box opens.
7. To add filters to the package, click the **Filters** tab and click **Add filter**.
8. In the **Root path** field, browse and select the **/apps/training** folder, click **OK**, and click **Done**, as shown:



9. Click **Add filter** to add the following additional filters. Click **Done** after adding each new filter.
   a. /apps/cq
   b. /conf/we-train
   c. /content/we-train
   d. /content/hello-world



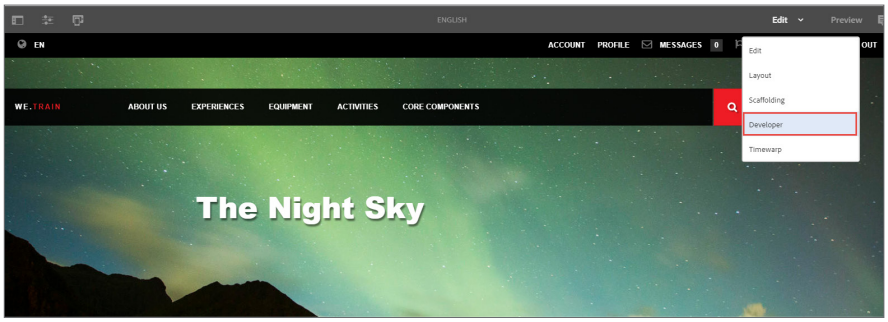
10. Click **Save**.
11. Click **Build** to build the package. The confirmation dialog box opens.
12. Click **Build**. The package is now ready for download.
13. Click **Download**, as shown, to download a copy of the package to your computer.



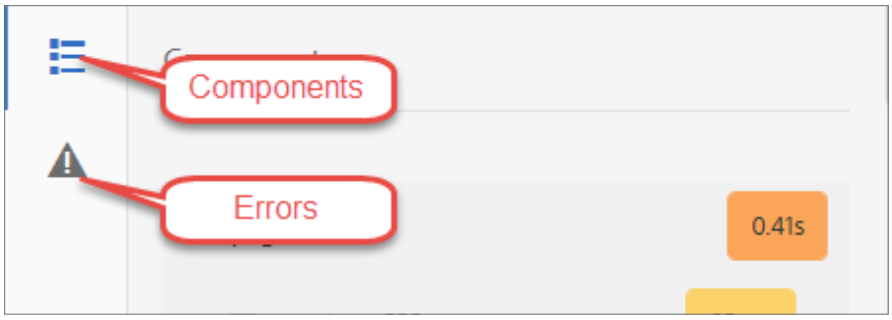
Note: If you are using a ReadyTech environment, you can upload your package to a file-sharing tool, such as the Adobe Document Cloud (https://cloud.acrobat.com), so that you can use the package after the class. You must use your Adobe ID to access the Document Cloud. If you do not have an Adobe ID, you may register for one for free. Alternatively, you can place the package file in C:\ReadyTech\Outbox, and your instructor can retrieve and email the package to you at the end of class.

Debug Errors in Developer Mode

The Developer mode option is available on the side panel of the page editor. To open this mode, select Developer from the mode selector in the toolbar of the page editor, as shown:



There are two tabs in the Developer mode, Components and Errors, as shown:



Components

The Components tab displays a component tree that:

- Outlines the chain of components and templates rendered on the page (SLY and JSP). You can expand the tree to view context within the hierarchy.
- Shows the server-side computational time needed to render the component.
- Enables you to expand the tree and select specific components within the tree. When you select a component, details such as the repository path and the links to scripts (accessed in CRXDE Lite) are displayed.
- Highlights the selected components (in the content flow, indicated by a blue border).

The Components tab provides associated script for each component. You can click the Edit icon to navigate directly to the script in CRXDE Lite.

Errors

The Errors tab displays a warning if:

- A component writes an entry to the error log with error details and direct links to the appropriate code within CRXDE Lite
- A component opens an admin session

Parameter Debugging

You can debug errors on a page by appending the following parameters in the URL:

- ?debug=layout
- ?debugClientLibs=true

?debug=layout

This parameter lists the information about renderers, selectors, and resources.

?debugClientLibs=true

This parameter lists the client libraries loaded onto the page in the page source HTML.

References

Use the following links for more information on:

- Resources for AEM developers:
    - WKND Tutorial:

https://helpx.adobe.com/experience-manager/kt/sites/using/getting-started-wknd-tutorial-develop.html

    - ACS Commons:

https://adobe-consulting-services.github.io/acs-aem-commons/

    - AEM Developer Chrome Extension:

https://chrome.google.com/webstore/detail/aem-developer/hgjhcngmldfpgpakbnffnbnmcmohfmfc?hl=en-US

    - AEM GEMs sessions:

https://docs.adobe.com/content/ddc/en/gems.html

    - AEM Community:

https://forums.adobe.com/community/experience-cloud/marketing-cloud/experience-manager

    - AEM Developer Docs:

https://helpx.adobe.com/experience-manager/6-4/sites/developing/user-guide.html

    - AEM Featured Videos:

https://helpx.adobe.com/experience-manager/kt/index/aem-6-4-videos.html

    - AEM Tutorials:

https://helpx.adobe.com/experience-manager/tutorials.html

- Dispatcher:

https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher.html

- Replication Agents:

https://helpx.adobe.com/experience-manager/6-4/sites/deploying/using/replication.html

- Performance Guidelines:

https://helpx.adobe.com/experience-manager/6-4/sites/deploying/using/performance-guidelines.html

- Security Checklist:

https://helpx.adobe.com/experience-manager/6-4/sites/administering/using/security-checklist.html

- Production Ready Mode:

https://helpx.adobe.com/experience-manager/6-4/sites/administering/using/production-ready.html

Appendix

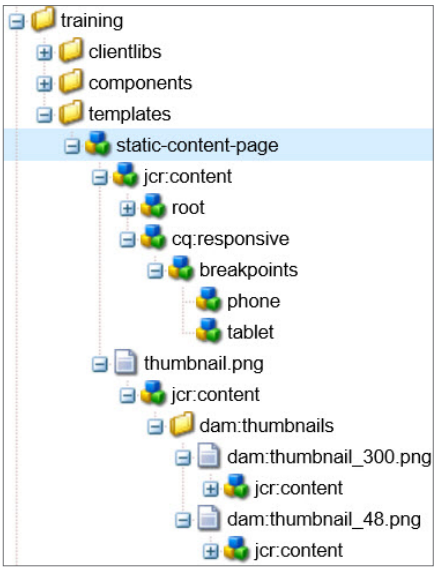
In this section, you will learn about static templates, the dialog coversion tool, and AEM and General Data Protection Regulation (GDPR) Compliance.

Static Templates

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

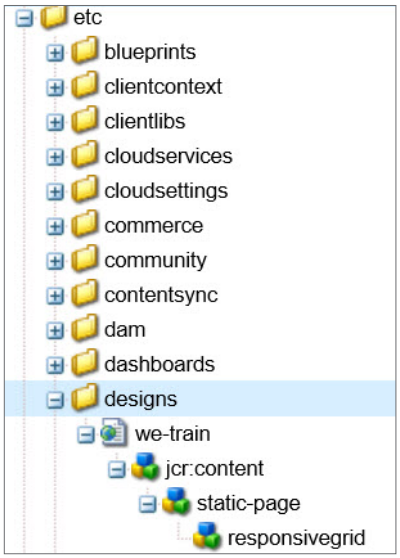
You can create webpages, based on your requirements, using static templates which will achieve the same output as editable templates you created in a previous course, "Creating Editable Templates and Pages in AEM".

To understand how the static templates work, you need to install **we-train-static-template.zip** on your AEM instance. The package contains a static template for the We.Train site that was created for you.
The static templates typically exist under **/apps/<my-app>/templates** folder, where as the editable templates exist under **/conf/<tenant>**. A page component for a static template must include all the structure components that are hard coded in the HTL as shown in the screenshot below. If you have installed the package, open CRXDE Lite, and navigate to the **/apps/training/templates/static-content-page** node and observe the changes.



The page component for an editable template typically has a layout container that allows template authors to drag and drop structure components onto the template.

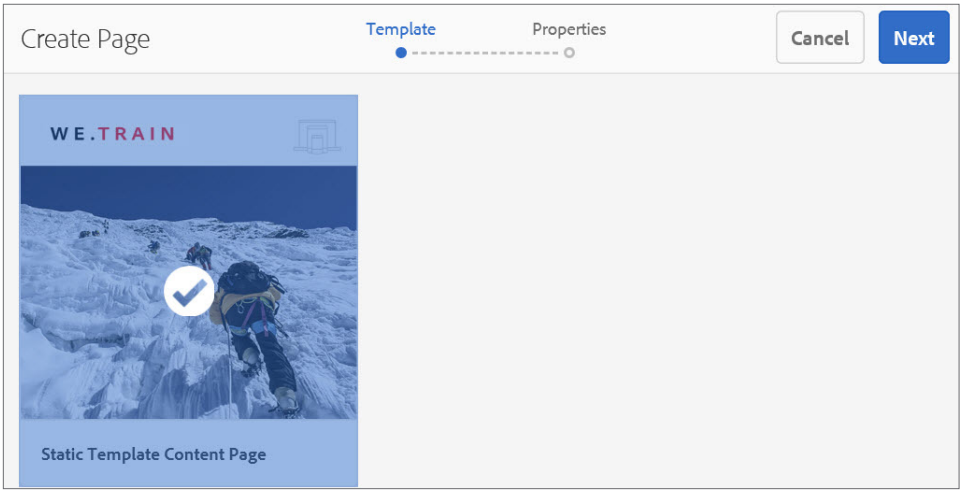
Any component configurations (Design Mode) for static templates exist under **/etc/designs/<my-design>** as shown in the screenshot below**.** Any component configurations (Content Policies) for editable templates exist under **/conf/<tenent>/settings/wcm/policies**.



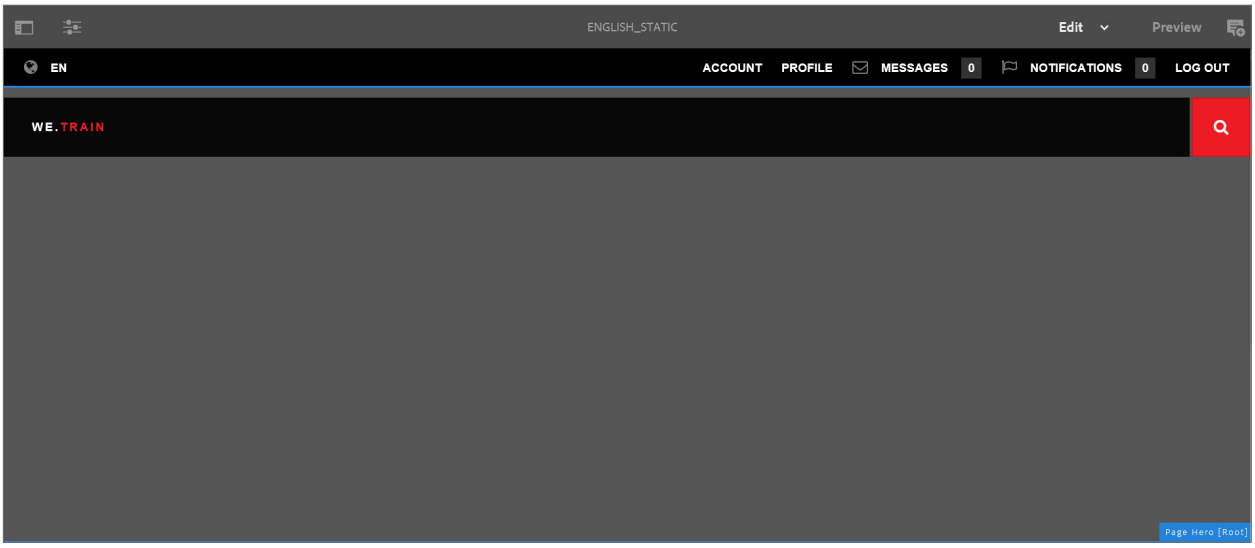
To use the static template for creating pages, you need to add the following properties to **We.Train** from the Sites console.

- Design: /etc/designs/we-train
- Allowed Templates: /apps/training/templates/.*

After, defining the above properties, when creating a new page in **We.Train** site, the **Static Template Content Page** appears in the **Create Page** dialog box as shown:



The page created from Static Template Content Page has the same structure as the page created from the editable template, as shown:



Dialog Conversion Tool

The dialog conversion tool is provided to convert Classic UI dialogs or dialogs based on Coral 2 to Coral 3. The tool uses the original dialog to create a duplicate dialog designed for the Touch UI, based on Granite UI and Coral 3.

Although the Dialog Conversion tool will create a new dialog, it will skip what it cannot convert. Therefore, the resulting dialog might contain nodes from the original dialog copied as-is, if no rule matched that specific component. In addition, a converted component might have some unconverted properties, because there was no appropriate rule to convert them.

The tool cannot cover every scenario, as its conversion rules are non-exhaustive and operate on a best-effort basis. It converts the most frequently used elements and properties, but the conversion will be incomplete when dealing with customizations or highly-specialized dialogs. Converted dialogs may require additional adjustments and all conversions must be reviewed.

You can use the following link for more information on:

- Dialog Conversion Tool:

https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/dialog-conversion.html

AEM and GDPR Compliance

The 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 will apply to any company doing business with individuals in the EU. Adobe recognizes that this presents a new opportunity for companies to strengthen their brand loyalty by focusing on consumer privacy while delivering amazing experiences.
AEM instances and the custom applications that might process Personally Identifiable Information (PII) data are owned and operated by AEM customers. This means that the Data Processor and Data Controller as defined in GDPR are both owned and managed by the AEM customer, so AEM 6.4 does not include any out-of-the-box service to handle GDPR requests. You can use the following links for more information on:

- GDPR Compliance
- AEM Foundation GDPR support