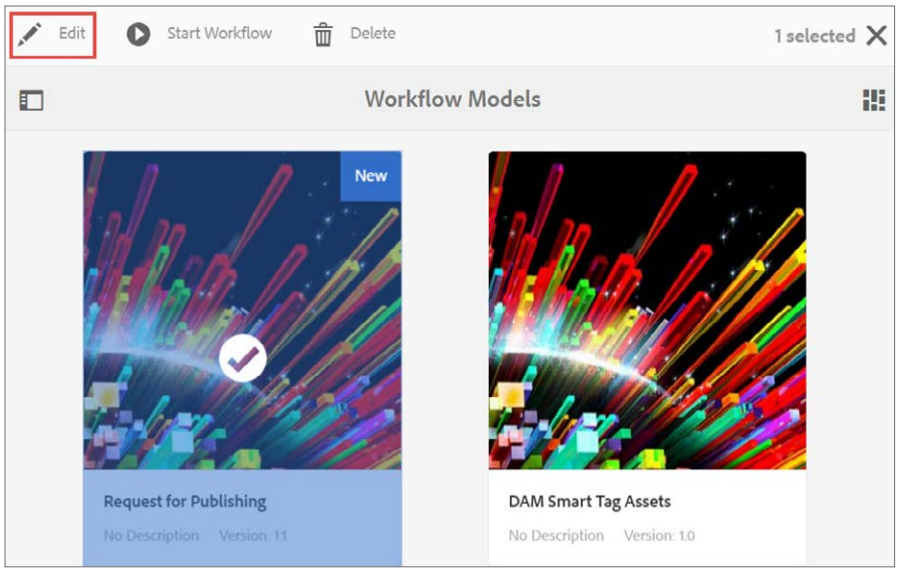
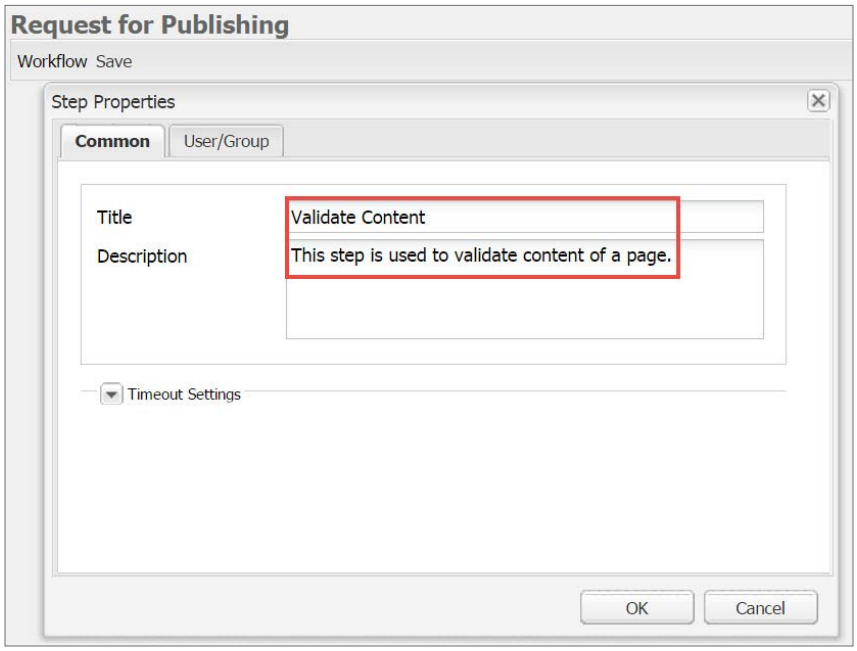
To edit the workflow model:

1. Select the **Request for Publishing_<username>** model that you created earlier.
2. Click **Edit** from the toolbar. A new window (named after the workflow) opens for editing and configuring the workflow.



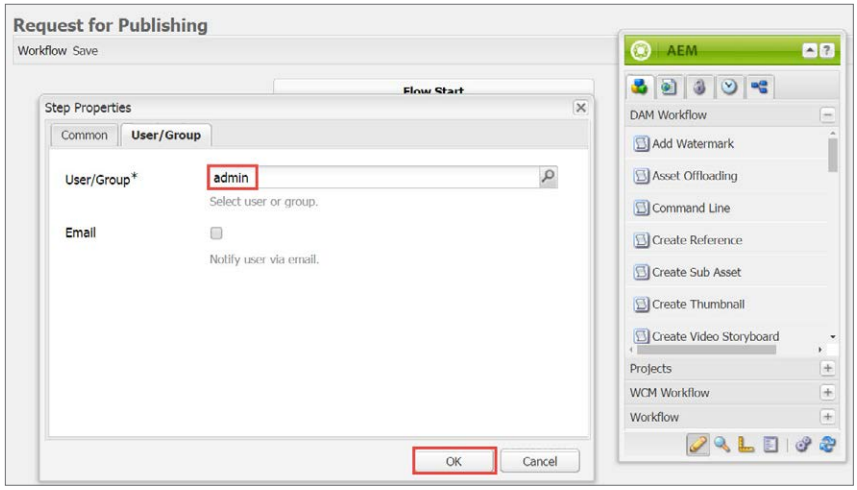
3. Three steps are created:
   a. Flow Start
   b. Step1
   c. Flow End
4. Double-click **Step 1**. The **Step Properties** dialog opens.

5. On the **Common** tab, add the following values:
    a. **Title**: Validate Content
    b. **Description**: This step is used to validate content of a page.

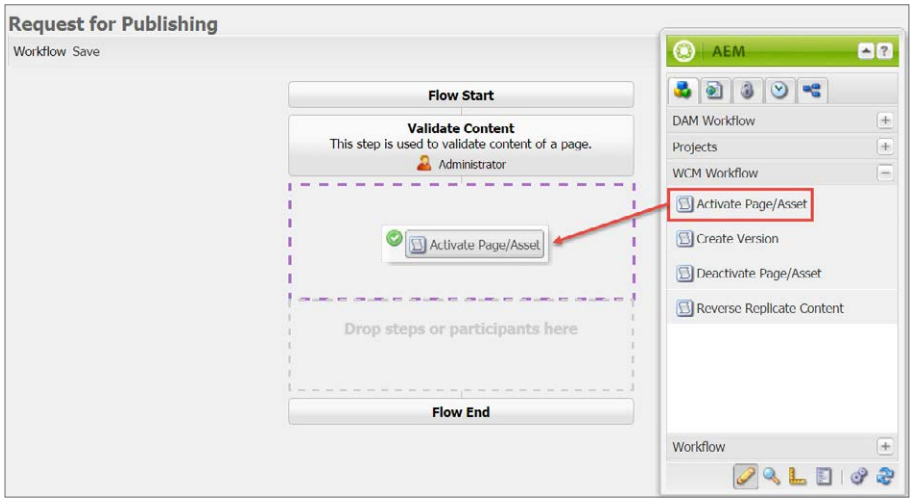


6. In **User/Group** tab, enter the following values:
    a. **User/Group**: Choose the user you want to assign this step/task to.
        i. Select any user from the drop-down to assign the task to that user, or assign the task by entering the user name in the field. For this task, assign it to yourself.

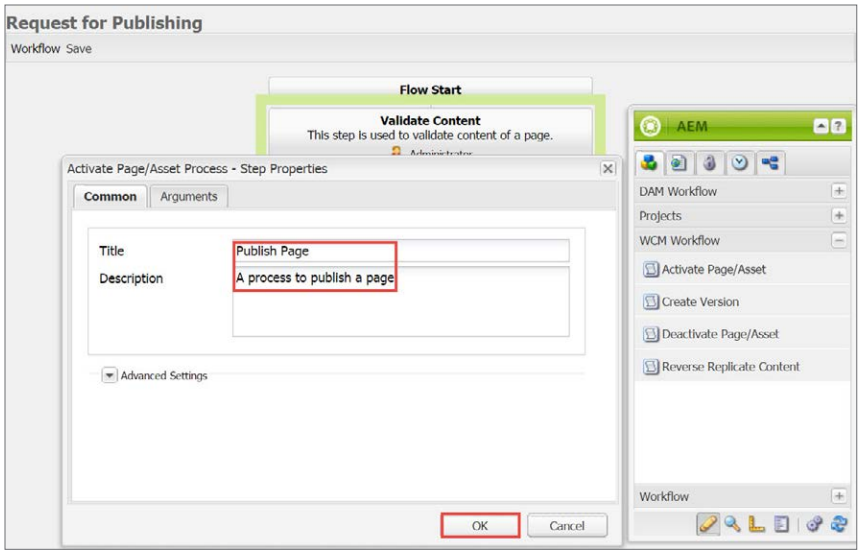
7. Click **OK** to save and close the dialog box.



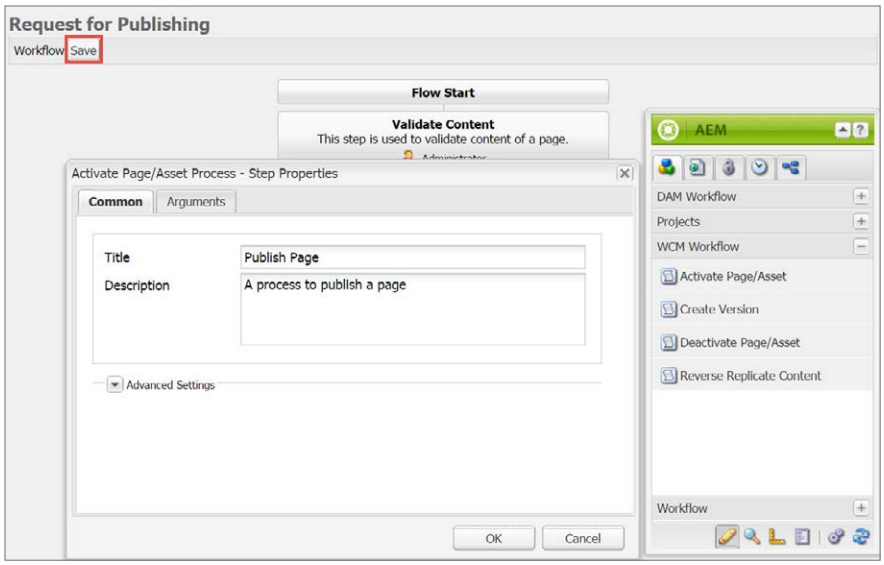
8. Drag and drop the **Activate Page/Asset** process step from the **WCM Workflow** onto the workflow.



9. Double-click the **Activate Page/Asset** process step. The **Step Properties** dialog opens.
10. On the **Common** tab, enter the following values:
    a. Title: **Publish Page**
    b. Description: **A process to publish a page**
11. Do not make any changes in the **Arguments** tab.
12. Click **OK** to save and close the dialog box.



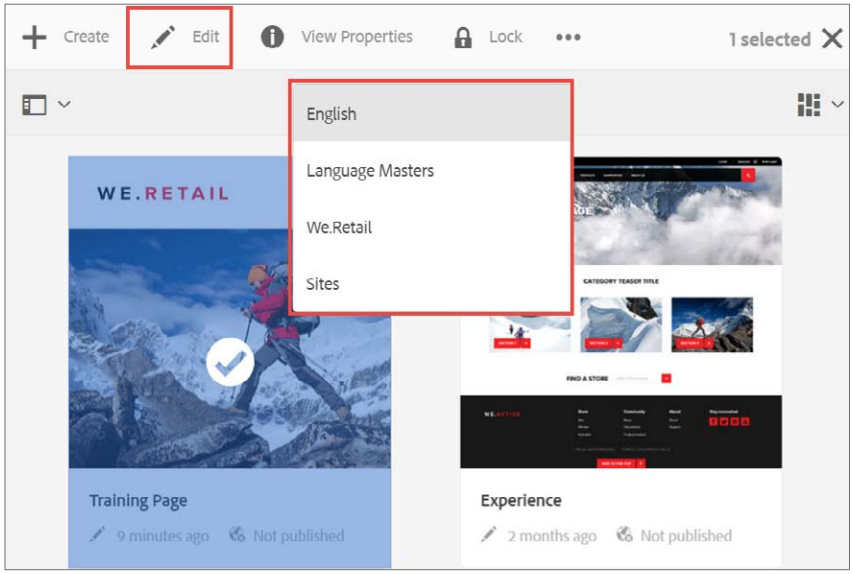
13. Click **Save** to save the changes made to the workflow.



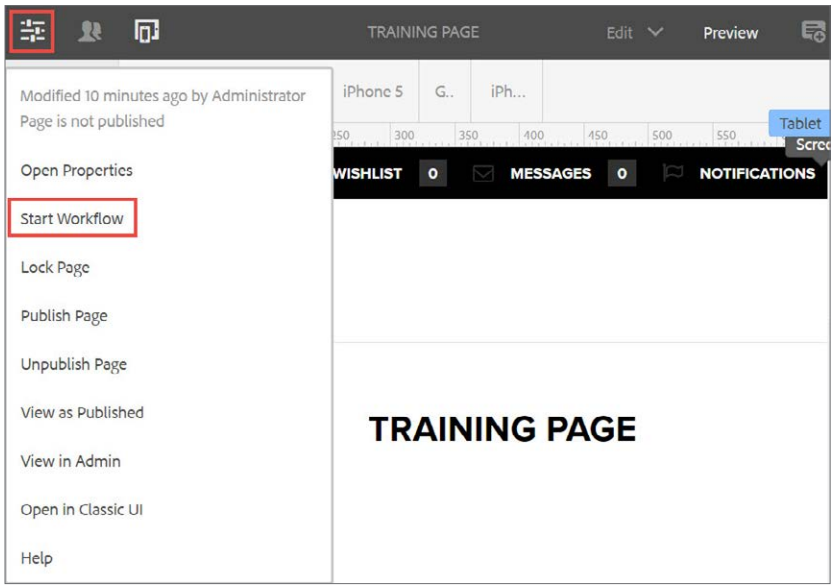
You have a workflow model with steps; let's start the workflow from a page that needs to be published.

To start the workflow from a page:

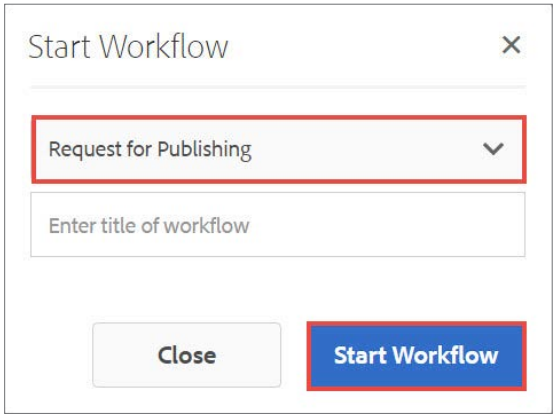
14. Open the **Training Page_<username>** in **Edit** mode.



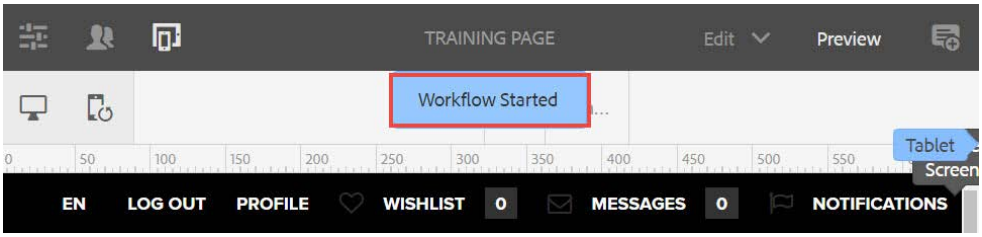
15. Select **Start Workflow** using the **Page Information** icon.



16. Select the **Request for Publishing_<username>** model from the drop-down, and then click **Start Workflow**.

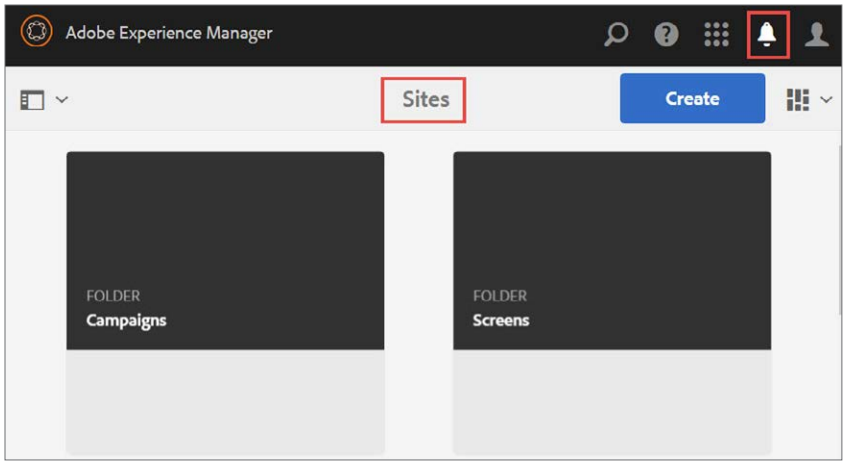


Workflow Started message appears on the page.

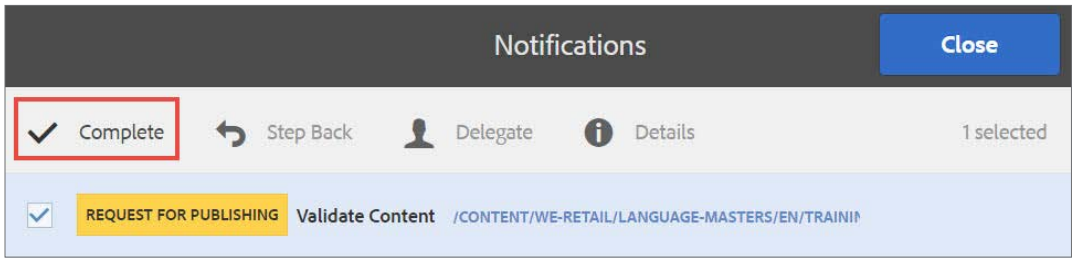


To complete the workflow:

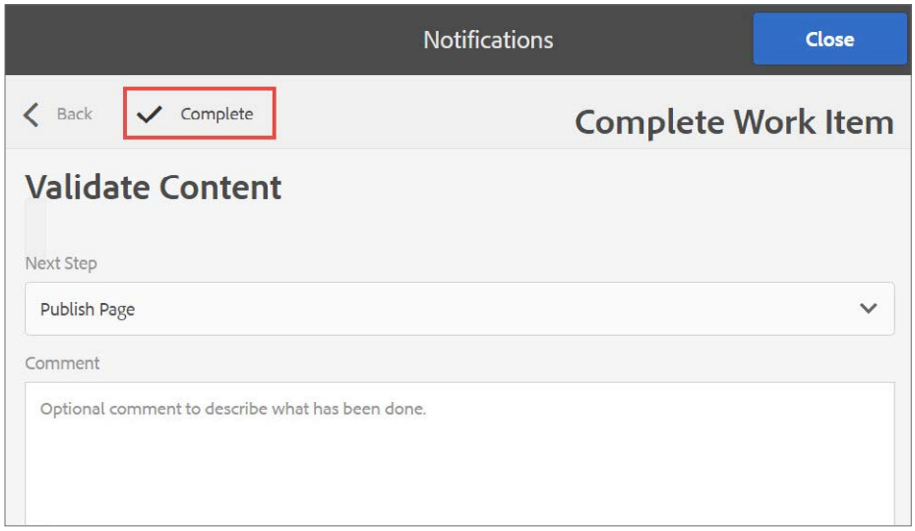
17. Navigate to the **Sites** console and click **Notifications** to see the actions assigned to you.



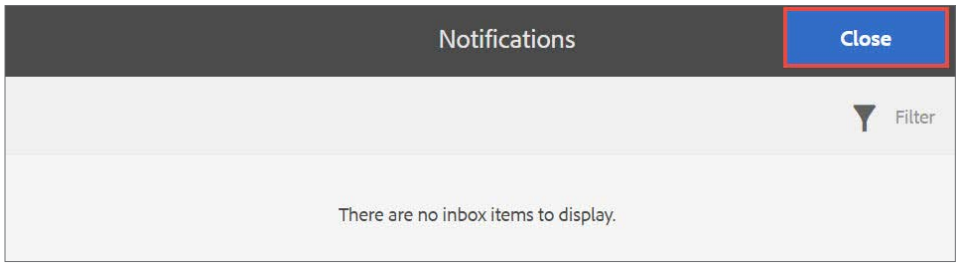
18. Select **VALIDATE CONTENT** and click **Complete**. The **Complete Work Item** screen opens.



19. Click **Complete** to complete **VALIDATE CONTENT** step of the workflow.
20. Optionally, you can add comments to describe the action taken to complete this step.

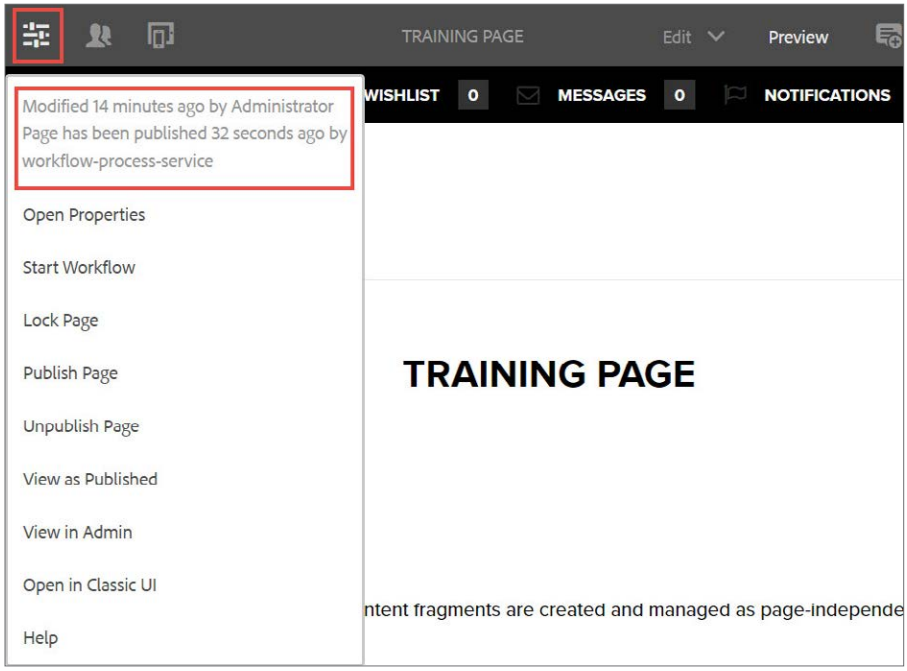


21. Click **Close** as there are no workitems in the inbox. You will be taken to the **Sites** console.



22. Open **Training Page_<username>**.

23. Click **Page Information** icon from the page toolbar. You will see that the **Publish Page** process step of workflow has published the page automatically.

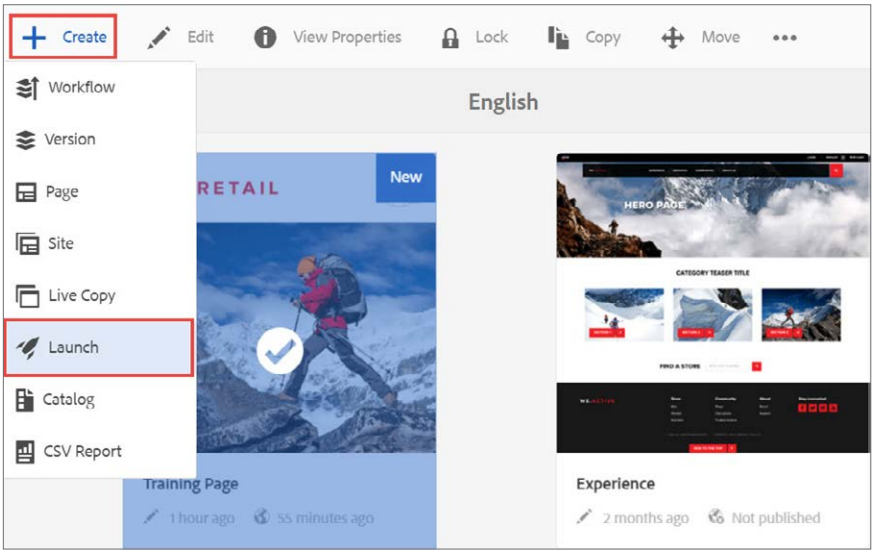


Try out yourself: Access Training Page_<username> in your Publish instance.

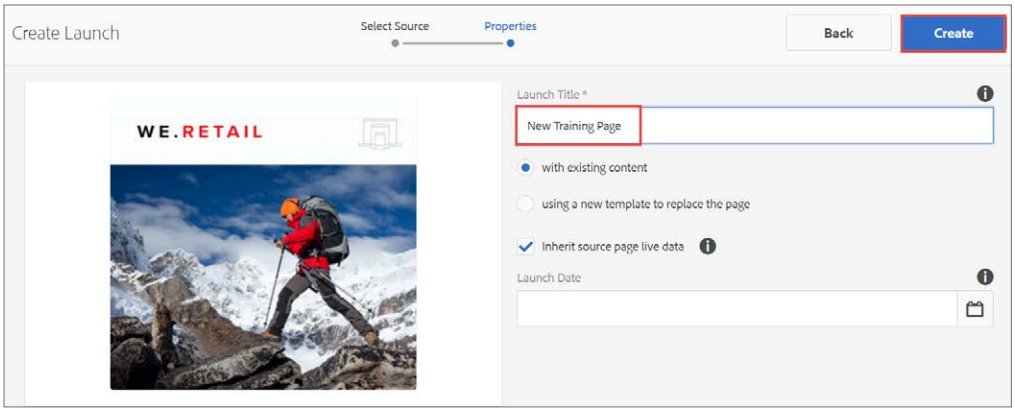
Task 4: Create, edit, and promote a launch

To create a launch:

1. Navigate to **Training Page_<username>** from **Sites** > **We.Retail** > **Language Masters** > **English**, and click **Create** > **Launch**.

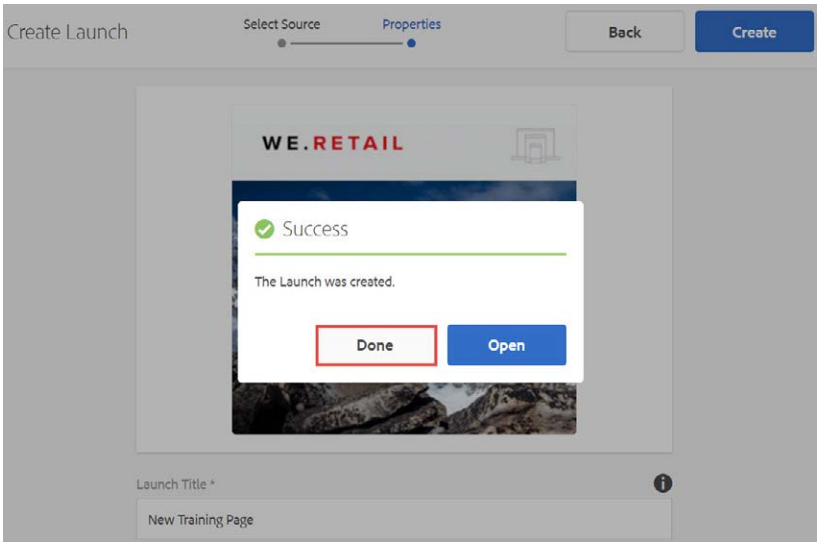


2. In the **Create Launch** step of the wizard, specify:
   a. **Launch Title**: New Training Page_<username>
   b. **with existing content**
   c. Select **Inherit source page live data** option

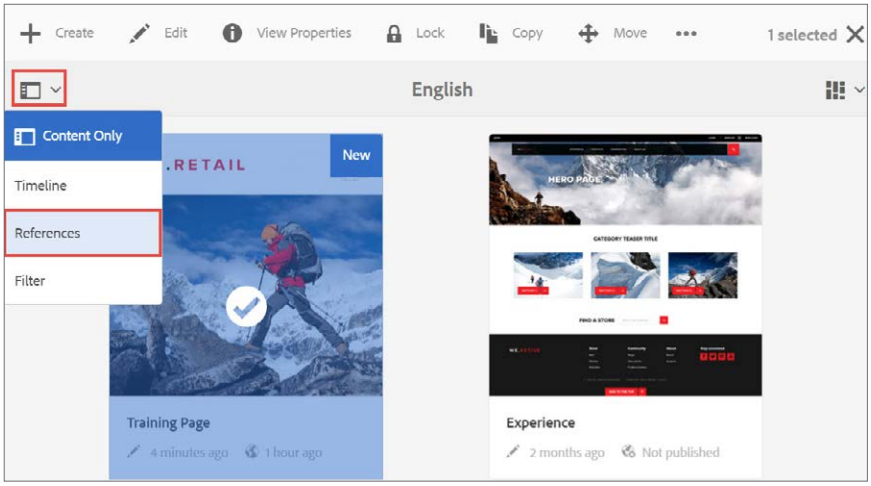


NOTE: It is important to create your launch with a unique identifier so you can identify your launch amongst those of your peers' launches being created in the same environment.

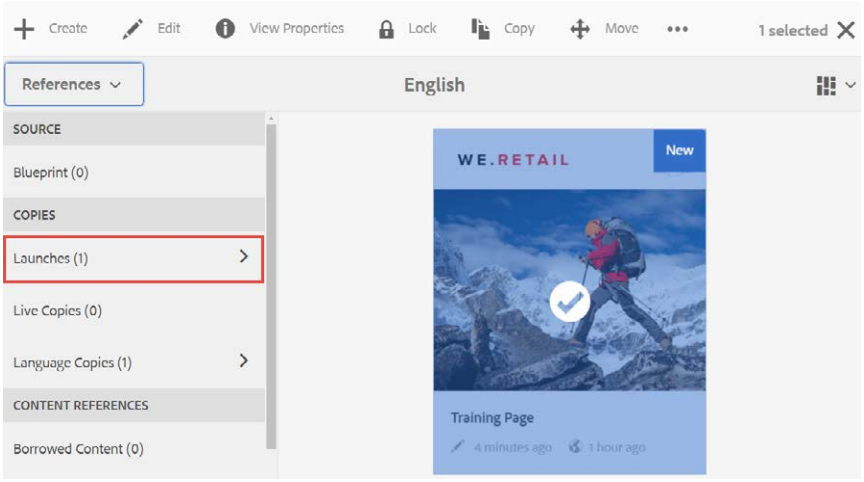
3. Click **Create** to complete the process. The confirmation dialog will ask whether you want to open the launch immediately. Click **Done**.

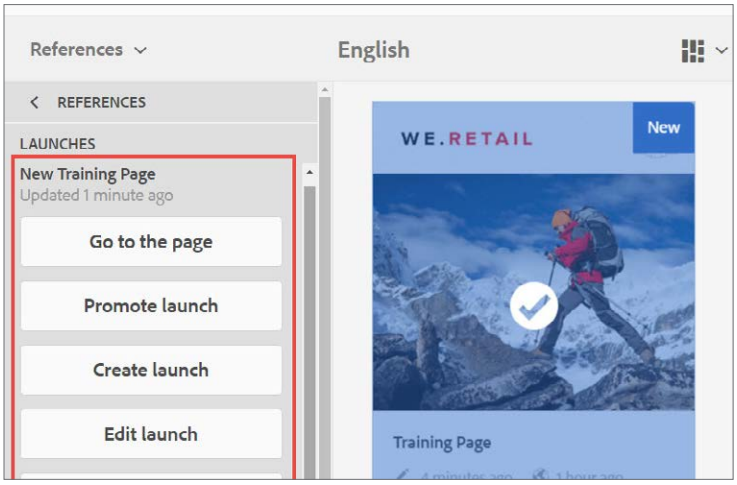


4. Return to the **Sites** console to access the launch click **Training Page_<username>**, and then click the left rail, and then select **References** from the drop-down.



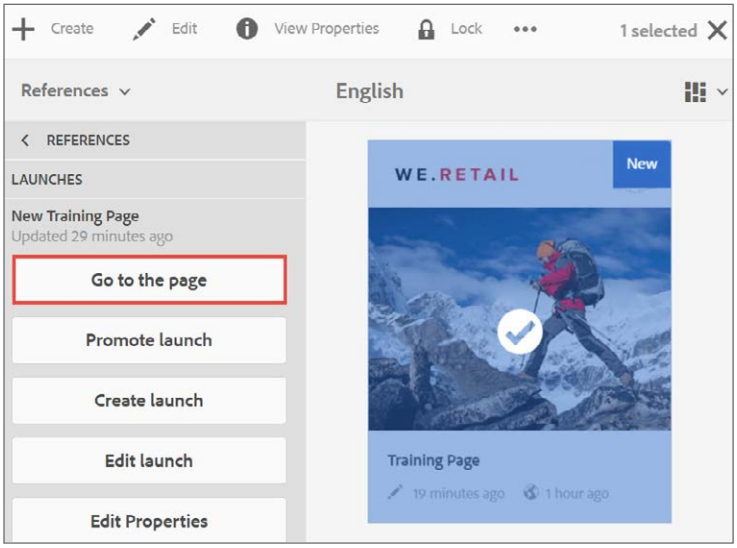
5. Click **Launches (1)**, to view **New Training Page_<username>**.



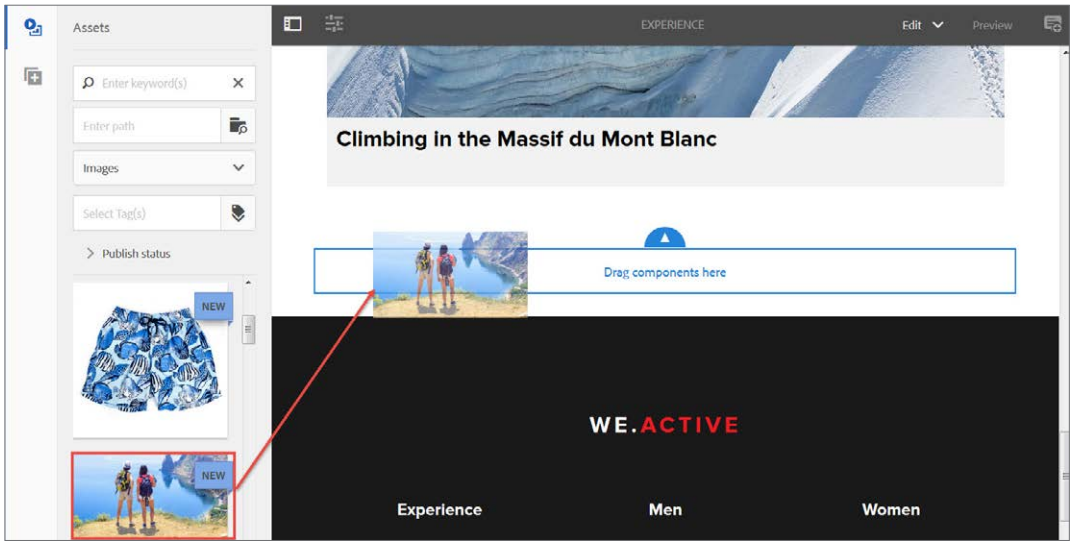


To edit a launch:

6. Click **Go to page** to edit and add new content to the launch. The launch page opens.

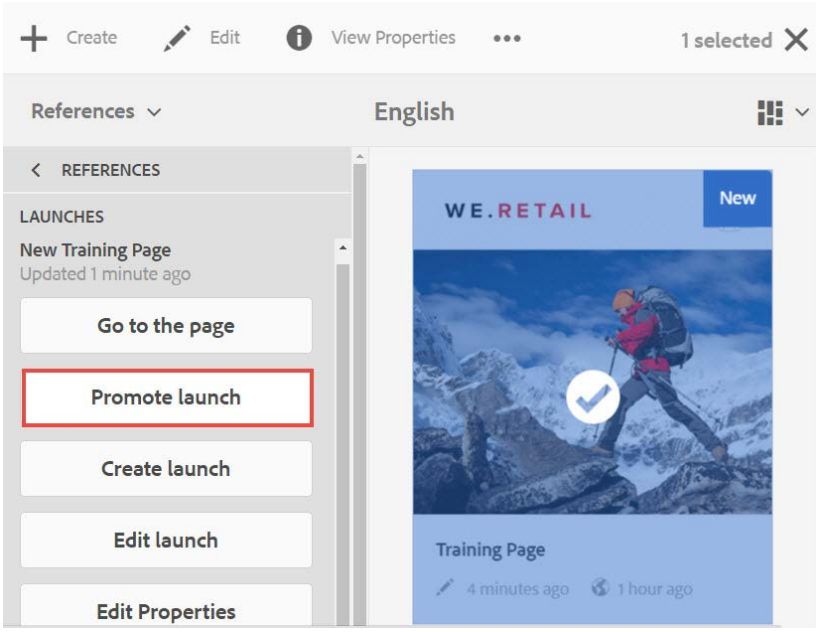


7. Add a few components to the launch.

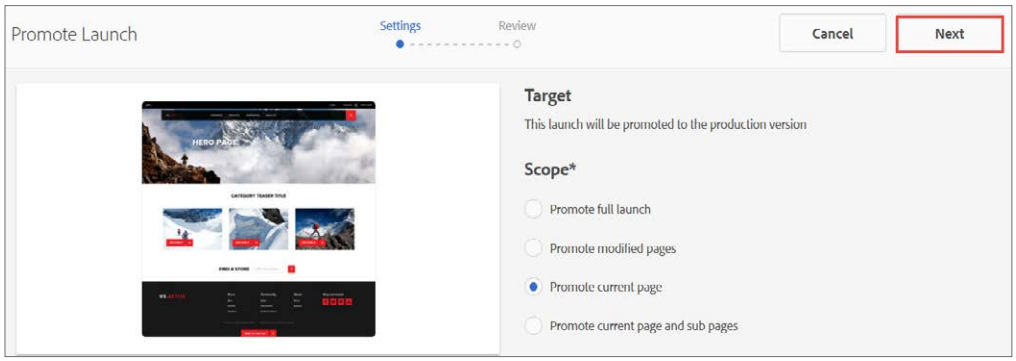


To promote a launch:

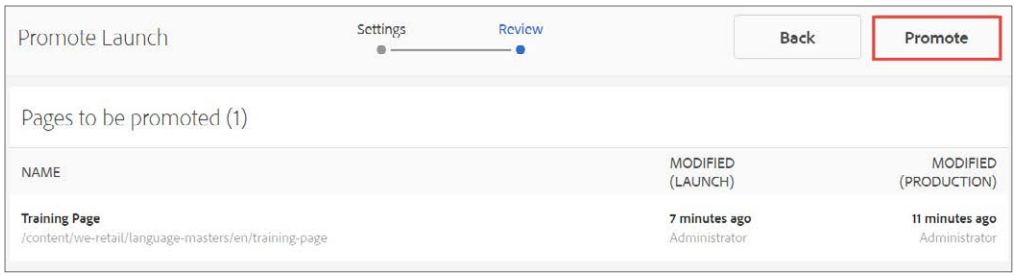
8. From the **Sites** console, navigate and select the **Training Page_<username>** for which you created a launch earlier.
9. Select **References** from the left rail. All references are shown.



10. Click **Launches** to see a list of the specific launches.
11. Select the specific launch **New Training Page_<username>** and click **Promote Launch** to open the wizard.

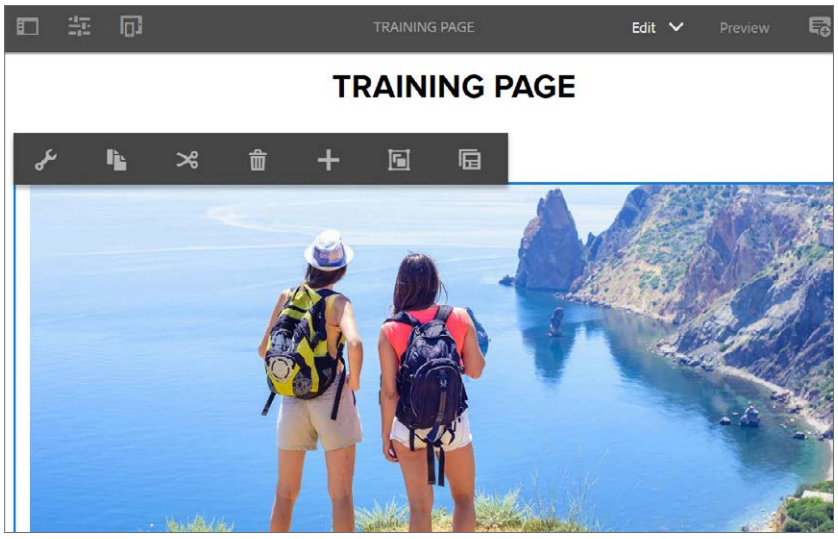


12. In the first step, specify the Scope of launch, select **Promote current page**, to only promote page changes for the current page,



13. Click **Next**.
14. In the second step, review the pages to be promoted.
15. Click **Promote**. The page is promoted to production.

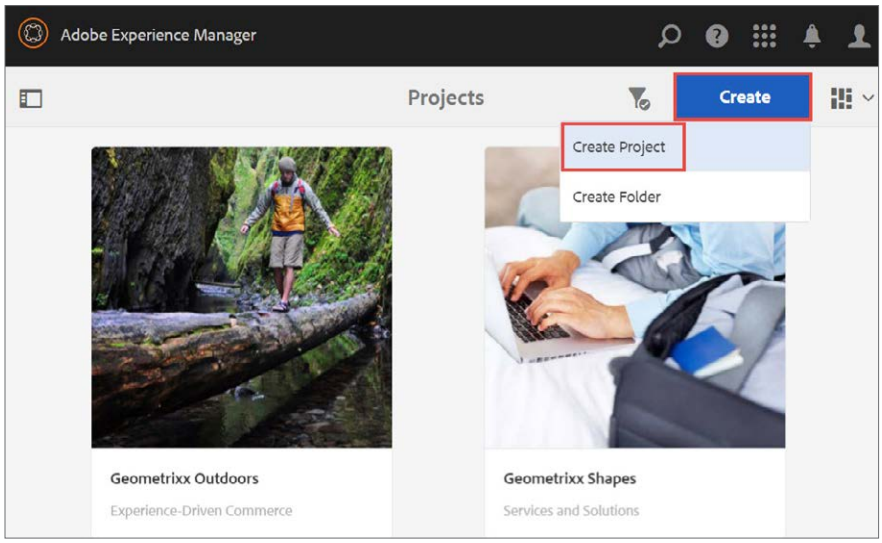
If you open the **Training Page_<username>**, you will see the content of the launch page (overrides the existing content).



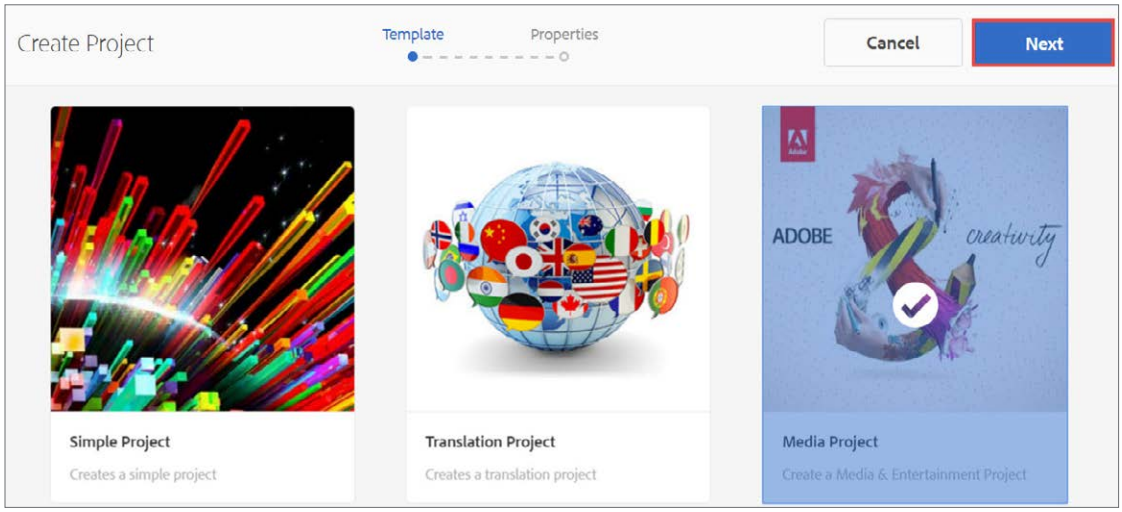
Task 5: Create a project, add members, assign a workflow, and perform the tasks

To create a project:

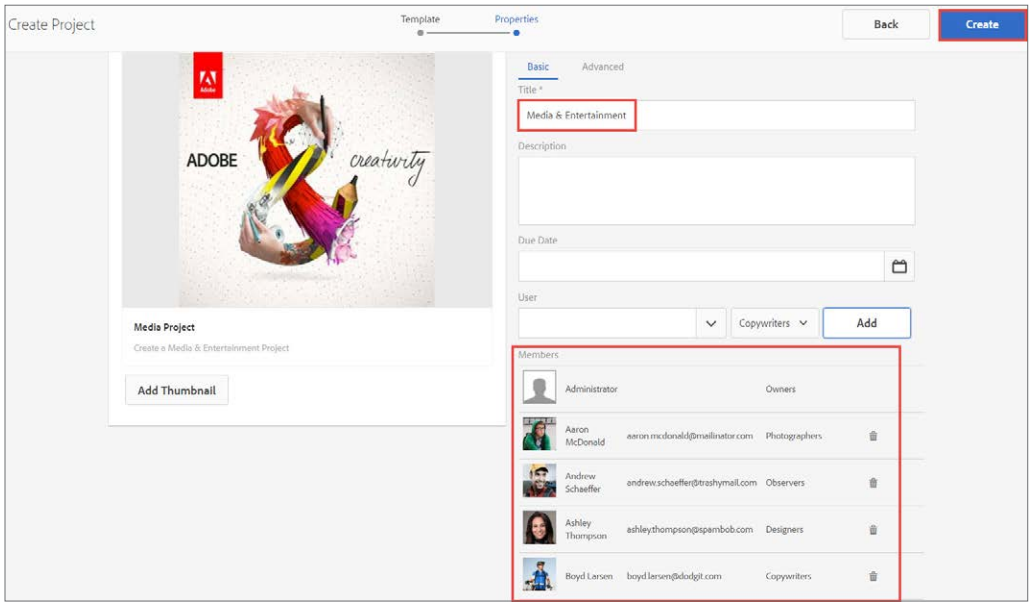
1. From the **Projects** console, click **Create** > **Create Project**. The **Create Project** wizard launches.



2. Select the **Media Project** template from the screen, and click **Next**.

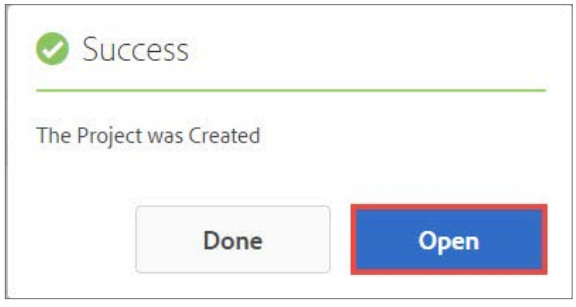


3. Enter the project **Title** as **Media & Entertainment_<username>**.
4. In the **User** drop-down list, select **users**, choose their **role**, and then click **Add**.
5. Click **Create**.

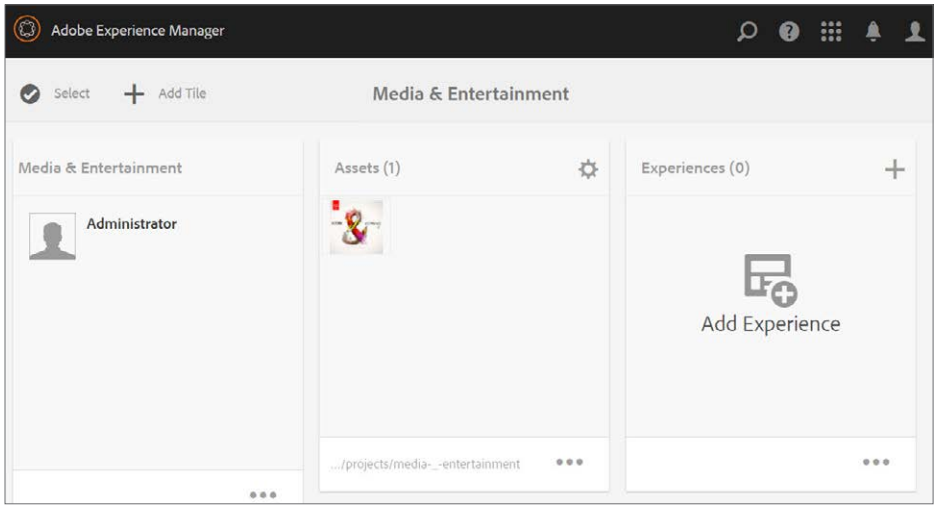


NOTE: It is important to create your project with a unique identifier so you can identify your project amongst those of your peers' projects being created in the same environment.

6. Click **Open** from the **Success** dialog box.

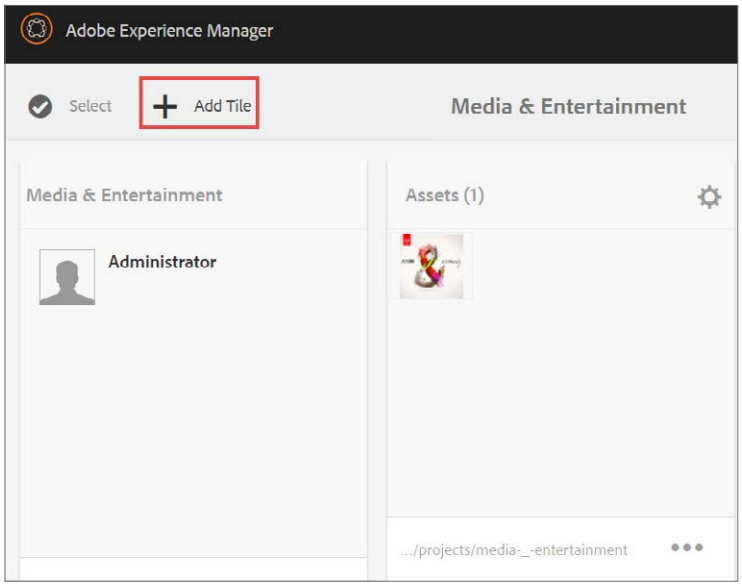


7. The **Media & Entertainment_<username>** project opens and you can view all the tiles associated with the project.

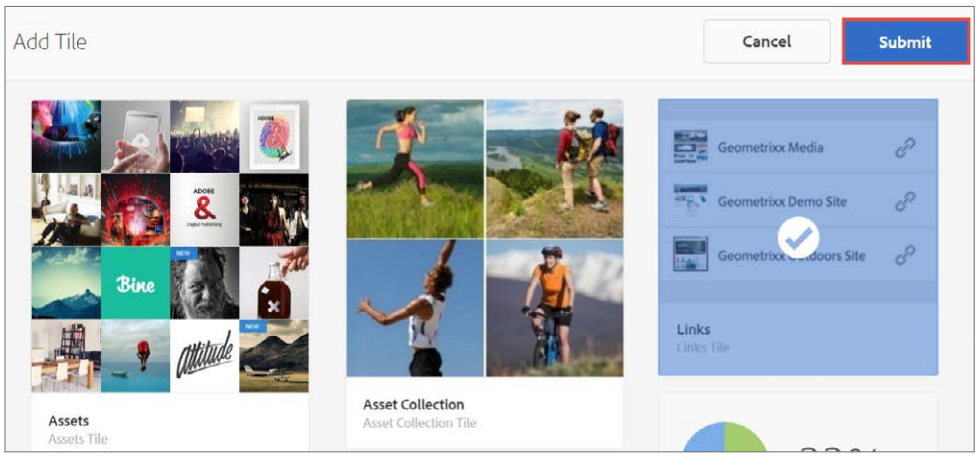


To add tiles to the project:

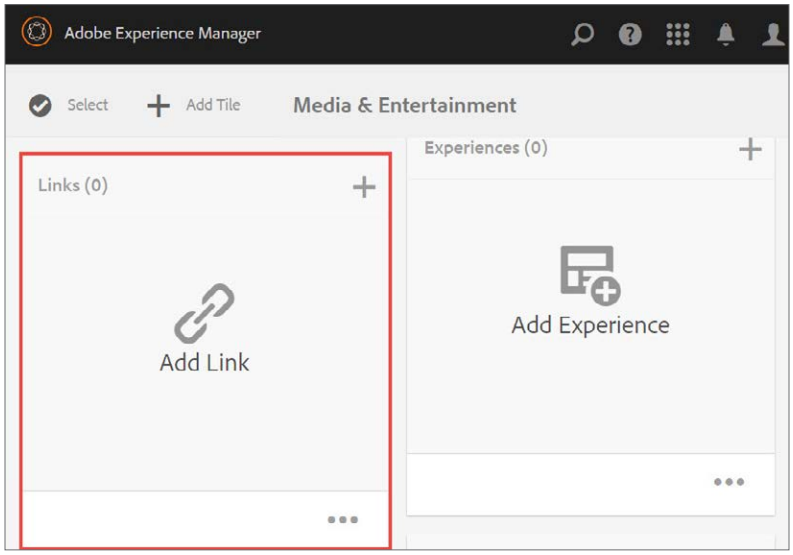
8. Make sure the **Media & Entertainment_<username>** project is open, and then click **Add Tile** from the actions bar.



9. Click the **Links** tile, and then click **Submit**.

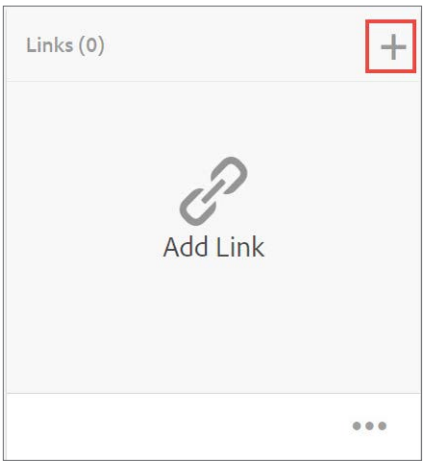


The Links tile is added to your project.



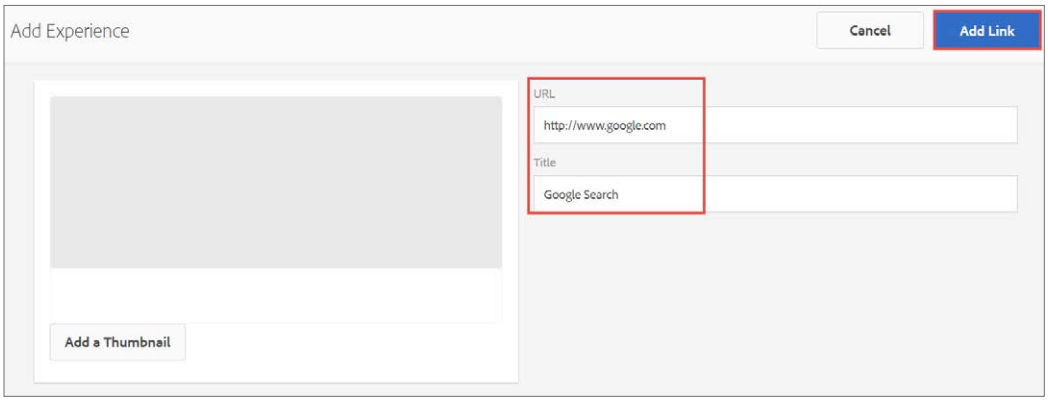
To edit tiles of the project:

10. To add an external URL for the project, click the + icon from the **Links** tile.

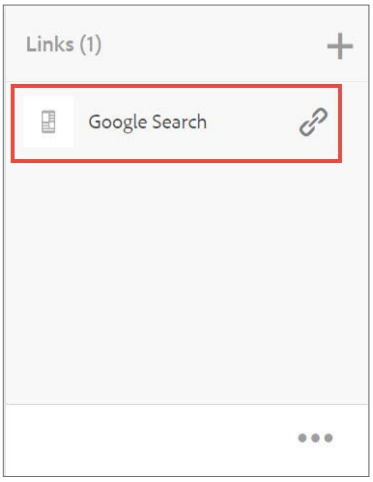


11. In the wizard, enter the following:
    a. **URL**: http://www.google.com
    b. **Title**: Google Search

12. Click **Add Link** to **complete** the process.

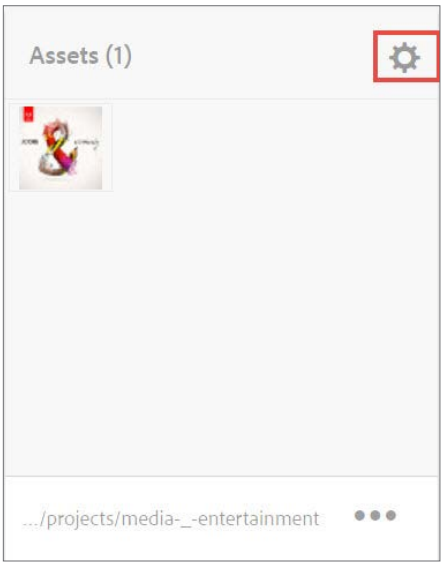


The new link is added and can be accessed by the members of the project from the Links tile.

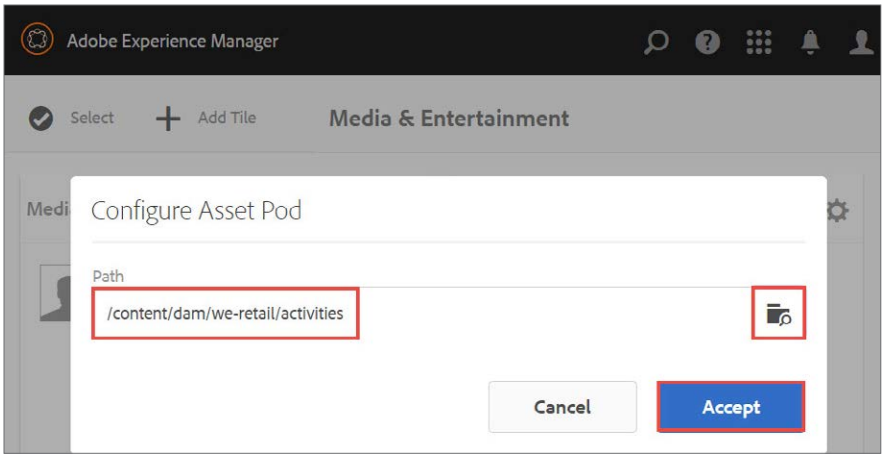


By default, you know the **Assets** tile points to the project's assets folder. Let's link the Assets tile to other asset folders of the Assets console.

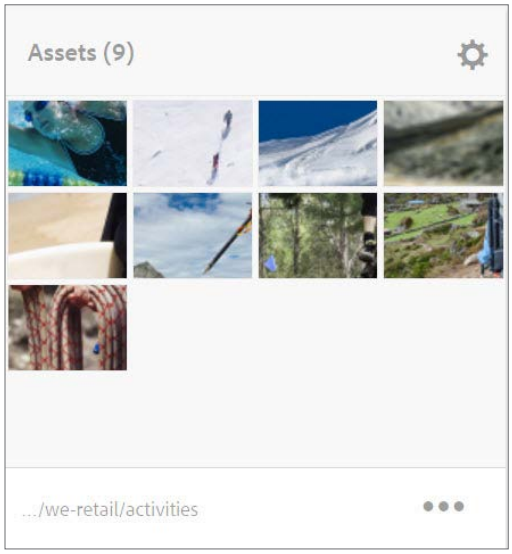
1. Click **Configure** (gear icon) in the **Assets** tile to link it to other folders. The **Configure Asset Pod** dialog box opens.



2. Click **Browse**. The **Select Path** dialog box appears.
3. Select **dam /we-retail / activities** path, and click **Accept**.

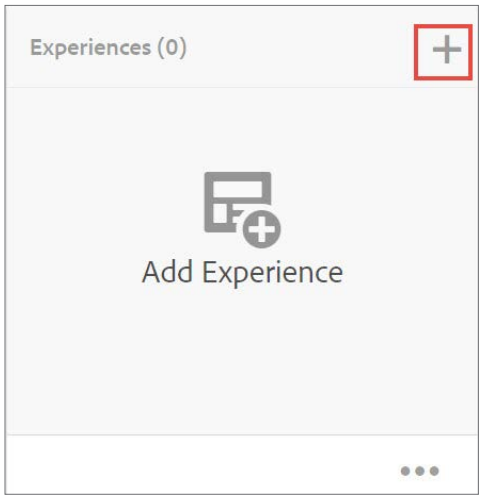


Now, the new Assets tile will have all the assets of **we.retail / activities**.

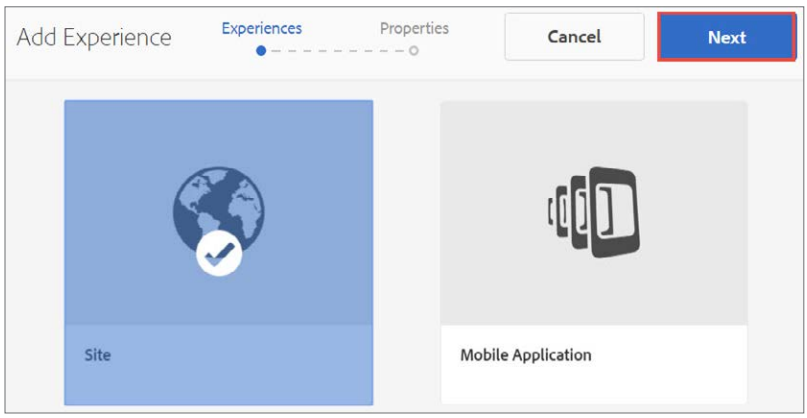


To add an experience to a project:

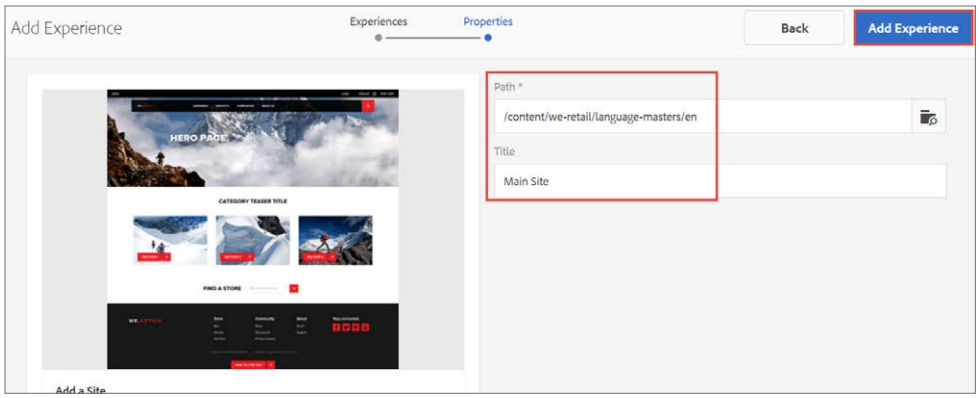
1. Click the + icon from the **Experiences** tile.



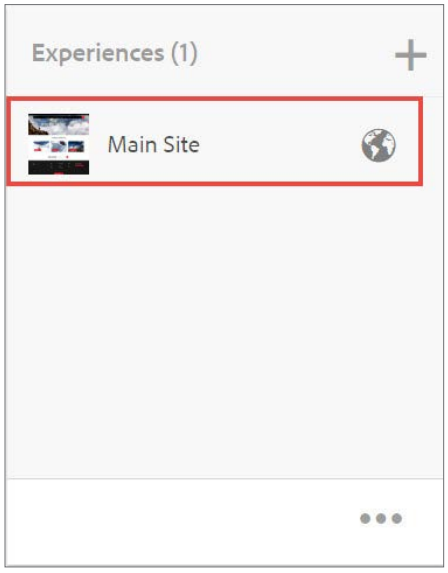
2. Select the type of experience you want to link (for example, **Site**), and click **Next**.



3. Enter the **Path** as **/content/we-retail/language-masters/en** to locate your website. Let the **Title** be **Main Site**.
4. Click **Create**. The site is now linked to your project and you will return to the **Experiences** tile.

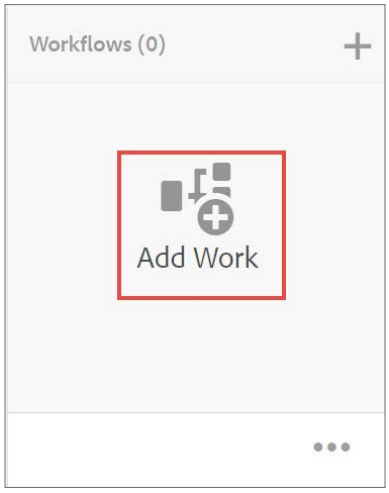


If you click **Site** from the Experiences tile, you will see the appropriate Adobe Experience Manager page linked with the site.

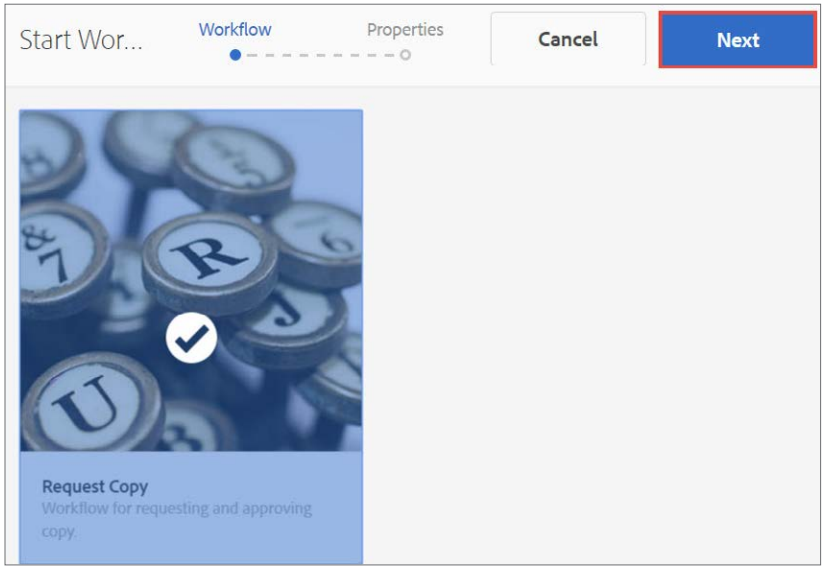


To initiate a workflow in a project:

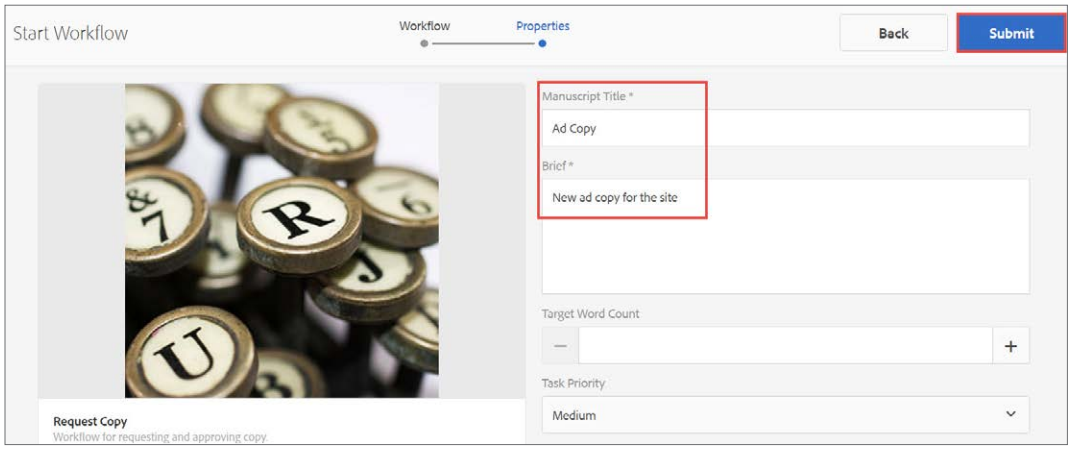
5. Click **Add Work** icon (or + from the Workflows tile) to start a workflow.



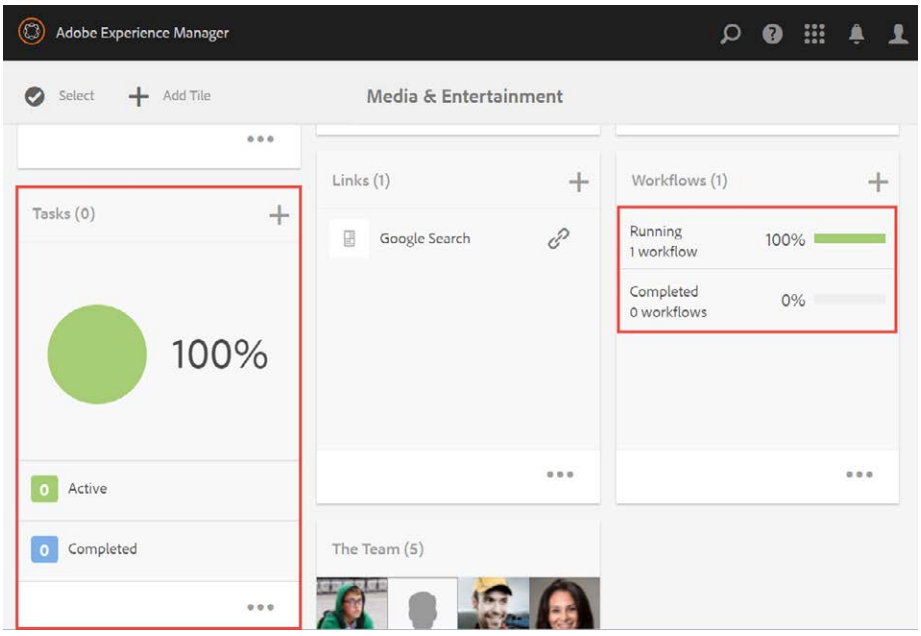
6. Select **Request Copy**, and then click **Next**.



7. Enter the following values:
   a. **Manuscript Title**: Ad Copy_<username>
   b. **Brief**: New ad copy for the site
8. Click **Submit**.

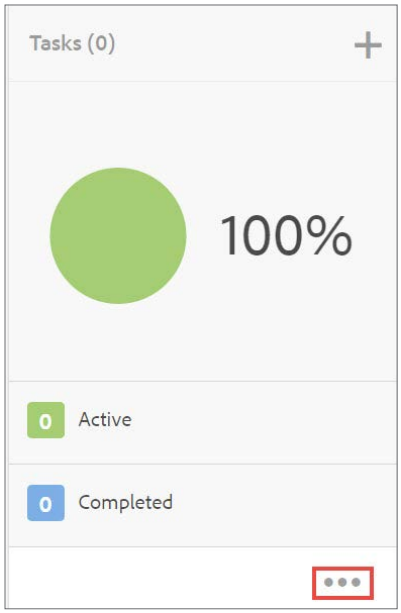


9. You will see the following changes in the project:
   a. The Workflows tile indicates that a workflow has started running.
   b. A new Tasks tile is added, which shows the status of tasks for this project.

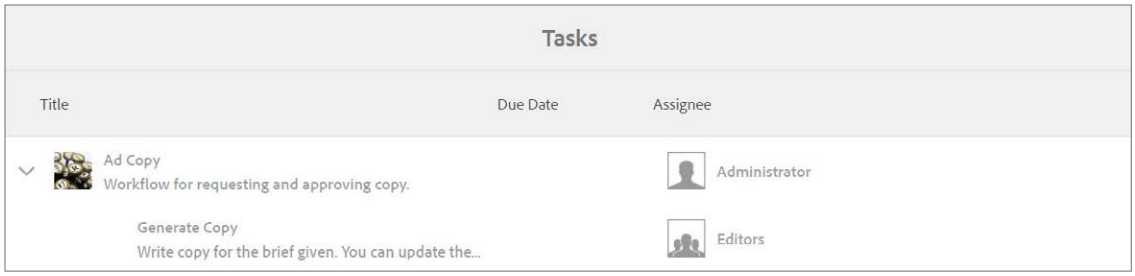


To complete the tasks assigned to the project:

10. Click the **...** icon in the **Tasks** tile to open the **Tasks** list screen. This screen displays all tasks related to the project.



Tasks associated with a workflow are grouped under the workflow entry. Notice that the **Generate Copy** task is nested in **Ad Copy** workflow.



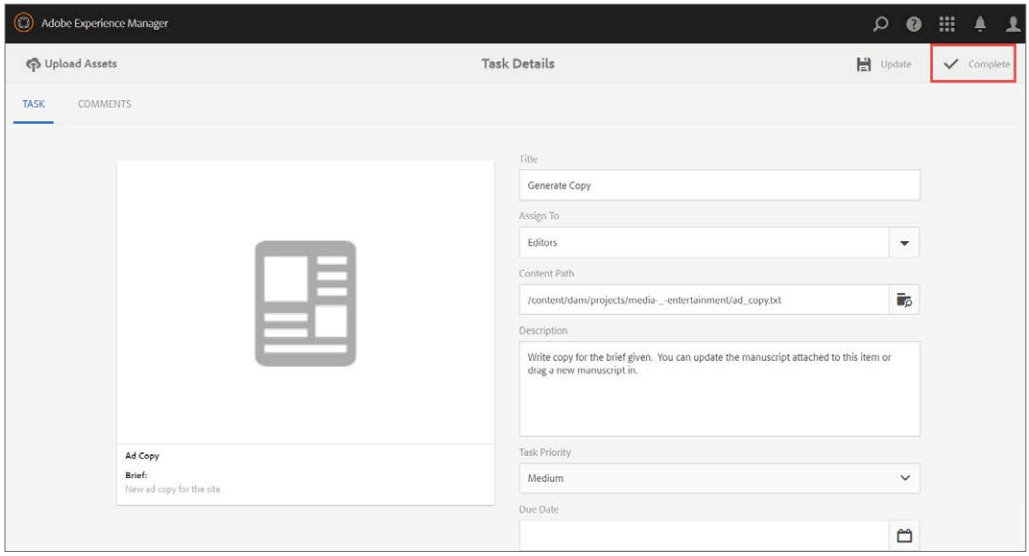
11. Click **Generate Copy** task to open the **Tasks Details** screen.

The Task Details screen is where users enter information, comment, and work on a task. It has two tabs:

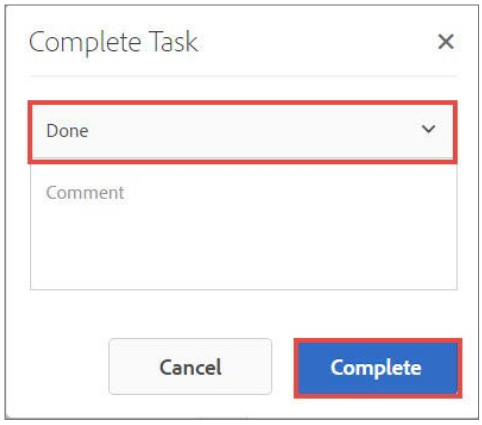
a. Tasks—to edit the properties of a task

b. Comments—to add quick task-related comment or question. Users can comment on a task even if they are not assigned a task.

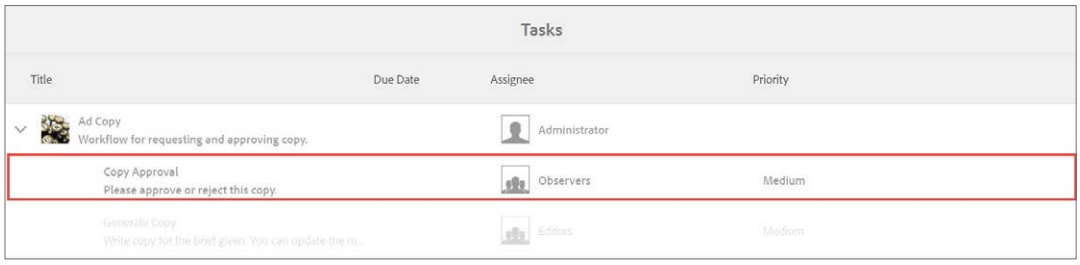
12. Click **Complete** to save the changes.



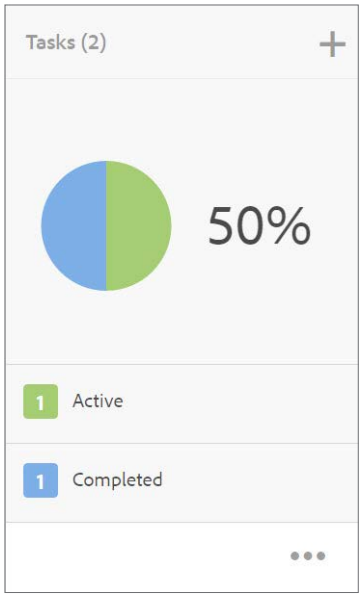
13. Select **Done** from the drop-down, and then click **Complete**. This returns you to the **Tasks** screen.



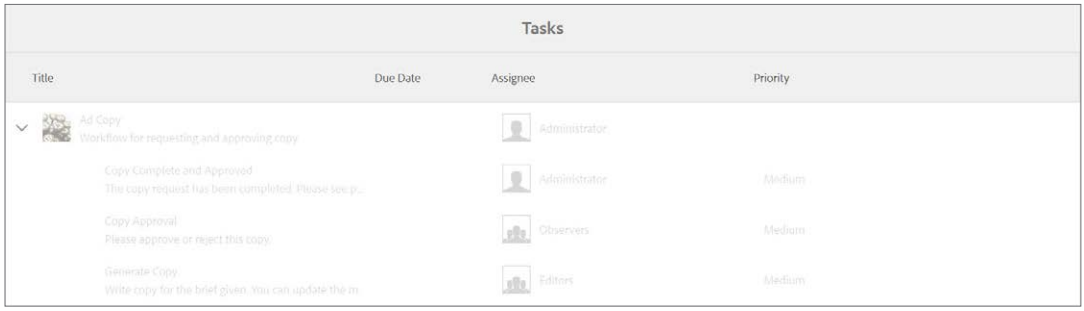
14. Notice that the task is completed, but a new one is also created. This is because the workflow moves to the next step in the procedure.



15. Navigate to the **Projects** console, open the **Media & Entertainment** project, and take a look at the progression of the workflow in **Tasks** tile of the project. You will notice the percentage and number of tasks Active and Completed.

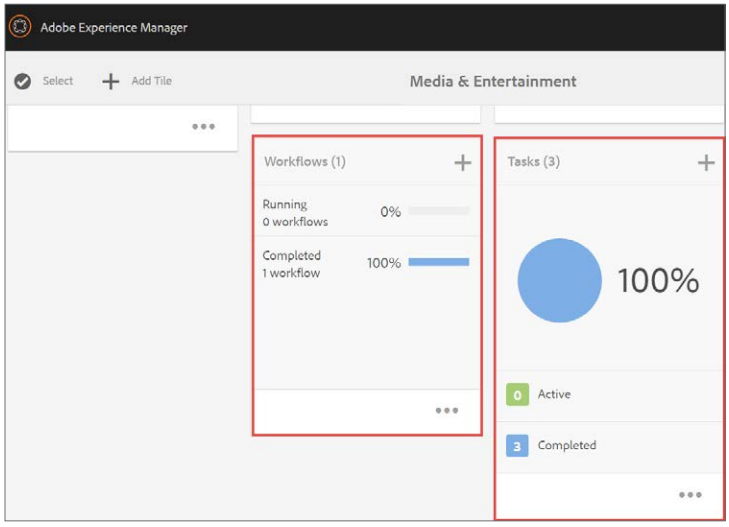


Follow the steps above and complete all the tasks of the workflow.



16. Once you complete all the tasks, navigate to **Media & Entertainment_<username>** project and observe **Tasks** and **Workflows** tiles.

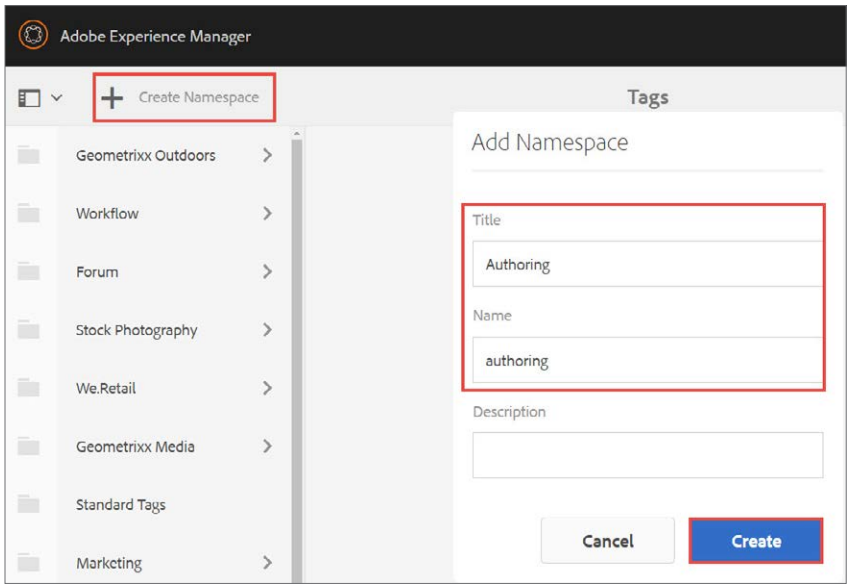
You will notice that the status of Workflows tile has changed to Completed 1 workflow - 100% and the Tasks tile now indicates that three (3) tasks are Completed. This means that all the tasks of the workflow completed successfully.



Task 6: Create a namespace, add tags, and apply tags to pages

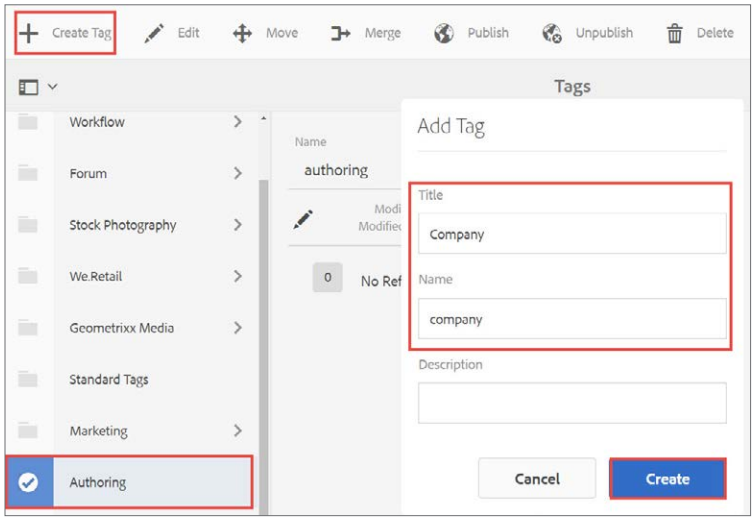
To create a namespace:

1. Navigate to **Tagging** console from **Tools** > **General** section
2. Click Create **Namespace** from the actions bar. The **Add Namespace** dialog box opens.
3. Provide a **Title** (for example, **Authoring_<username>**), the **Name** gets auto-generated, and then click **Create**. The new namespace is added to the Tagging console.

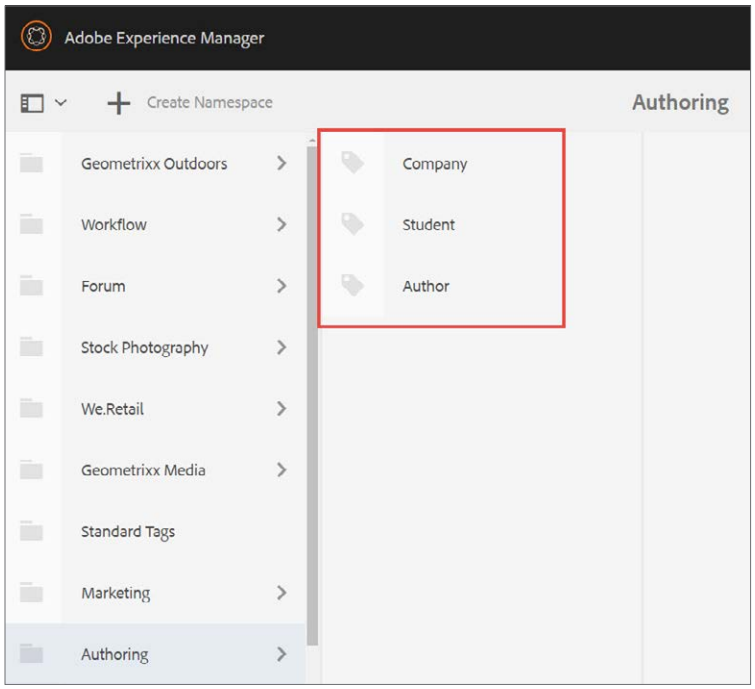


To add tags to the namespace:

4. Select the **Authoring_<username>** namespace.
5. Click **Create Tag**. The **Add Tag** dialog box opens.
6. Provide a **Title** (for example, **Company_<username>**), the **Name** gets auto-generated, and then click **Create**.

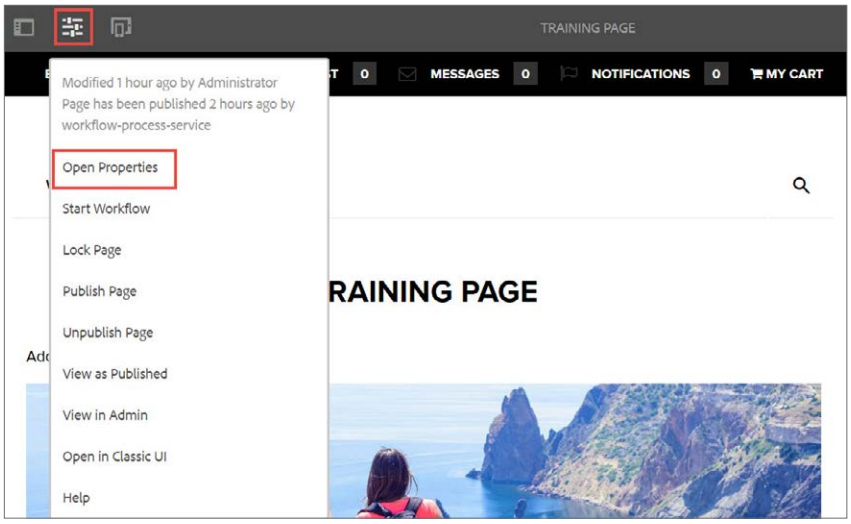


7. To add more tags to **Authoring_<username>** namespace, repeat steps 4 through 6.

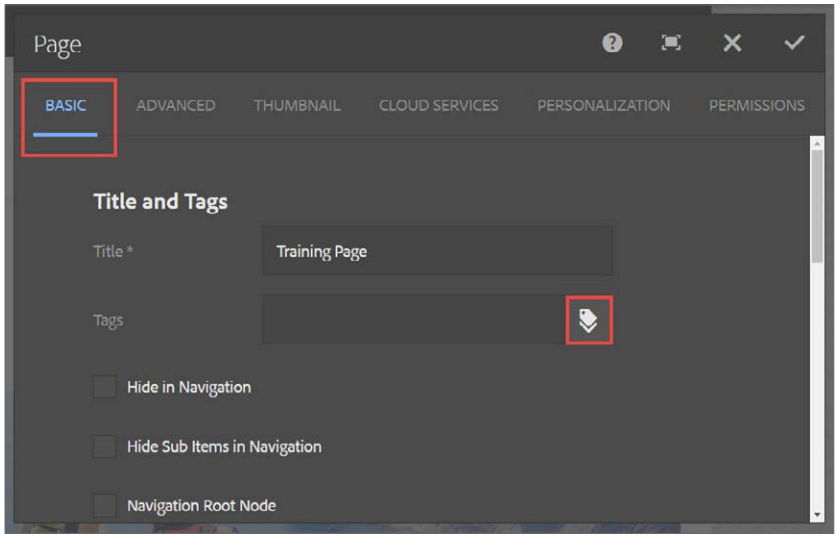


To add tags to a page:

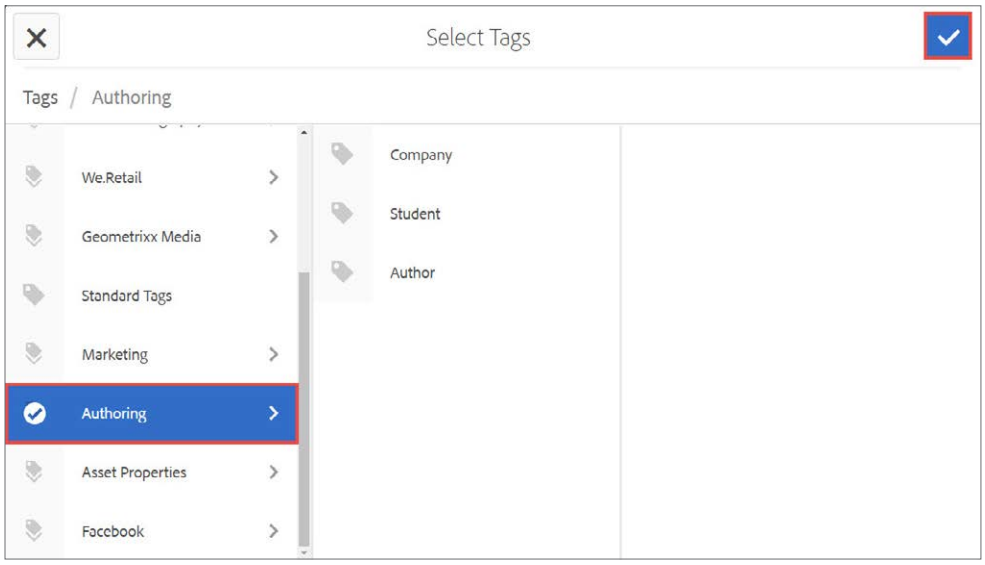
8. Open the **Training Page_<username>**.
9. Click the **Page Information** icon from the toolbar, and select **Open Properties** from the drop-down. The **Page** dialog box opens.



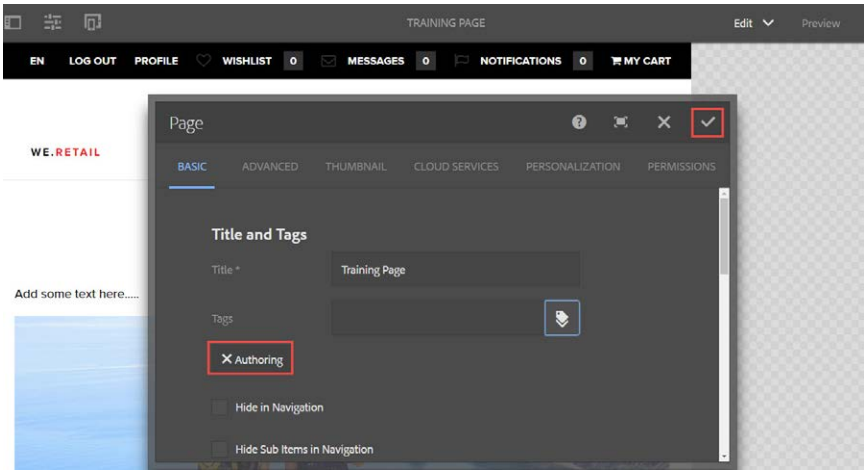
10. On the **BASIC** tab, click **Browse** from the **Tags** field to add some tags to the page. The **Select Tags** dialog box opens.



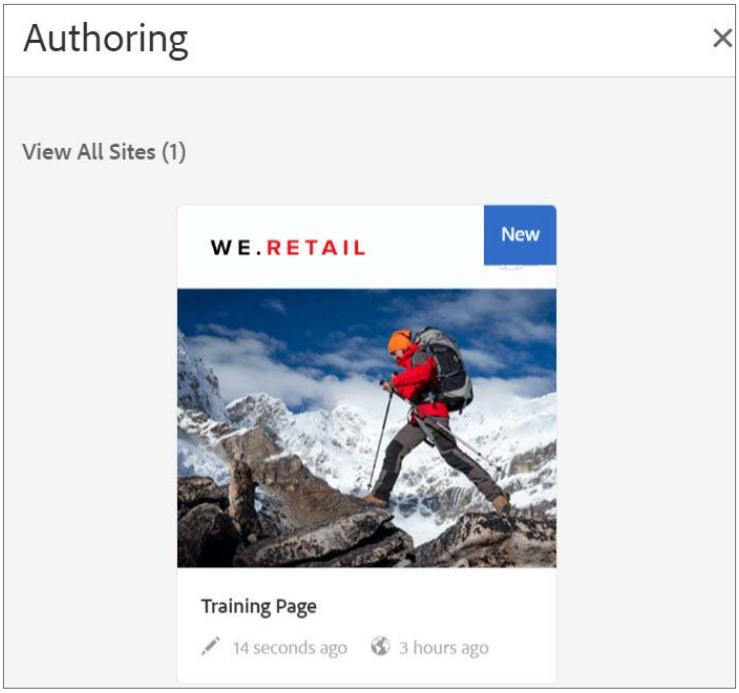
11. Select all the tags from the **Authoring_<username>** namespace that you created earlier, and then click **Confirm**. You can select multiple tags using Ctrl+left click on the tags you want to add.



12. Click **Done** in the **Page** dialog box. All the tags of **Authoring_<username>** namespace are added to the page.



13. Go back to the **Sites** console, and search for tags (for example, **Authoring_<username>**). You will notice the **Training Page_<username>** appears as a search result since it has all the tags of Authoring_<username> namespace.



Scenario Conclusion

By performing these tasks, you:

- Created an editable template to have intuitive and flexible page structure
- Created and used content fragments to optimize your content according to the specific channel.
- Created an efficient workflow model reflecting the business processes.
- Developed content for future releases.
- Created and managed projects in your WCM to ensure that the tasks in your project life cycle are completed as expected.
- Created and managed tags to provide seamless search functionality embedded in the website.

Summary

You should now be able to:

- Define Template Editor
- Create and use editable templates in pages
- Explore Design mode of pages
- Create and use Content Fragments in pages
- Define workflows
- List the interface elements of Workflow console
- Create and use a workflow to automate business processes
- Define launches
- Create and promote launches
- Create and promote nested launches
- List the interface elements of Projects console
- Create and manage projects
- List the interface elements of Tagging console
- Create and add tags to a namespace
- List the reasons to use tags in pages

Personalization and Content Targeting

Overview

This module shows you how to use the framework of tools for authoring targeted content and presenting personalized experiences in Adobe Experience Manager

Objectives

By the end of this chapter, you will:

- Define personalization and its use in today's world
- Define the tools used for personalization and content targeting
- Explore Personalization console
- Define ContextHub
- Explore user profiles in ContextHub
- Explain the process to create activities
- Map audiences with experiences
- Simulate an experience using ContextHub
- Define landing pages
- Create and import landing pages
- Perform various actions of landing pages

Introduction to Personalization

There is an ever-increasing volume of content available today, be it on Internet, Extranet, or Intranet websites.

Personalization centers on providing the user with a tailor-made environment displaying dynamic content that is selected according to their specific needs - whether it's based on predefined profiles, user selection, or interactive user behavior.

Personalization involves three main elements:

1. Users:
   - › Have profiles, both individual and group. These profiles contain characteristics (such as job description, location, and interests) that can be used to personalize the content they can see and take actions. These can then be analyzed and matched against behavior rules to tailor the content they see.
2. Content:
   - › Is what the user wants to see. Preferably content of interest and use to them for fulfilling their tasks.
   - › Can be categorized, and therefore made available to users according to predefined rules. The content must be dynamic; in other words, the content must, in some way, be dependent upon the user – if every user sees the same content, then personalization would be redundant.
3. Rules:
   - › Define how personalization actually happens – which content the user can see, and when.

Personalization can be used in many cases, for example:

- Intranet pages:
  - › Content can be proffered based on the location, department, and/or role of a user – already defined within an internal network.
  - › Dependent on the choice available, the user can make further selections.
- Specific, limited, target user groups (extranets):
  - › Users require a login for authorization; this is linked to a profile providing information required for personalization; possibly details such as their location, relationship to the product, usage history, budgeting responsibilities, etc.

    - Such instances can range over sites such as:
        - Companies that provide websites to a highly specialized section of their market (for example, a pharmaceutical company providing a specialized website for physicians).
        - Companies that provide websites allowing their customer to view current account and billing information (for example, telephone providers).
- Sales and Distribution website:
    - Sales and distribution websites, such as Amazon, can combine a user profile, the user's sales history, and their browsing history to make suggestions as to what might interest the user next.
- Search websites
    - Many of the major search engine websites have very powerful analytical tools that record user behavior, the search terms they use, and the websites they actually visit. This is then used to customize the content provided – particularly with regard to displaying advertisements.

Personalization and Content Targeting Tools

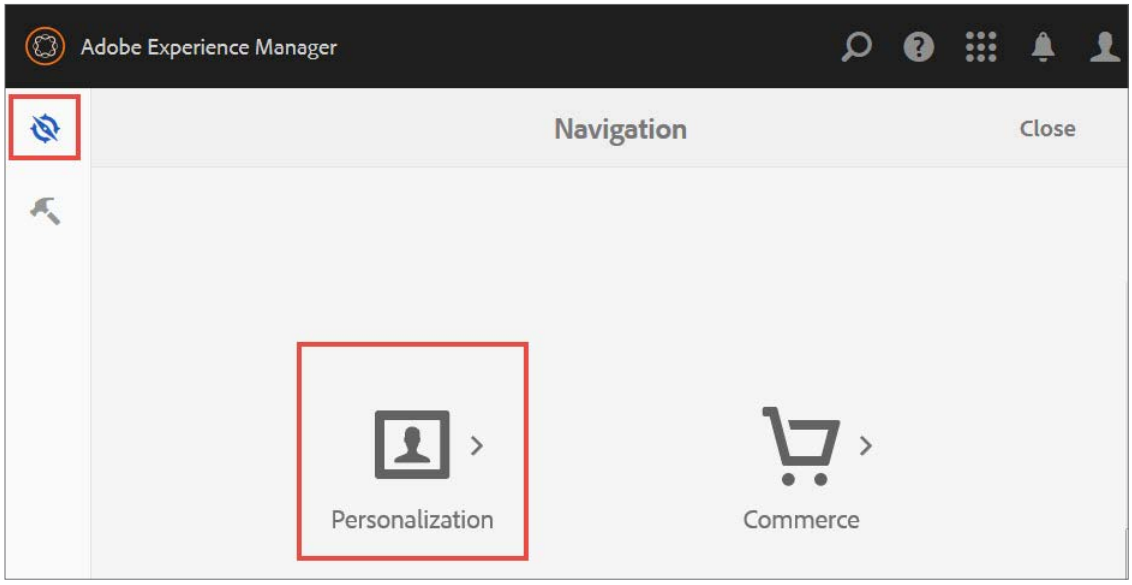
Adobe Experience Manager provides a framework of tools out-of-the-box for authoring targeted content and presenting personalized experiences:

- **Brands**—are a collection of activities.
- **Activities**—define and organize your marketing efforts. It comprises the audiences that you are targeting, and the period of time when the targeting is applied.
- **Experiences**—for each activity, you define one or more experiences that identify the audiences that you are targeting.
- **Offers**—are the content that appears at a location on a page for an experience. Use different offers for different experiences to maximize the effectiveness of the content for your audiences.
- **Audiences**—are based on marketing segments created in either Adobe Experience Manager or Adobe Target.
- **Targeting Mode**—provides tools for creating content for the experiences of your marketing activities.
- **Targeting Engine**—is the mechanism that drives the logic for targeted content. Two targeting engines are available:
    - › Adobe Experience Manager—has a built-in targeting engine that processes page requests and determines the content to display.
        - » You can use the segments created in Adobe Experience Manager for defining the audiences of your experiences.
    - › Adobe Target—uses information gathered from page visits.
        - » You can use the segments that you import from Adobe Target to define the audiences for your experiences.
        - » Activities that use Adobe Target engine are synchronized to Target.

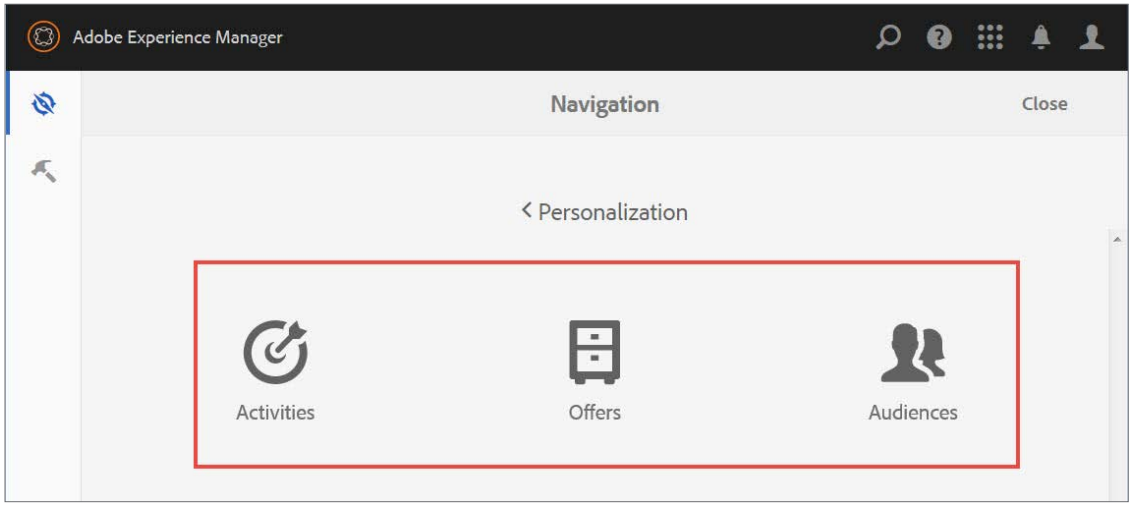

NOTE: You can use Target engine, only when you have integrated Adobe Experience Manager with Adobe Target.

Personalization Console

On installing Adobe Experience Manager, you can see the **Personalization** console from the **Product Navigation**. You can use this console to author targeted content and present personalized experiences.



You can navigate within the **Personalization** console by using the arrowheads (>).

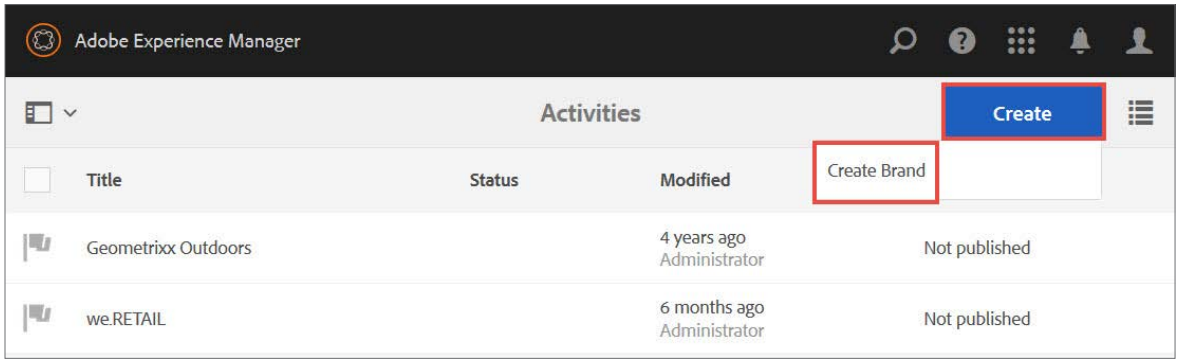


The Personalization console has the following consoles:

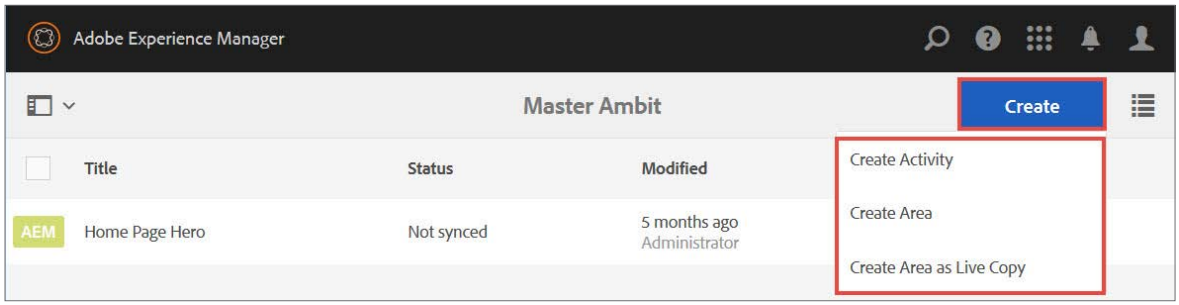
Activities

Lets you create, organize, and manage the marketing activities of your brands. Using the Activities console, you can:

- Add brands

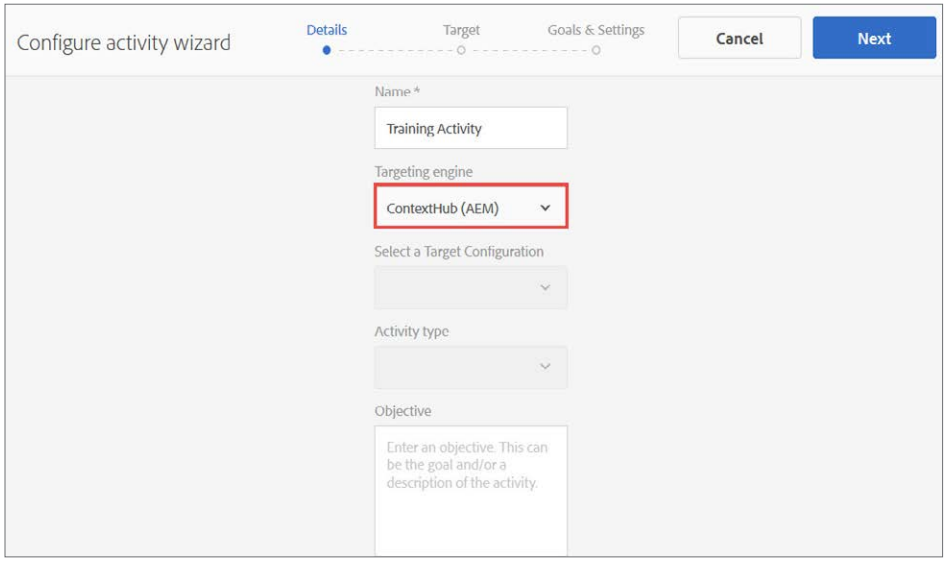


- For each brand, you can add, edit, publish, unpublish, configure, and administer activities.

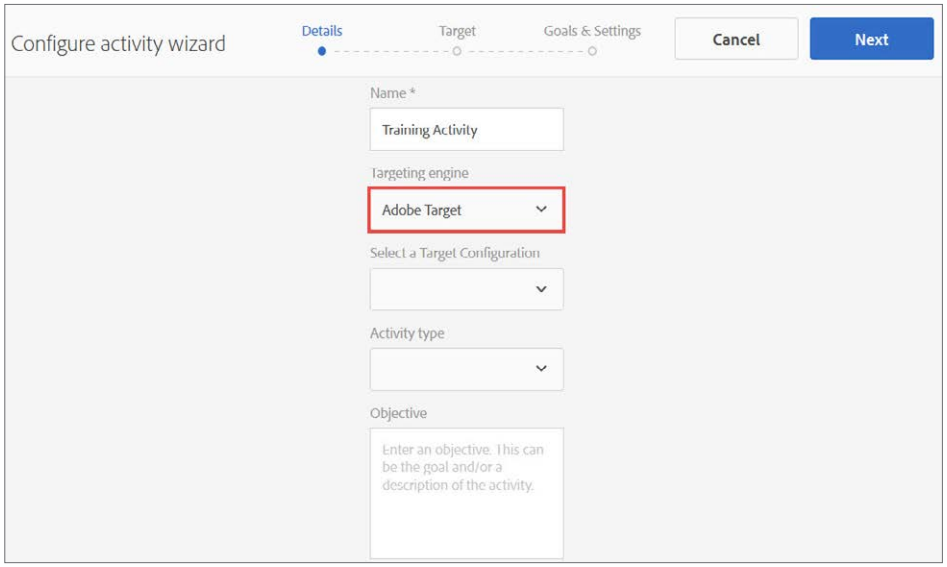


- Create Area and Create Area as Live Copy allows you to manage as activities, experiences, and offers between the sites, you can take advantage of Adobe Experience Manager's built-in multisite support for targeted content.
- **Areas** separate targeted content (activities, experiences and offers) used in different sites and provide an Multi-Site Manager (MSM)-based mechanism to create and manage the inheritance of targeted content together with site inheritance.
  - This prevents your having to re-create targeted content in inherited sites as was required in Adobe Experience Manager prior to 6.2.
  - In an area, only activities linked to that area are pushed to live copies. By default, the Master Area is selected. After you create additional areas, you can link those to your sites or pages to indicate which targeted content is pushed.
- A site or **Live Copy** links to an area containing the activities that need to be available on that site or live copy. By default the site or live copy links to the master area, but you can link other areas besides the master area as well.

- Add one or more experiences to the activity.
  - › You can add experiences from ContextHub (AEM) and Adobe Target engines.
    - » Using ContextHub (AEM) engine: This lists all the existing segments and experiences available within Adobe Experience Manager.



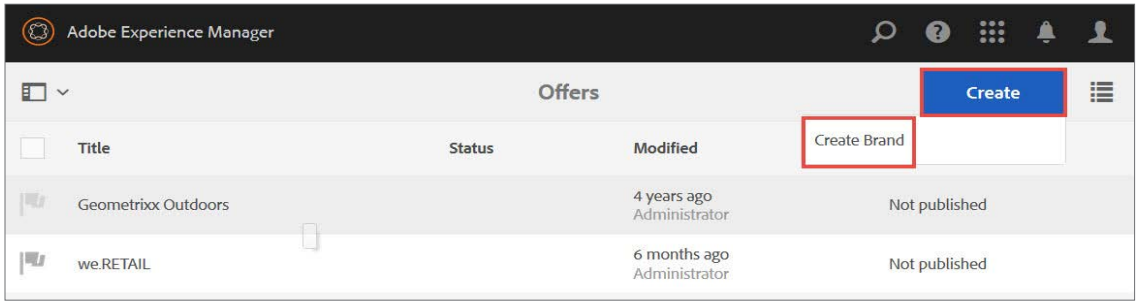
    - » Using Adobe Target engine: You need to specify Target Configuration and you must have an Adobe Target account to access the segments).



Offers

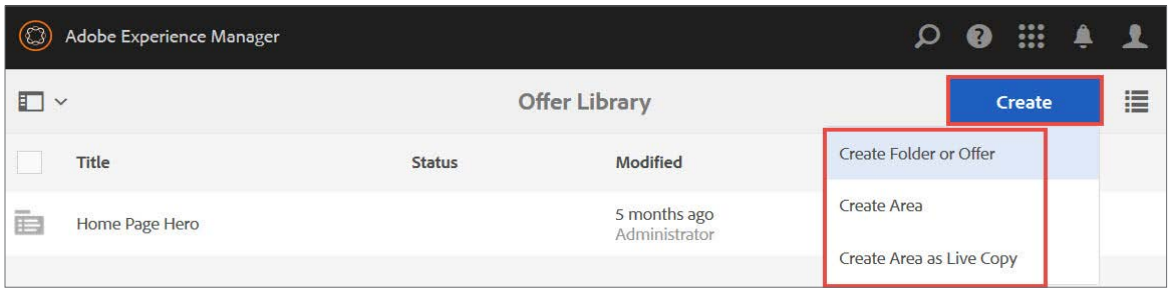
Allows you to create offers that can be used in activity experiences. Using the Offers console, you can:

- Add brands
  - › When you create a brand using the Offers console, it also appears in the Activities console, where you can add and administer activities for the brand.
- Add a folder to an Offer Library—to organize and store offers. You can create a folder under



Brand, or under any other folder.

- › Create an offer once in the library and use it in multiple experiences of your brand activities.
- › You can edit, search, and delete offers.
- › Similar to Activities, you can **Create Area** and **Live Copy** of offers that can be used across multiple sites.



Audiences

Enables you to create, organize, and manage audiences for your Adobe Target account or manage segments for ContextHub.

Audiences, called segment in ContextHub and ClientContext, is a class of visitors defined by specific criteria, which then determines who sees a targeted activity. When you target an activity, you can either select audiences directly in the Targeting process or create new ones in the Audiences console.

In the Audiences console, audiences are organized by brand. Audiences are available in Targeting mode for authoring targeted content, where you can also create audiences (but you need to create Adobe

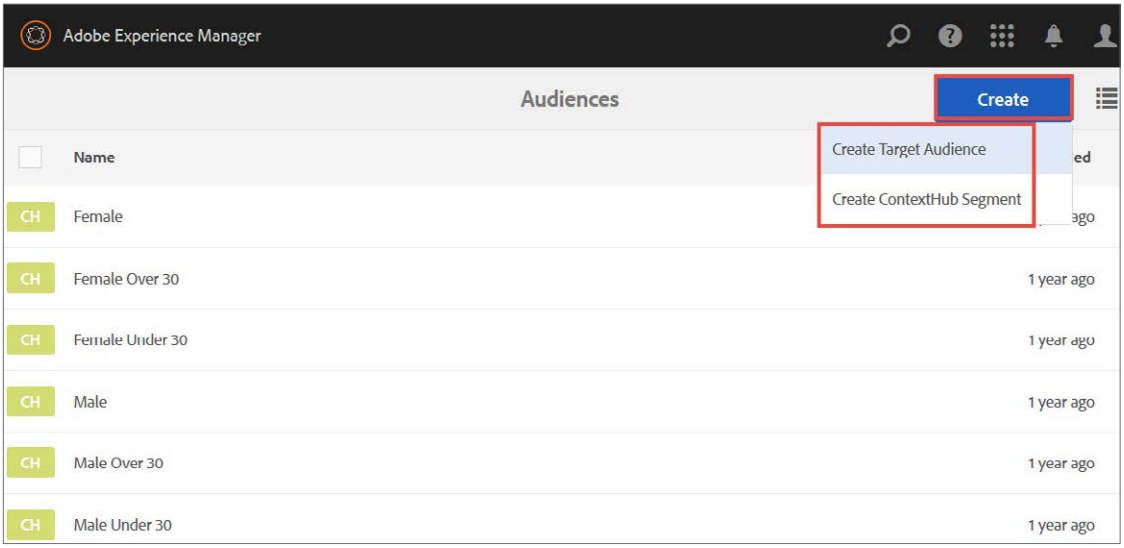
Target audiences in the Audiences console). Audiences that you create in Targeting mode appear in the Audiences console.

Audiences are displayed with a label describing what kind of audience is defined:

- CH - ContextHub segment
- CC - Clientcontext segment
- AT - Adobe Target audience

Using Audiences console:

- Add Audiences - either Adobe Target audiences or ContextHub Segments.
- Manage audiences.

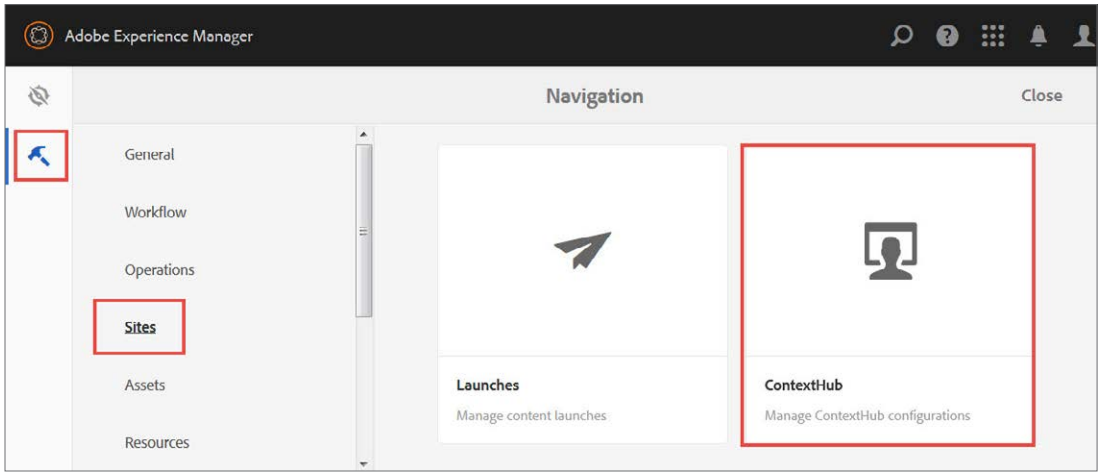


ContextHub

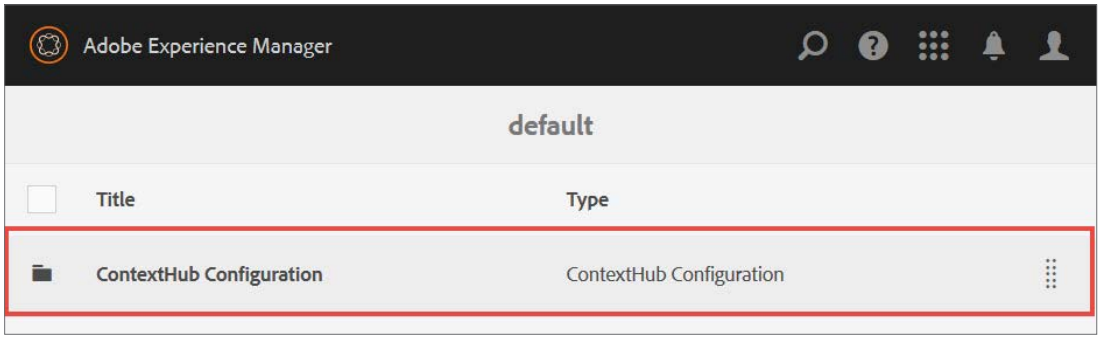
ContextHub is a framework that lets marketers access page and visitor information. It allows marketers to simulate visitor experience—what they want to see, and what they actually see on a page. You will use ContextHub to test the experiences based on visitor interactions.

ContextHub Console

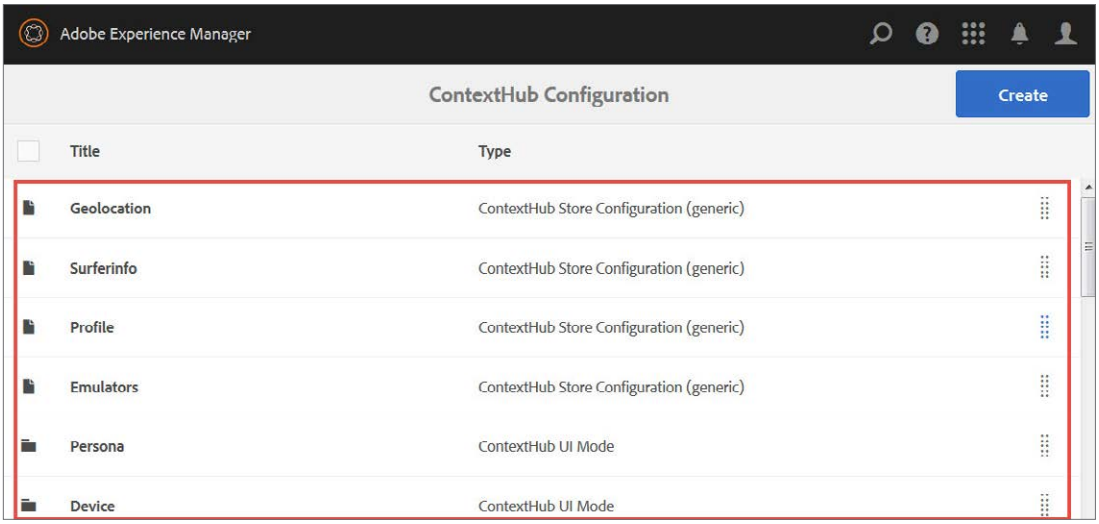
You can access and view **ContextHub** from the **Sites** section of **Tools** console.



ContextHub console allows you to manage various configurations from **Configuration Container.**



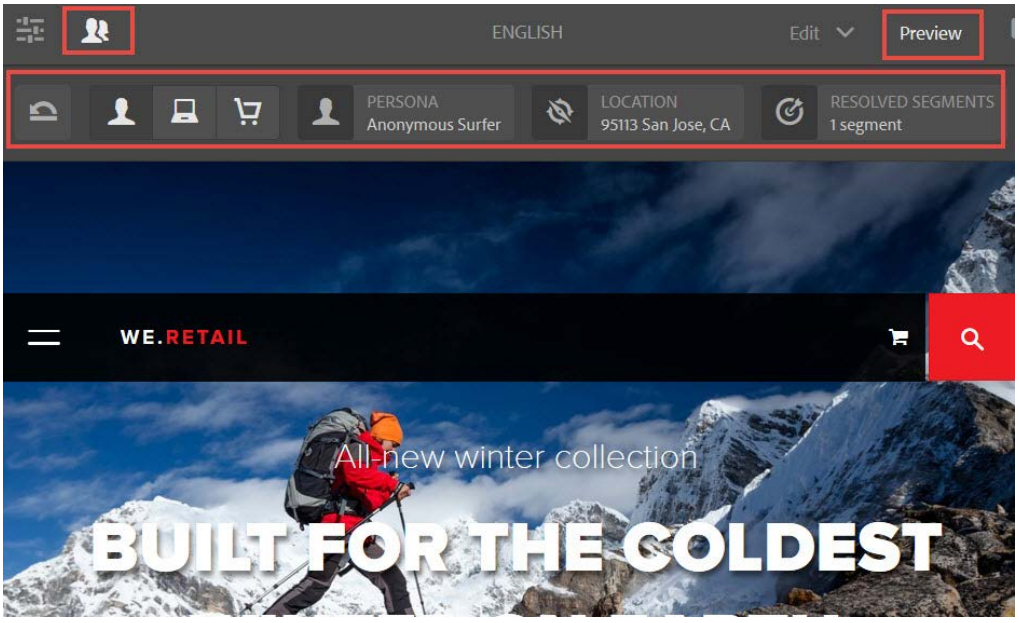
ContextHub consists of ContextHub stores to persist context data on the client and UI modes to access the store.



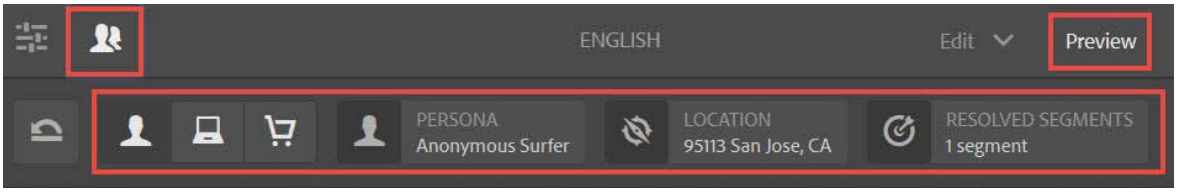
ContextHub Toolbar

In a page, you can view the **ContextHub** toolbar in **Preview** mode.

The ContextHub toolbar displays data from ContextHub stores and enables you to change store data. It is useful for previewing content that is determined by data in a ContextHub store.



The toolbar consists of a series of UI modes that contain one or more UI modules.

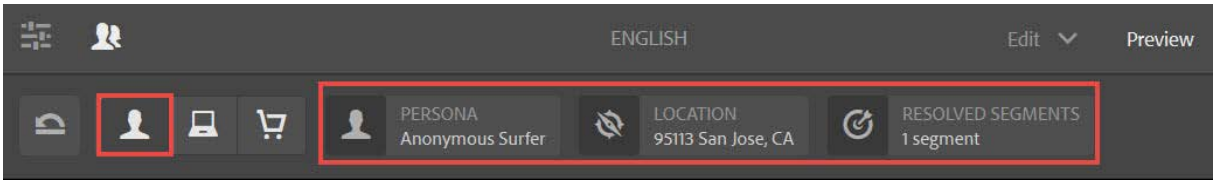


- UI modes are icons that appear on the left side of the toolbar. When you click or tap an icon, the toolbar reveals the UI modules that it contains.
- UI modules display data from one or more ContextHub stores. Some UI modules also let you manipulate store data.

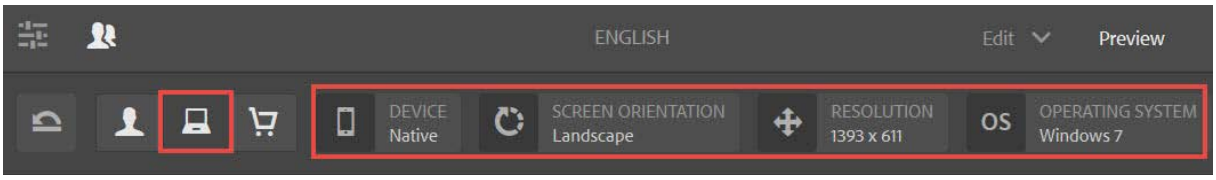
ContextHub UI Modes

ContextHub has the following UI modes:

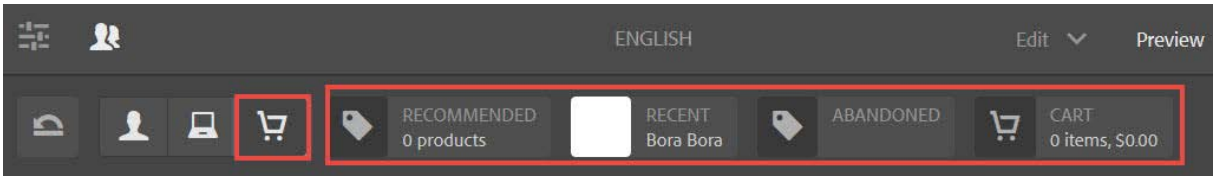
- **PERSONA**—provides information about the visitor such as profile, location, and the segments associated with the visitor.



- **DEVICE**—provides information about the visitor's device such as type of device and screen orientation.



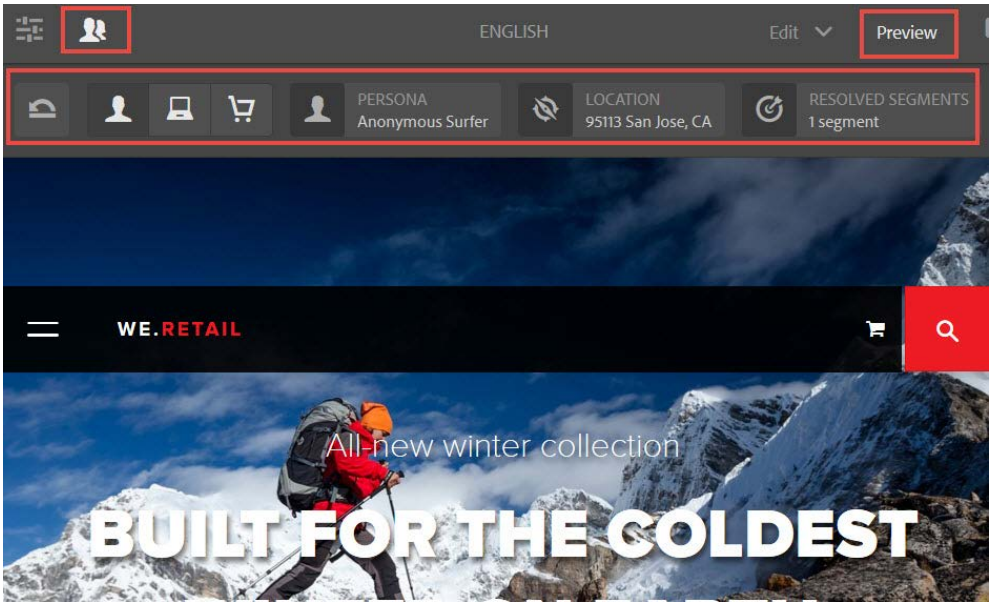
- **COMMERCE**—provides visitor's cart details and values such as promotions, vouchers, and so on.



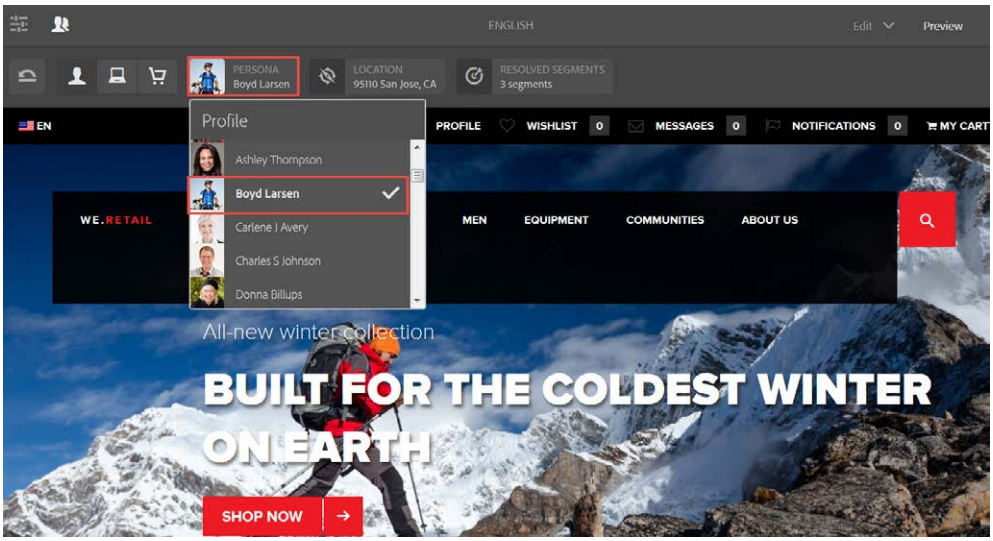
Exploring User Profiles in ContextHub

Let us see how to explore user profiles in ContextHub:

1. Navigate to **Sites** > **We.Retail** > **United States** > **English**, and then open the **Training_<username>** page in **Preview** mode.
2. Click **ContextHub** on the toolbar. ContextHub modes, **PERSONA**, **DEVICE**, and **COMMERCE** appear.

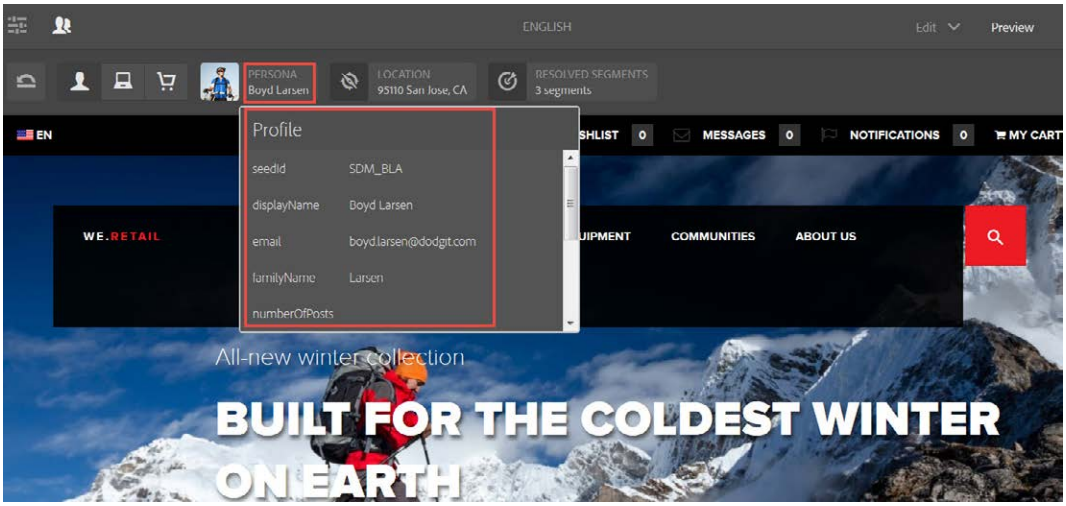


3. Click the **PERSONA** icon to display a list of profiles. Select **Boyd Larsen** from the drop-down.

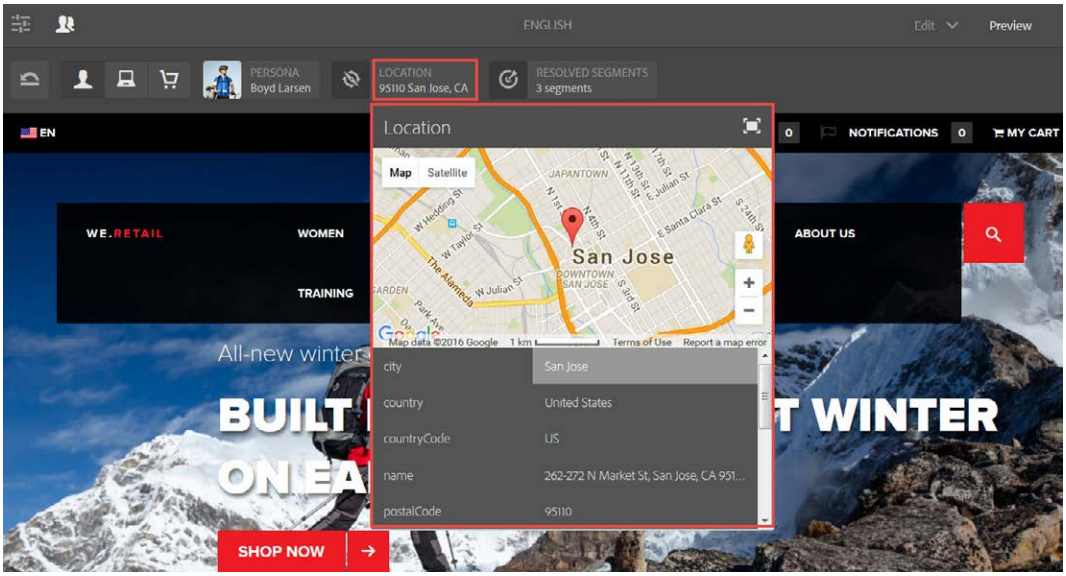


4. When you click:

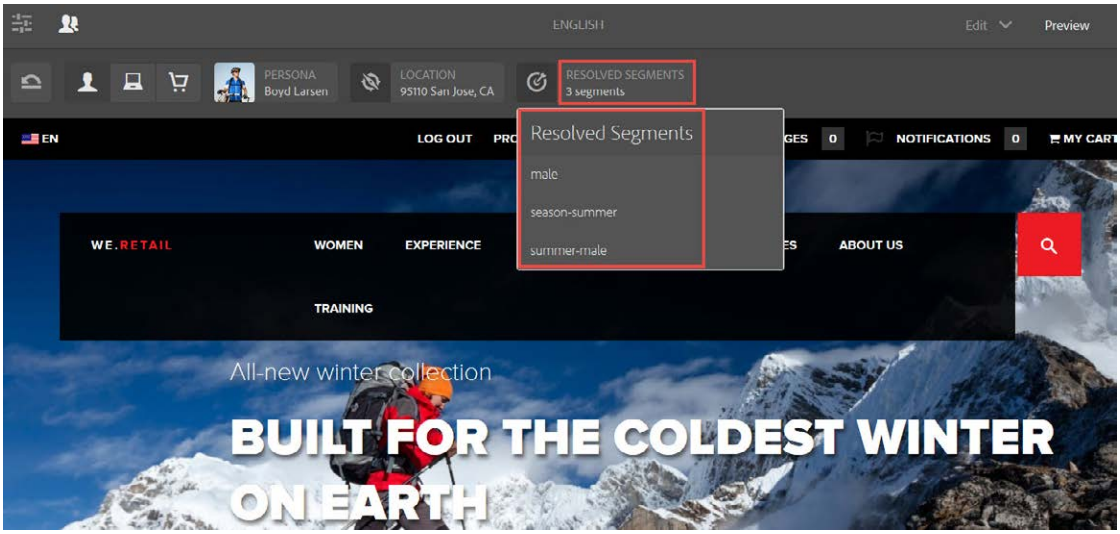
a. PERSONA, **Boyd Larsen** visitor details such as name, email, and gender are displayed.



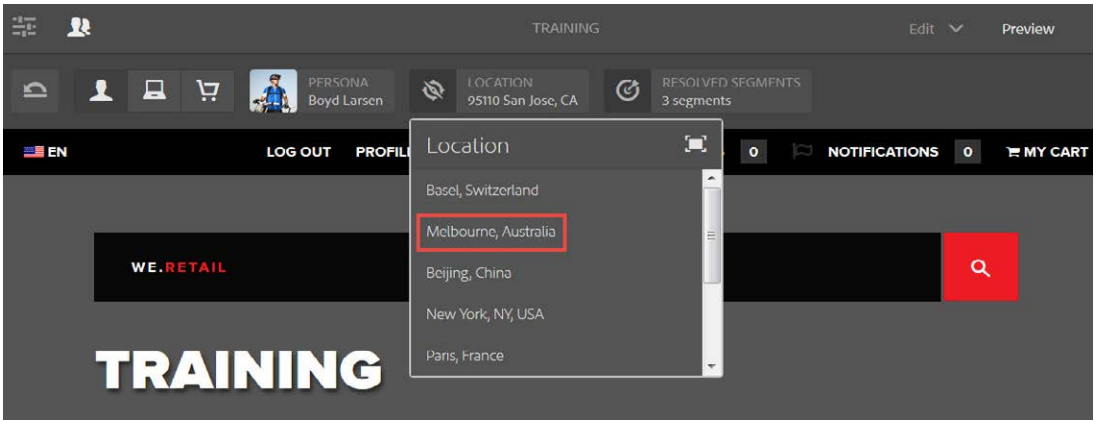
b. LOCATION, a map, latitude and longitude details of the visitor's location are displayed.

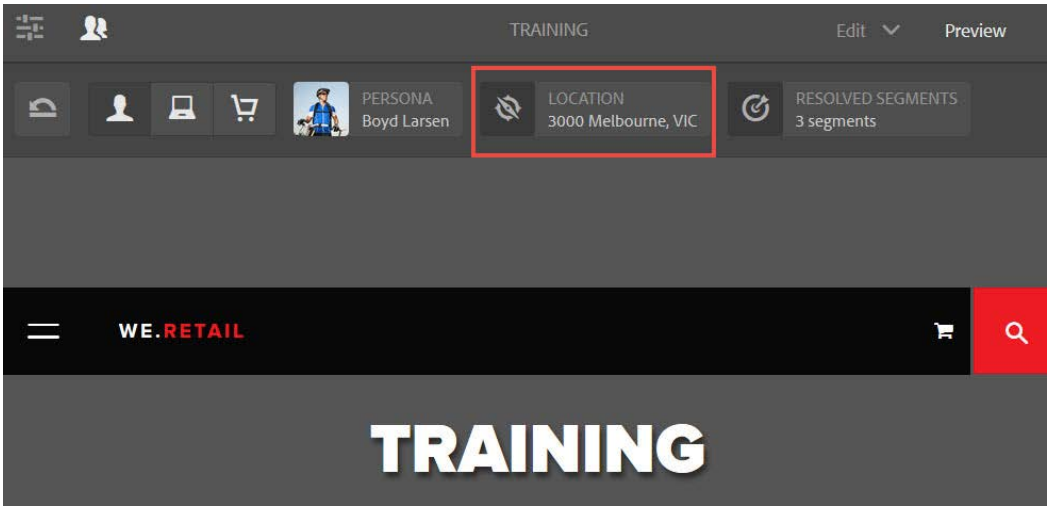


c. RESOLVED SEGMENTS, the list of segments that the visitor is mapped to display.



5. You can change the location of the visitor (for example, to Melbourne, Australia) using the **LOCATION** tab.





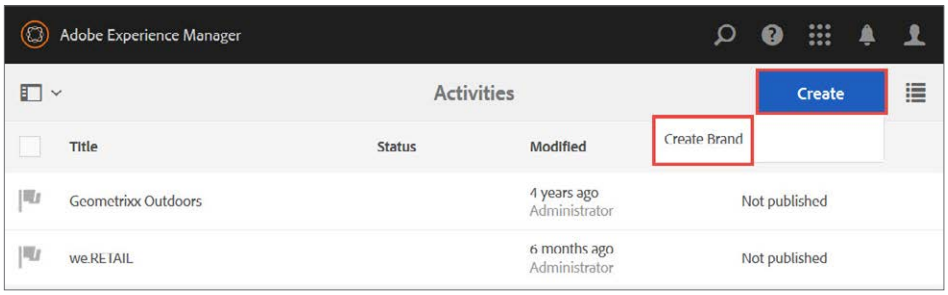
Personalization and Content Targeting in Adobe Experience Manager

The steps involved in personalization and content targeting process are:

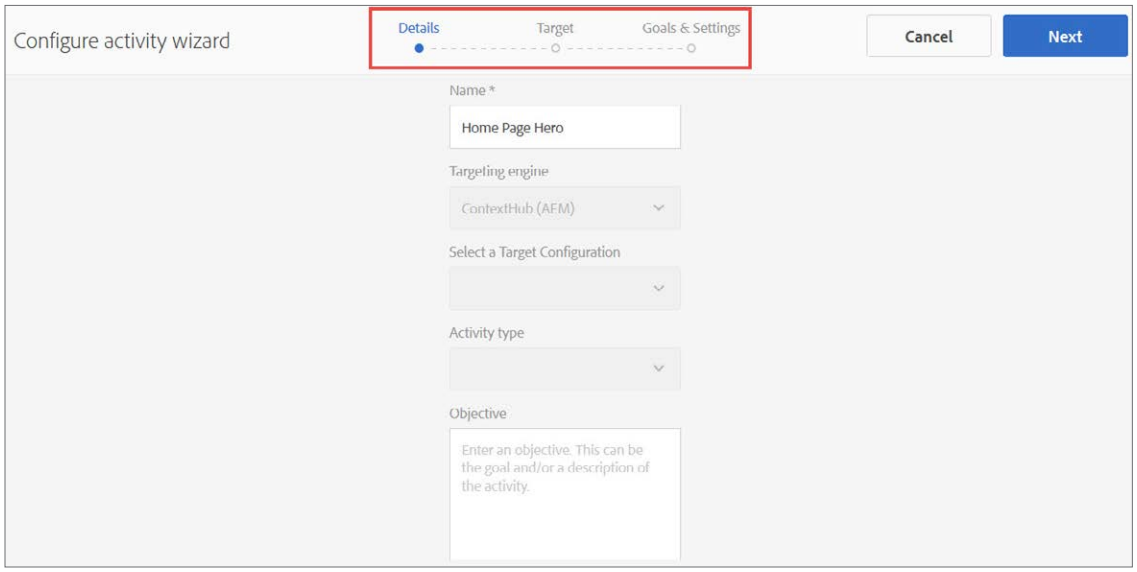
1. Create a brand
2. Create an activity for the brand
3. Create experiences
4. Map experiences with audiences
5. Target a component
6. Simulate experiences

In personalization and targeting process first you need to:

1. Create a brand



2. Create an activity for the brand has three sub-steps:
    a. **Details**: Select **ContextHub (AEM)** target engine add a **Name** to the activity
    b. **Target**: Add audience to the activity and create and map audiences with experiences
    c. **Goals & Settings**: For scheduling the Targeting process



Following the above steps, you will have a brand-related activity, and audiences are mapped with experiences, that can be used for content targeting.

Perform **Task 1: Create an activity and map audience with experiences to manage marketing efforts,** from the Lab Activity section.

Content Targeting

Content targeting allows you to:

- include components on your webpage that dynamically displays personalized content based on available visitor information.
- configure any component in the Adobe Experience Manager authoring system to display targeted content by assigning an activity to it.

You can author targeted content using the Targeting mode of the Touch UI.
For authoring targeted content:

1. Open the page, where you want to place the targeted content.
2. Add components to the page.
3. Select Targeting mode from the toolbar.
4. Select a brand and an activity to see the experiences.
5. Add offers to an experience by creating offers or using offers from a library.
6. Simulate user experiences using ContextHub.

Simulating an Experience

Once a page is created with targeted content, verify the mappings of audiences and their experiences. Let's simulate a visitor's experience to verify how an experience changes based on the visitor. When simulating, load different user profiles and see the targeted content for that user.

The criteria that determine the content when simulating a visitor experience are:

- Data in the user's session store (through ContextHub)
- Activities that are live
- Rules that define the segments
- Content of the experiences in the target components

Tools used for simulation are:

- Activity in Targeting mode: The page displays the offers for a user that is currently selected in ContextHub. You can edit the offers that target a user.
- Preview mode: Use ContextHub to select the users and locations that satisfy the criteria of the segments that your experiences are based on. When the ContextHub selections change, the targeted content changes accordingly.

 Perform **Task 2: Create and simulate the targeted content**, from the Lab Activity section.

Additional Information on Content Targeting

You can use also use Adobe Target for creating personalized and targeted content (you must have a valid Adobe Target account to use Adobe Target). If you are using Adobe Target, you must configure the integration first. See instructions for Integrating with Adobe Target.

If you use Adobe Target as the targeting engine all the steps described in Personalization and Content Targeting in Adobe Experience Manager section remain the same except the Targeting process.

Targeting Process of Adobe Target

Targeting mode enables you to configure several aspects of an activity. Use the following three-step process for creating targeted content for a brand activity:

- Create: Add or remove experiences, and add offers for each experience.
- Target: Specify the audience that each experience targets. You can target a specific audience and if using A/B testing decide what percentage of traffic goes to which experience.
- Goals & Settings: Schedule the activity and set the priority. You can also set success metric goals.

If you have integrated Adobe Experience Manager with Adobe Target or want to integrate in future, you can refer to Targeting in Adobe Target to understand the process in detail.

Landing Pages

The landing pages feature allows marketers to import the design and content right into an Adobe Experience Manager page. The web developers prepare the HTML and additional assets. You can import these assets into a page. This functionality helps to create marketing landing pages that are dynamic and easy to create.

Defining Landing Pages

Landing pages are single or multi-page sites that lead to the "endpoint" of a marketing outreach. A landing page can serve various purposes, but all have one thing in common—the visitor should fulfill a task, which is what defines the success of a landing page.

Landing pages let you narrow your focus and remove the clutter from your pages that could distract your visitor from taking the action you want them to take. It allows you greater control to direct them and helps them find what they're looking for much faster—and this, in turn, can ripple out to affect your search engine rankings too.

Types of Landing Pages

Landing pages can be:

- Desktop landing pages—used in desktops.
- Mobile landing pages—which is the mobile version of the landing page. It is the child page of the desktop landing page.

NOTE: If the desktop landing page is deleted or deactivated, the mobile landing page is also deleted or deactivated.

Benefits of Using Landing Pages

The benefits of using landing pages are:

- Provides a better user experience because you are providing personalized content
- Increases conversion rates as visitors are able to precisely find what they are looking for
- Provides more opportunities for visitors to relate to your content

Components of Landing Pages

To make parts of the landing page editable within Adobe Experience Manager, the content of landing page HTML must be mapped with the Adobe Experience Manager components directly.

The design importer understands the following components of the page:

- Text—any form of text
- Title—content in the form of headings
- Image—images included in the page
- Call to Action
  - Click through Link
  - Graphical Link
- Lead Form—used to capture user information
- Paragraph System (parsys)—used to add content to the existing components, and to add new components

In addition to the above components, you can also add custom components to the landing page.

Call to Action Component

A landing page design can have several links in the form of text, image, buttons, and so on.

Call to Action (CTA) is an image or text that prompts visitors to take action. It is literally, a "call" to take an "action". For example, "Download an eBook", "Sign up for a webinar", "Get a coupon", or "Attend an event".

Click through links and Graphical links are CTA components and have similar options. A Click through Link has additional rich text options.

Click through Link

A Click through link is a text link with a target URL. This component is used to take the visitor to the target URL specified in the component properties.

Graphical Link

A Graphical link is an image that, when clicked, takes the visitor to a target URL. The image can be a simple button or any graphical image used as a background. When the image is clicked, the user is taken to the target URL specified in the component properties.

Creating Landing Pages

The landing pages feature in Adobe Experience Manager allows marketers to work with web designers at agencies or internal creative teams to create page designs that can be imported into Adobe Experience Manager. and the landing pages can still be edited by the marketers and published under the same governance as the rest of the Adobe Experience Manager-powered sites.
To create a landing page, follow these steps:

1. Create an importer page.
2. Prepare HTML for importing (this step is performed by a web developer).
3. Import the design package.

Creating an Importer Page

Before you import the Landing Page design, you must create an importer page (for example, under an activity). The importer page template lets you import your full HTML landing page. The page contains a drop box where you can import the landing page design package using drag and drop functionality.

Importing a Design Package

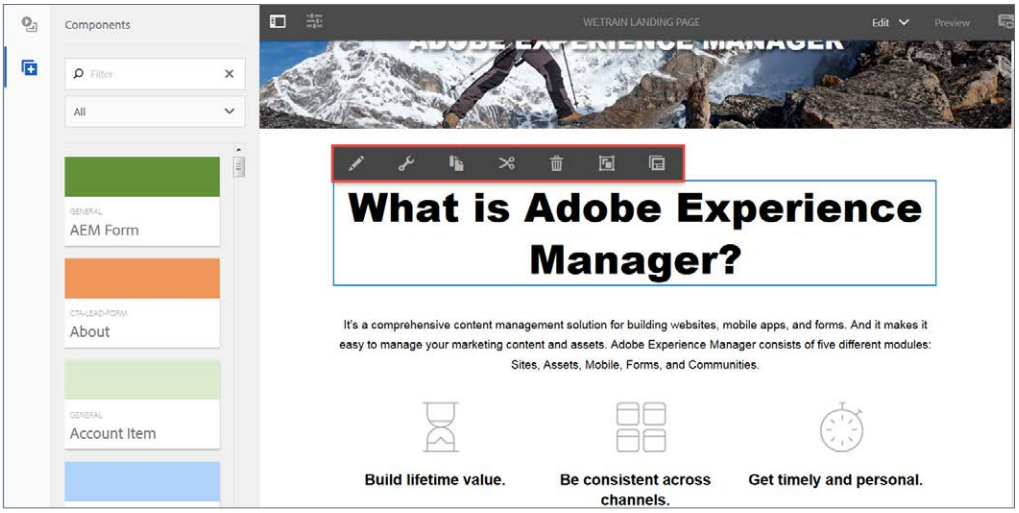
After creating a blank landing page, you can import a design package onto it. Let's see how this works by performing a task.

Perform **Task 3: Create an importer page and import the page design package**, from the Lab Activity section.

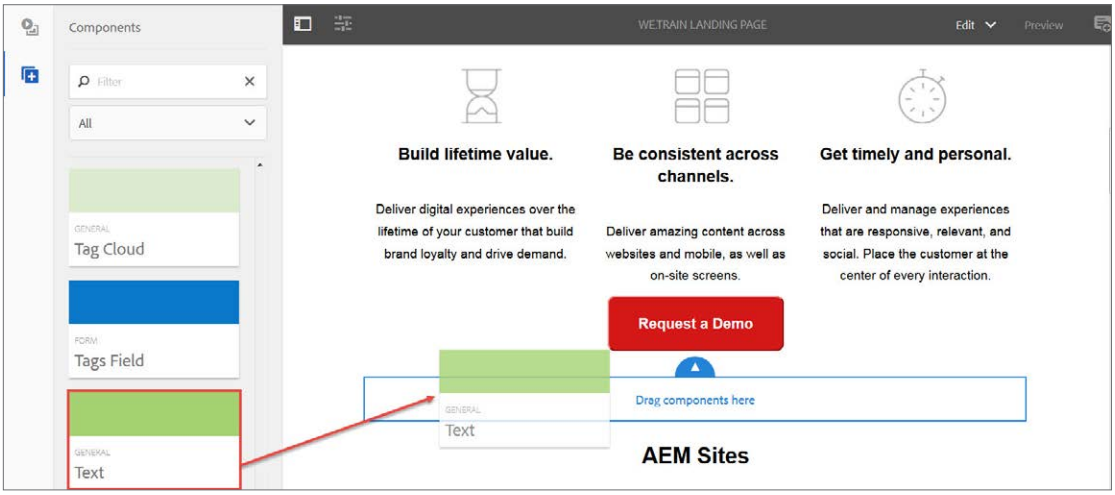
Landing Page Actions

You can perform different actions on landing pages, with a few of those actions listed here:

- Modify and add components to the landing page:
  - › Double-click the existing component to open and edit as you edit any other component.



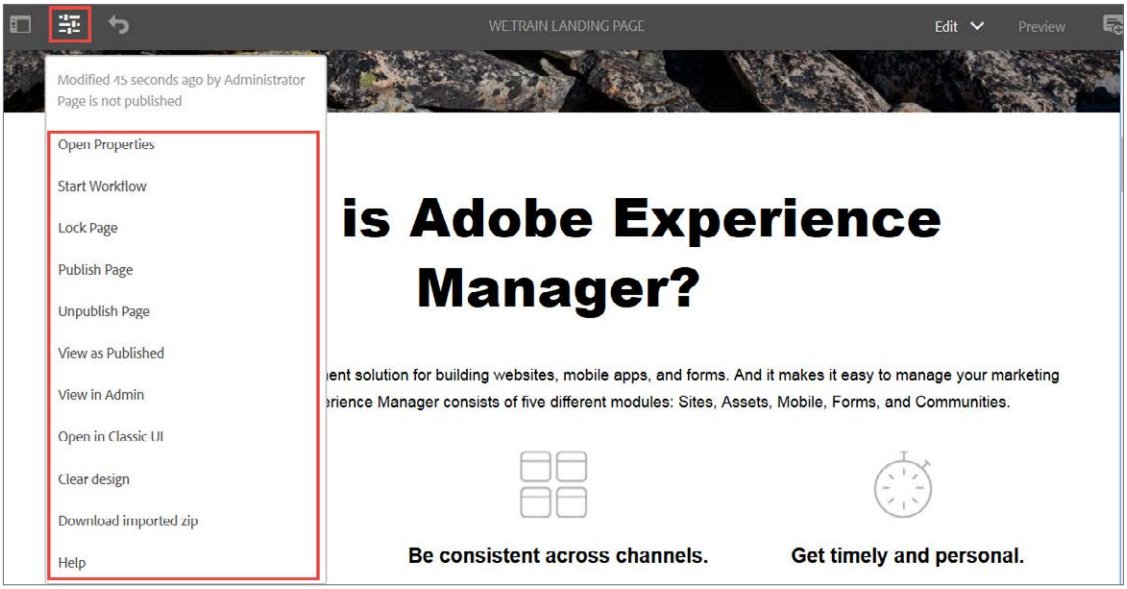
  - › Drag and drop components from the Side Panel to the landing page.



Additional Actions

Similar to pages you can perform the following actions on landing pages, except **Clear design** and **Download imported zip**.

- **Clear design**: In case you want to re-import your landing page design package after making some changes to it, you can "clear" the landing page by clicking Clear design deletes the imported landing page and creates a blank landing page.
- **Download imported zip**: Lets you record which zip was imported with a particular landing page. Note that changes made on a page are not added to the zip.



Lab Activity

Scenario

XYZ is an organization in the education sector. They are keen on expanding their digital marketing opportunity to target global customers. They are looking for a solution that segments customer/ audience information based on their purchasing pattern—across web and mobile channels.

Challenge

- Creating a unique brand experience that maps to different audience segments
- Testing the brand experience by simulating different visitor profile
- Creating landing pages for better user experience and increased customer conversion

Overview

To complete the challenge, you will need to:

- Create an activity to create personalized content for the visitors
- Map experiences with audiences for better targeting
- Simulate experiences based on visitor profiles to test the targeted content
- Create landing pages by importing the design package directly onto the page

Pre-requisites

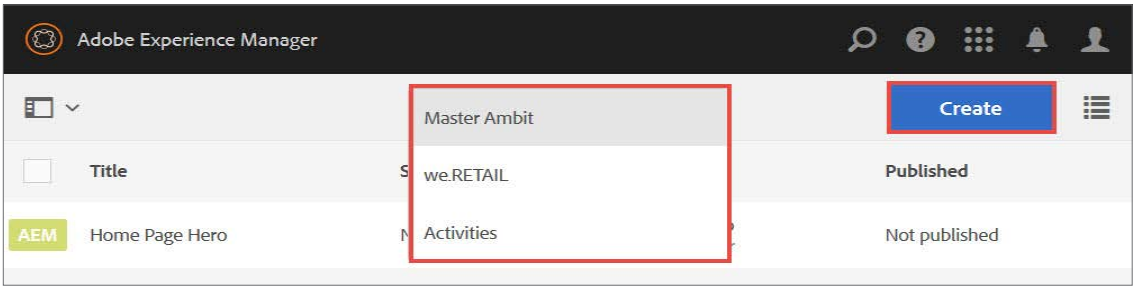
- You should have performed the Adobe Experience Manager installation steps, and have a running Author instance and Publish instance.
- USB content with **We.Train Landing Page** design package.

Steps

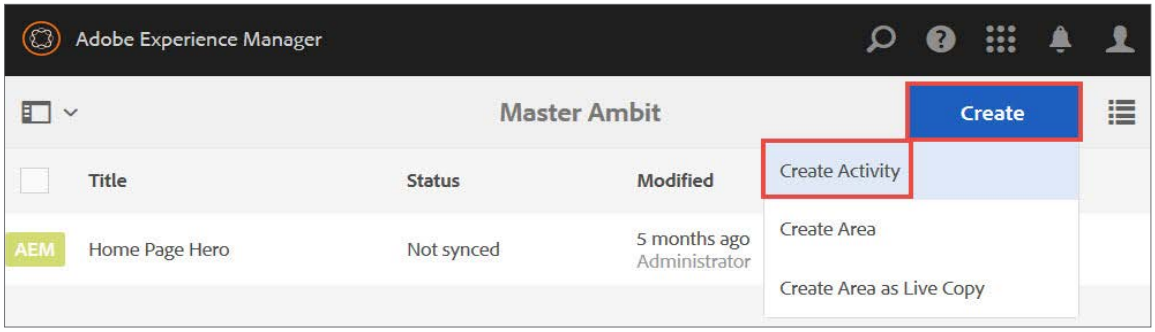
Task 1: Create an activity and map audience with experiences to manage marketing efforts

Let's define and organize marketing efforts:

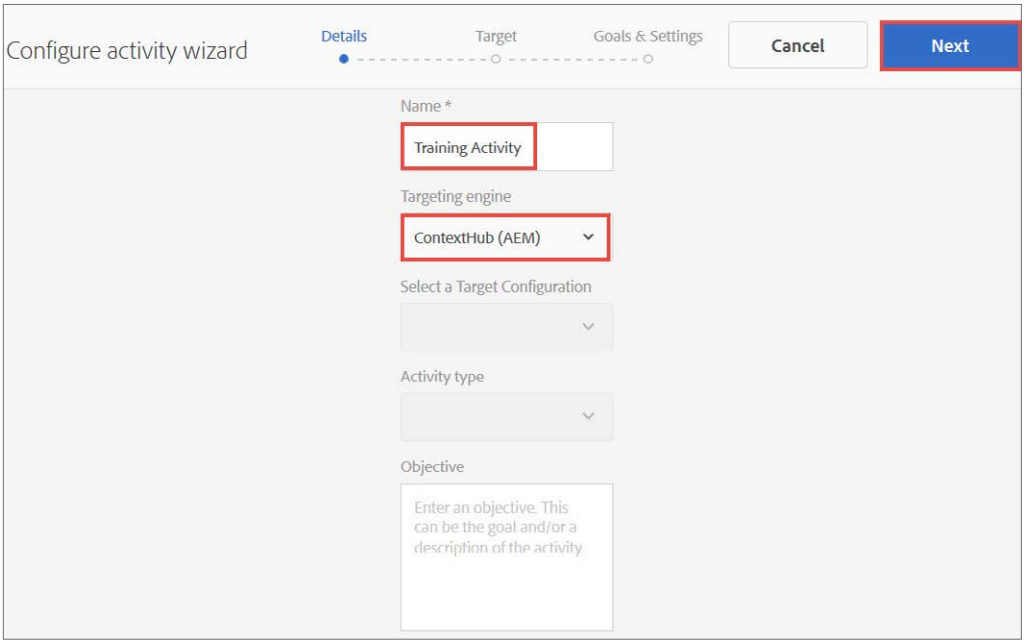
1. From the **Personalization** console (**Navigation** > **Personalization**), navigate to **Activities** > **we.RETAIL** > **Master Amit**.



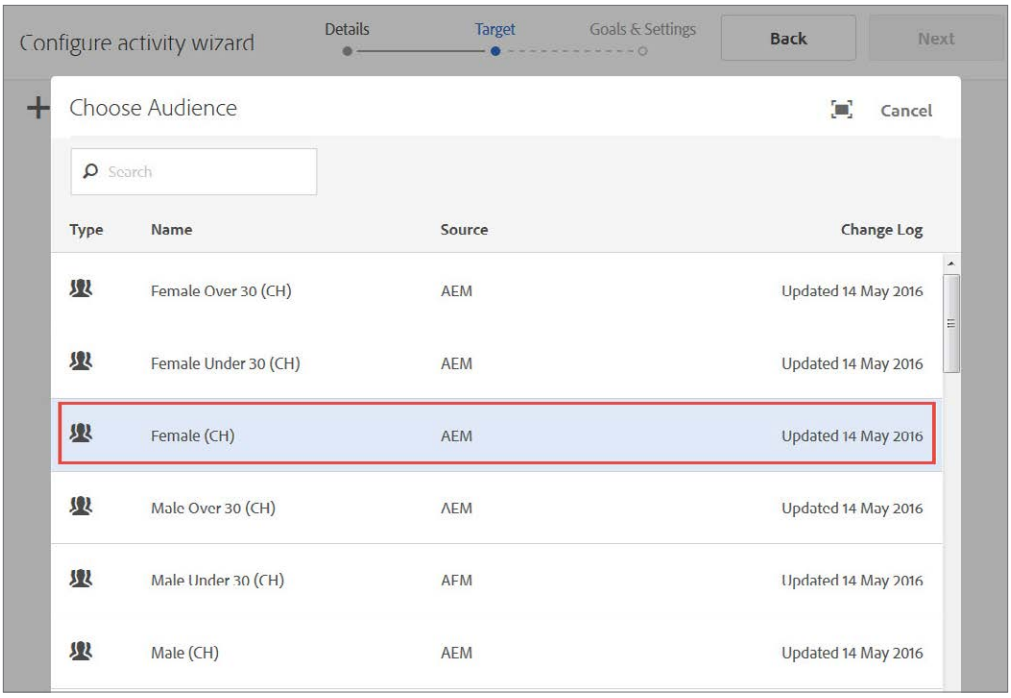
2. Click **Create** > **Create Activity** from the actions bar.



3. In **Configure activity wizard** has three tasks; **Details**, **Target**, and **Goal & Settings**.
4. In **Details** add the following:
   a. Name: **Training Activity_<username>**
   b. Targeting engine: Select **ContextHub (AEM)** from the drop-down
5. Click **Next**.

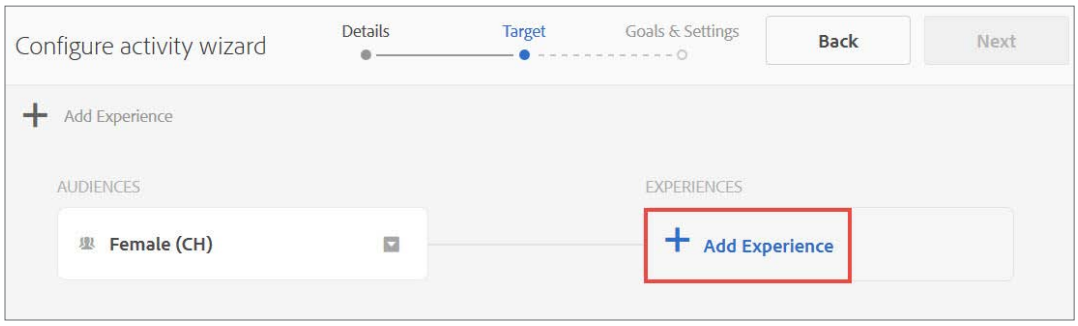


6. In **Target** perform the following:
   a. Click **Add Experience**, select an existing audience for example, **Female (CH)** from **Choose Audience** dialog box.

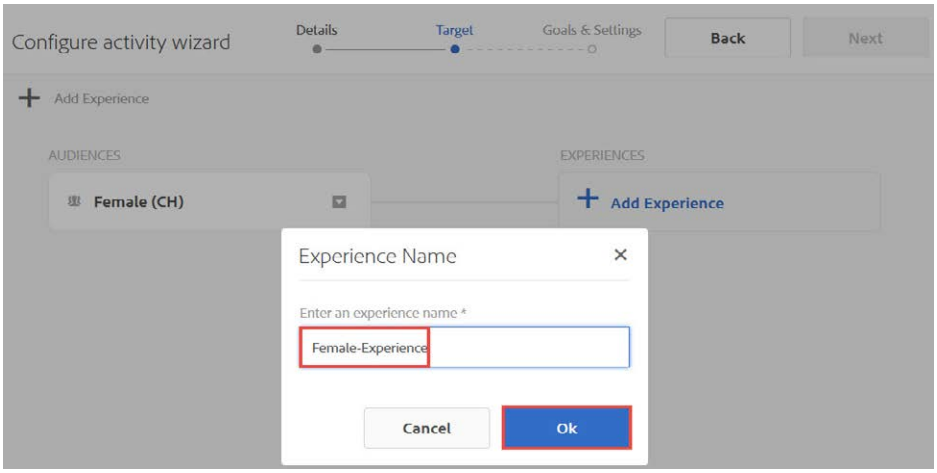


For this audience you have to map it with an experience.

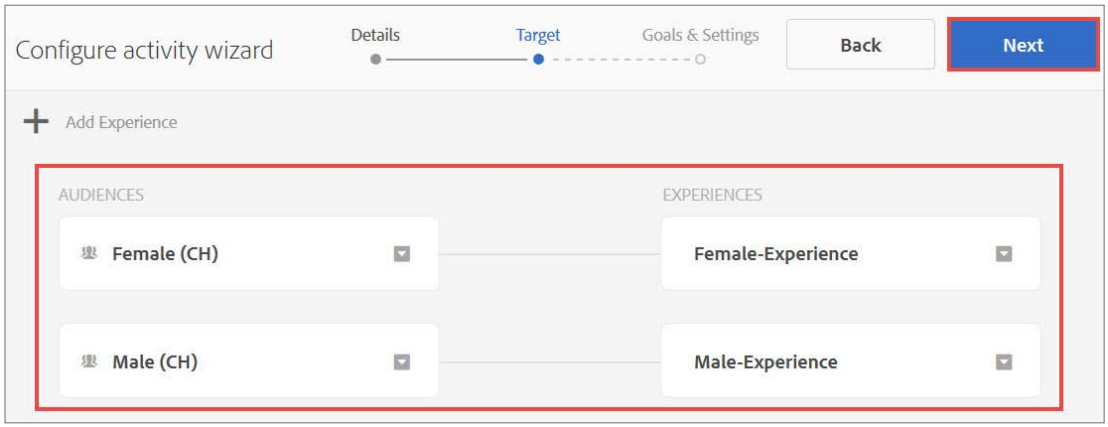
b. Click **Add Experience** beside **Female (CH)** to map the audience with experience. **Experience Name** dialog box appears.



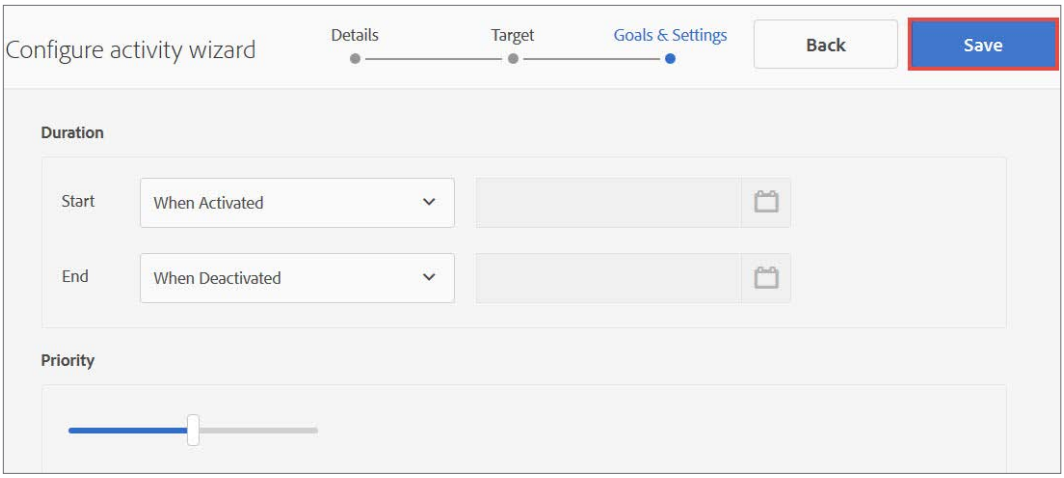
c. Enter name for example, **Female-Experience** in **Enter an experience name** field, and then click **OK**.



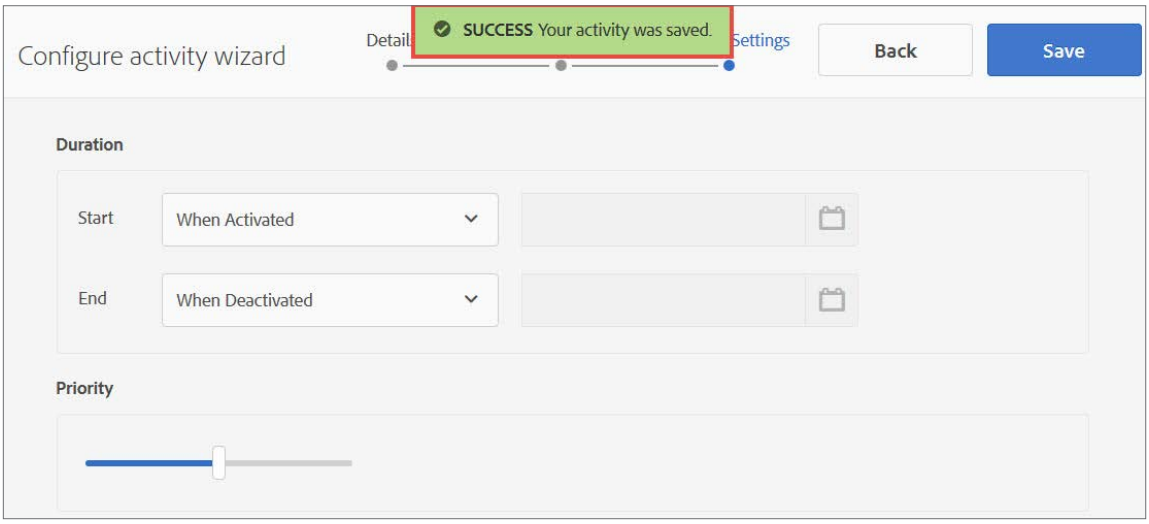
7. Follow Steps 6a-c to add **Male (CH)** audience, and to map it with **Male-Experience**, and then **Next**.



8. In **Goals & Settings** you configure Targeting to work during a specific time period. Do not make any changes, and then click **Save**.



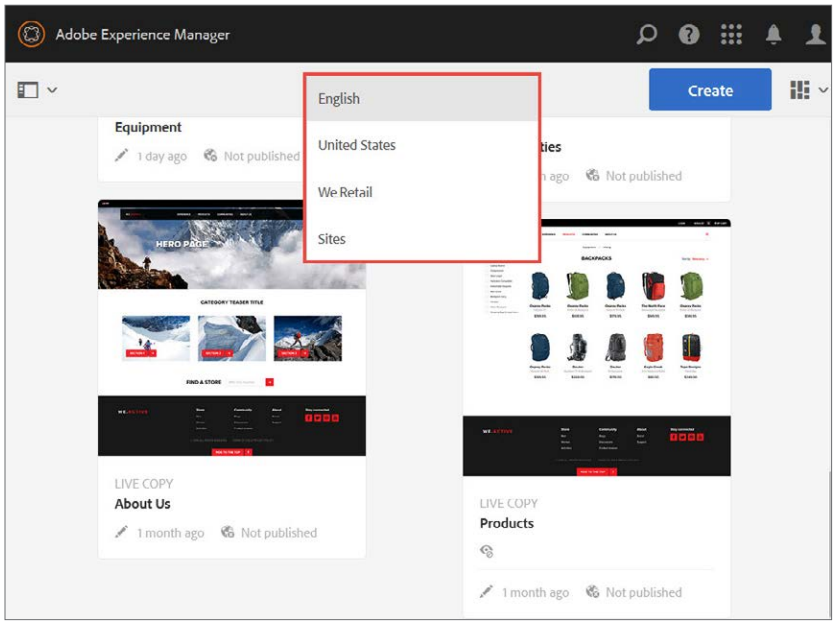
Your activity was saved message appears.



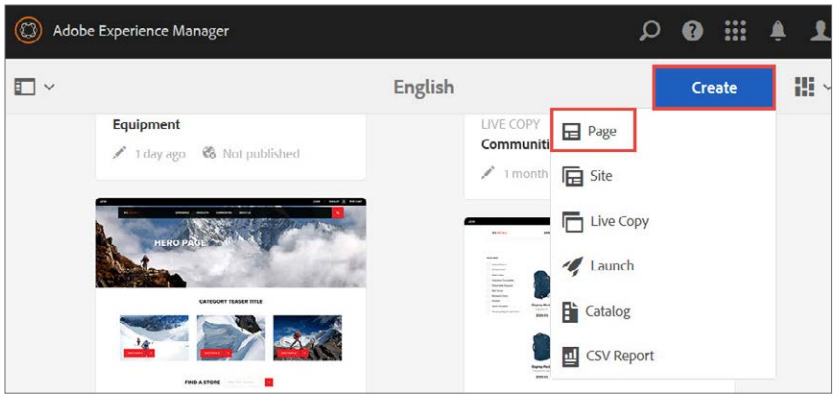
Task 2: Create and simulate targeted content

To create a targeted component in a page:

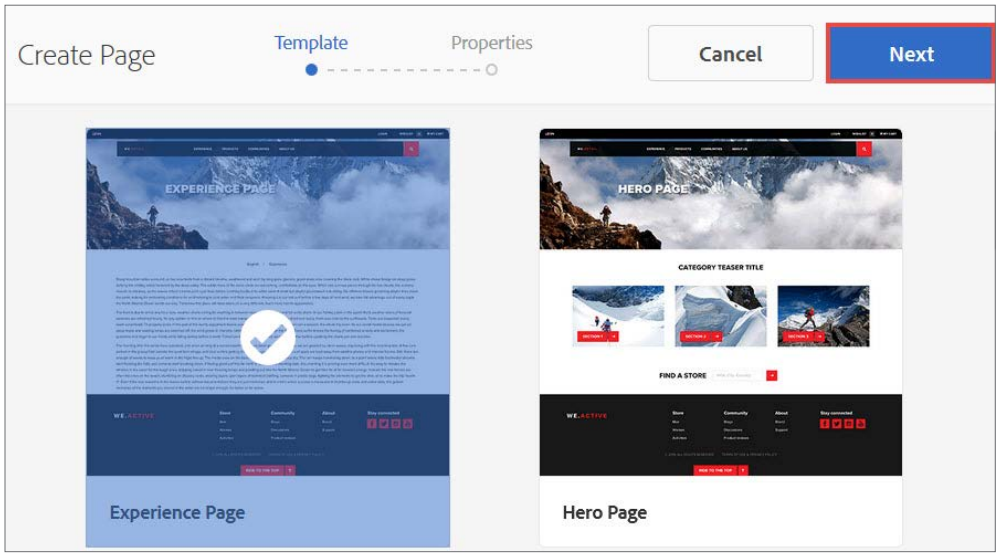
1. Navigate to **Sites** > **We.Retail** > **United States** > **English**.



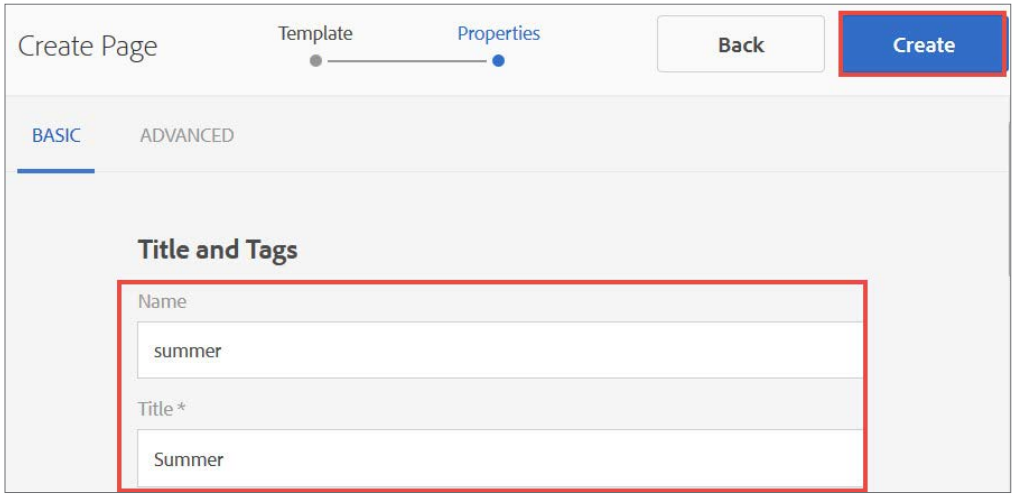
2. Click **Create** > **Page** from actions bar.



3. Select the **Experience Page** template, and then click **Next**.

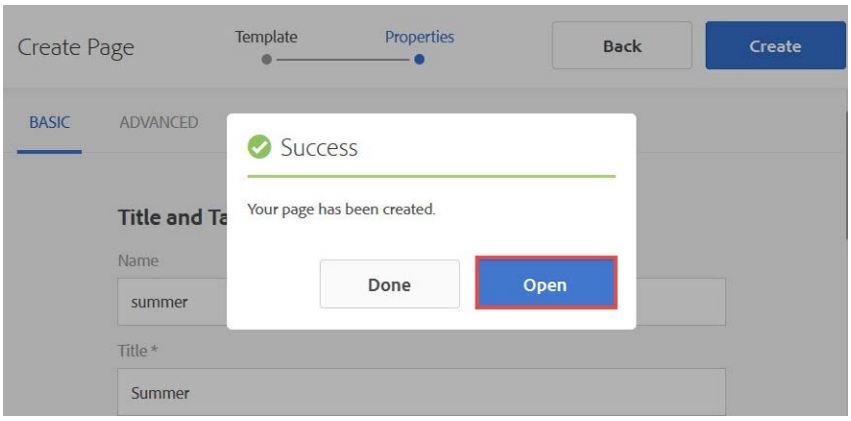


4. Provide **Name** as **summer**, Title as **Summer_<username>**, and then click **Create**.



NOTE: It is important to create your page with a unique identifier so that you can identify your page amongst those of your peers' unique identifiers being created in the same environment.

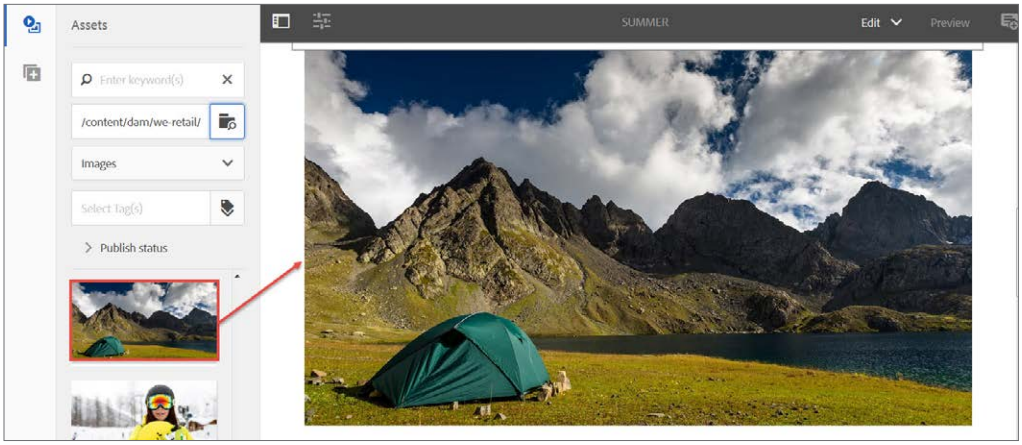
5. Click **Open** from **Success** dialog to open **Summer_<username>** page. Ensure the page is in **Edit** mode.



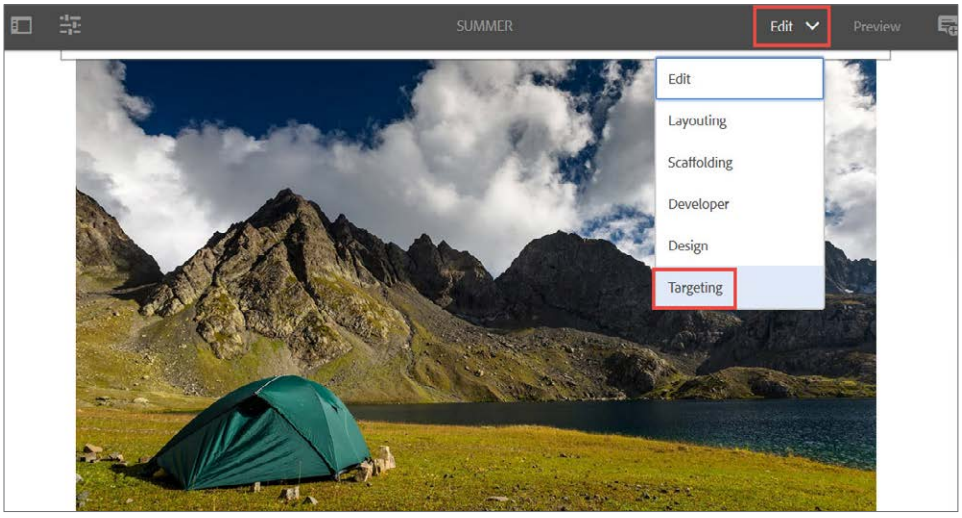
6. Click the **Side Panel** on the top-left of the screen.

7. Drag and drop any image from the **Assets** browser onto the **Drag component here** area.

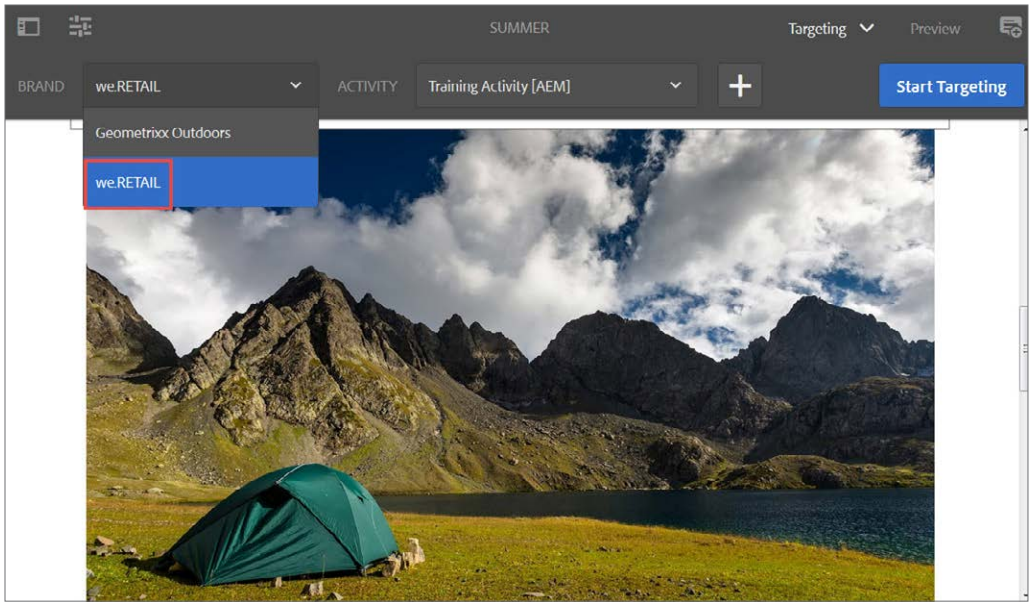
This will be the default content for the component (if there are no targeted activities on the component).



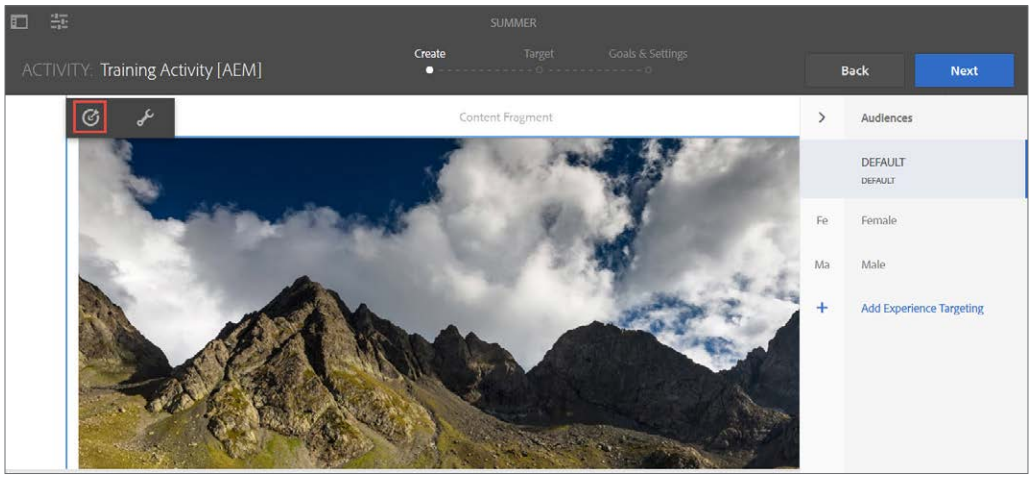
8. Select **Targeting** mode from the **Edit** drop-down.



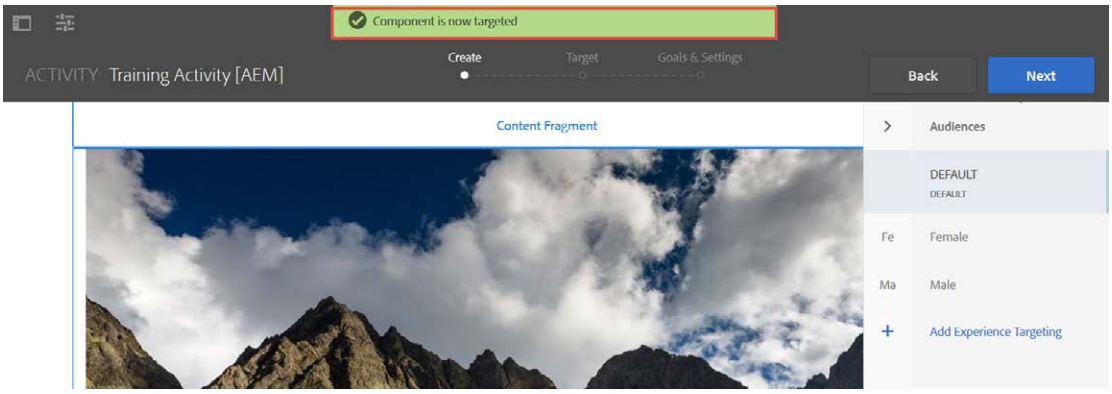
9. Select the **we.RETAIL** Brand and the **Training Activity,_<username>** from their respective drop-downs, and then click **Start Targeting**.



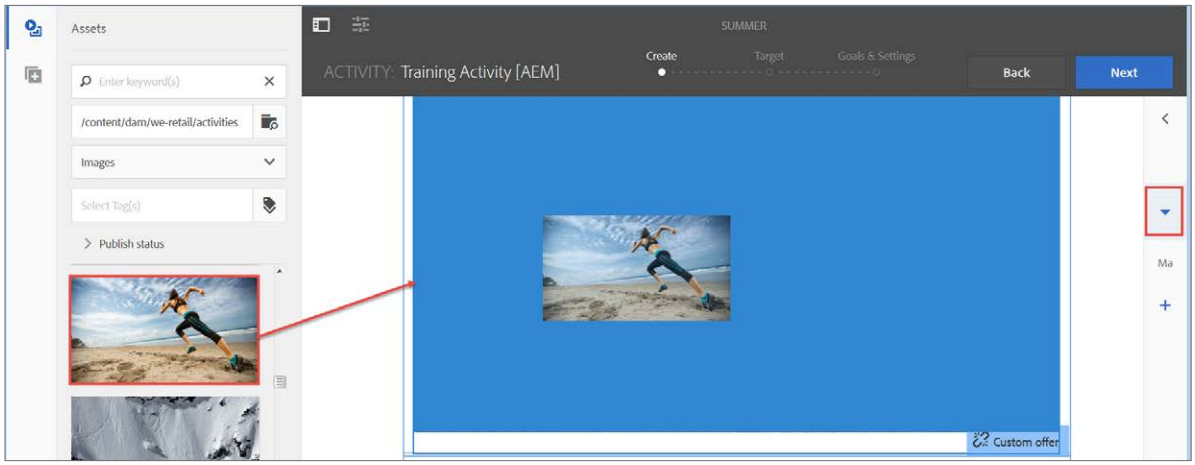
10. Select the image you added, and click **Target** from the component toolbar to make it a targeted component.



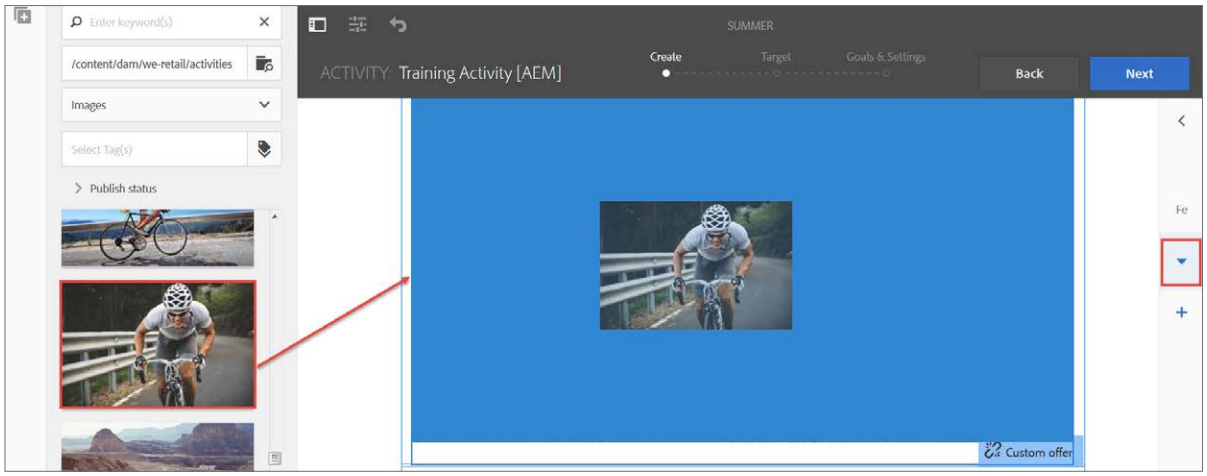
The **Component is now targeted** message appears.



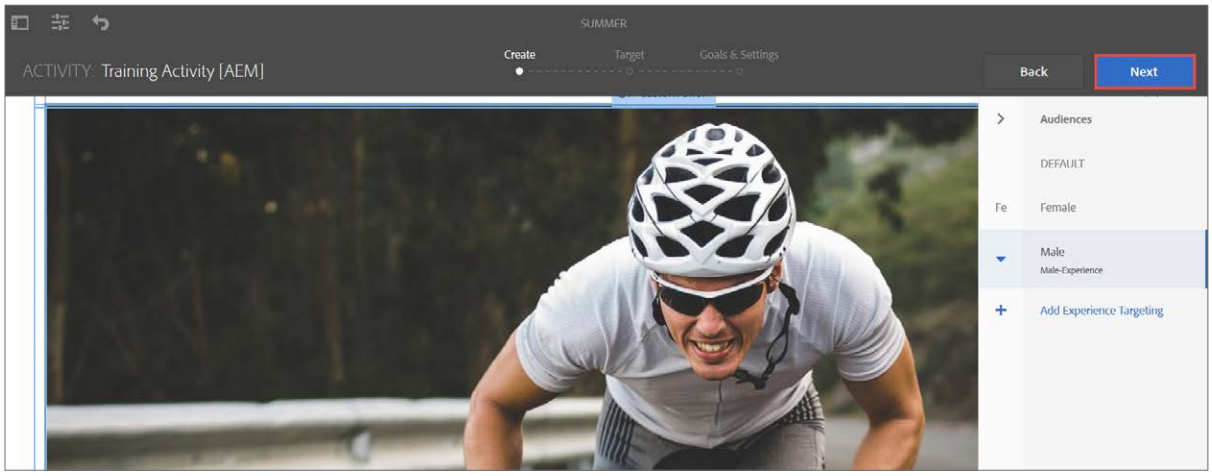
11. Click **Female-Experience** from the **Audiences** panel towards right side of the page, and then select the targeted image that is targeting enabled.
12. Drag and drop a new image into the targeted component, as shown in the following image.



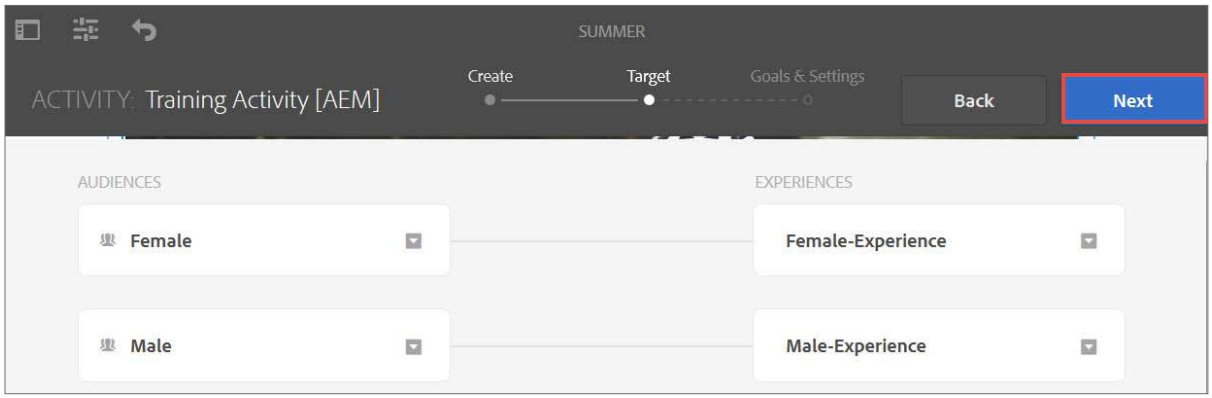
13. Click **Male-Experience** from the targeting menu, and the select the targeted image component.
14. Drag and drop a new image into the targeted component, as shown in the following image.



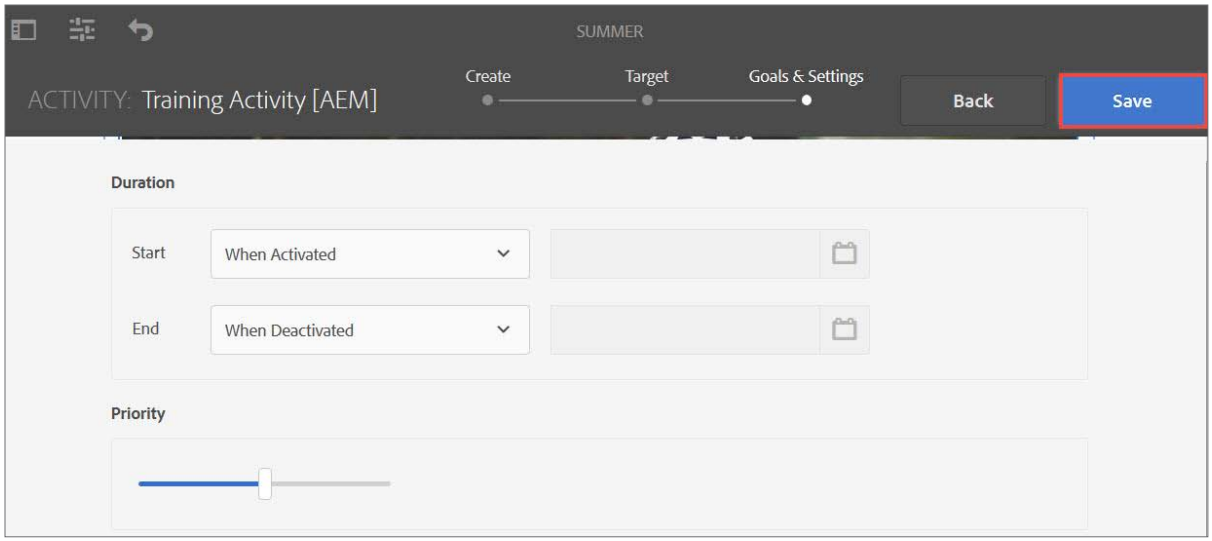
15. Click **Next** to complete **Create** process and move to **Target**.



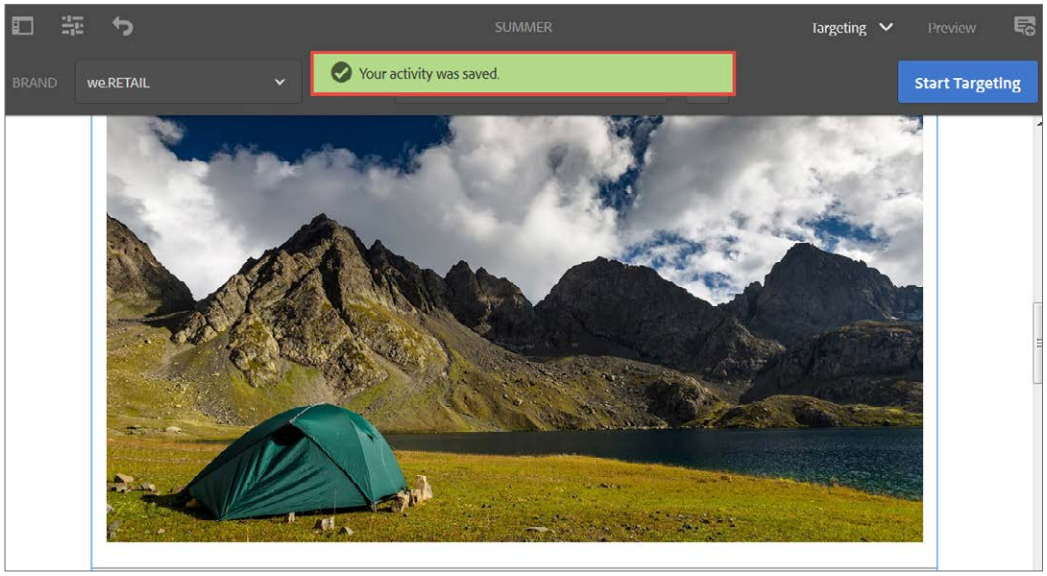
16. In **Target**, you can verify the segments that correlate with the experiences, and then click **Next**.



17. You can configure Targeting to work during a specific time period through **Goals & Settings**, and then click **Save** to save the targeting configurations.



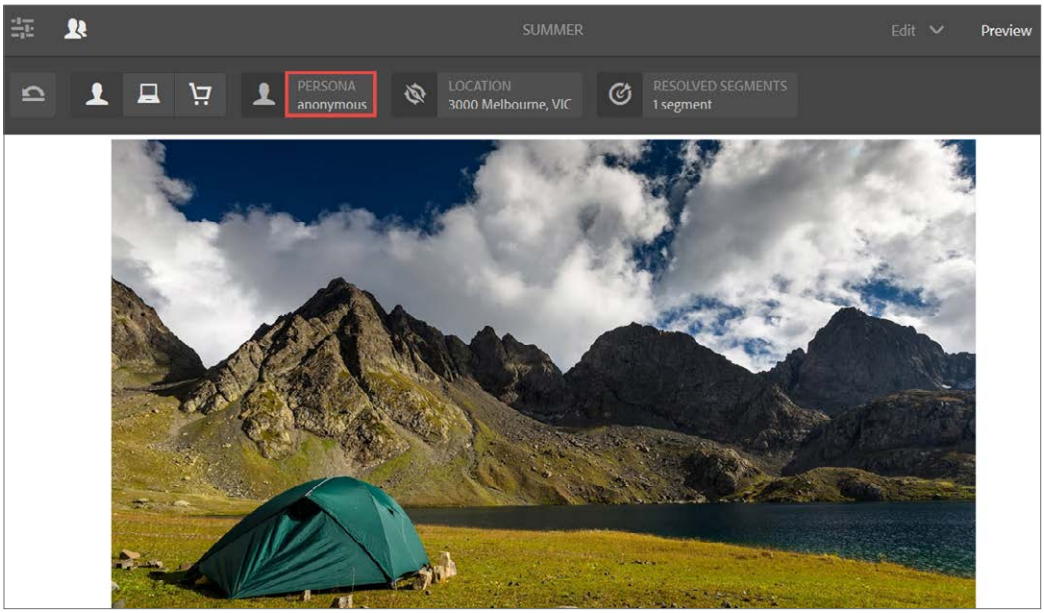
Your activity was saved message appears once the process is completed.



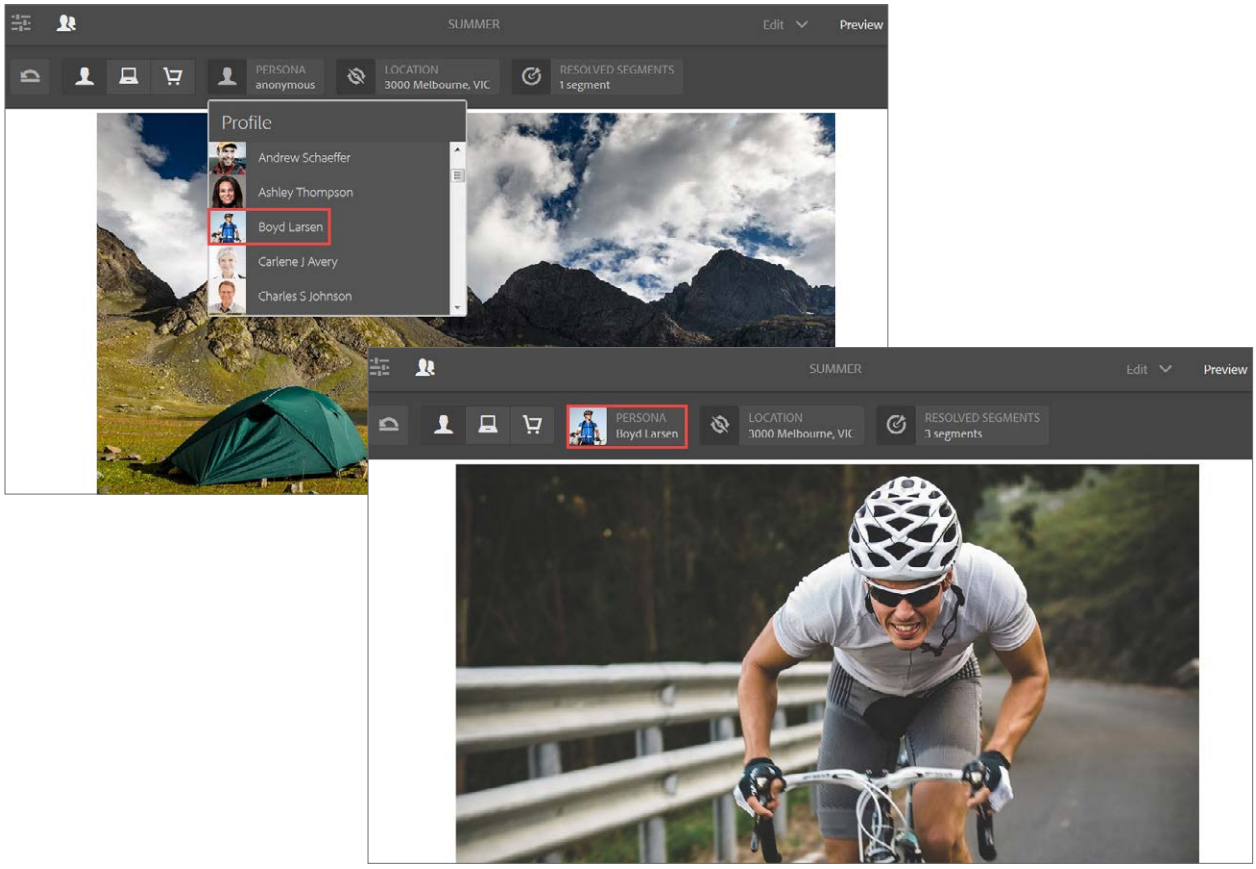
You created a page with targeted content and verified the mappings between audiences and their experiences.
Let's simulate a visitor's experience to verify how an experience changes based on the visitor.

To simulate an experience:

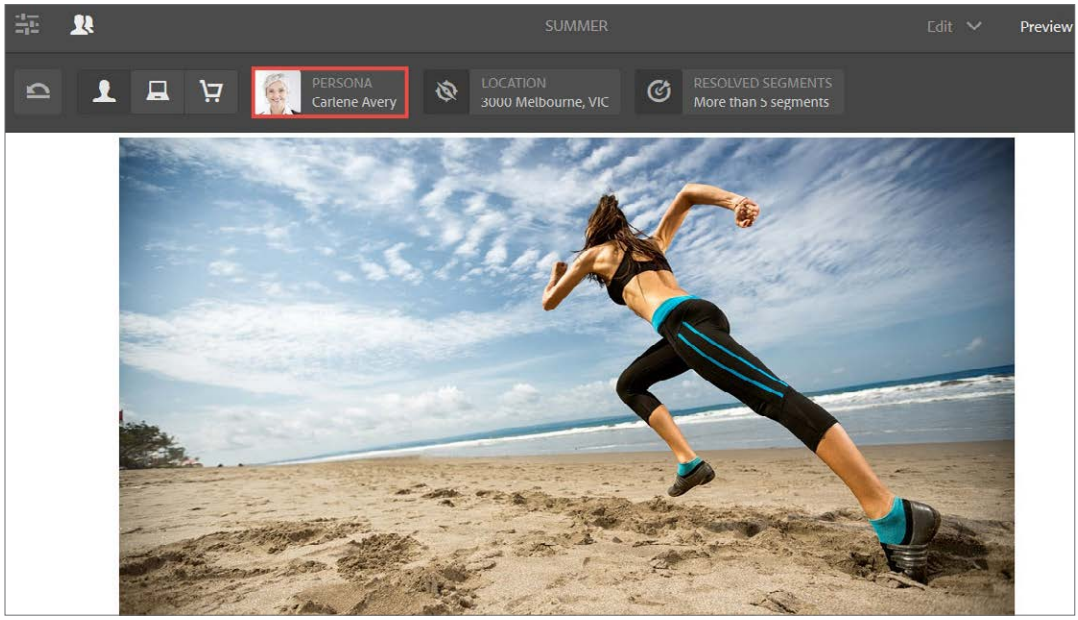
18. Open **Summer_<username>** with targeted content in **Preview** mode.
19. Click the **ContextHub** icon from the toolbar; all the modes of ContextHub appear.
20. Click **PERSONA** and choose **anonymous**. The image that was added to the **Default** audience appears.



21. Now, choose a male profile (for example, **Boyd Larsen**), and see how the default image changes to display the image associated with the Male (CH) audience.



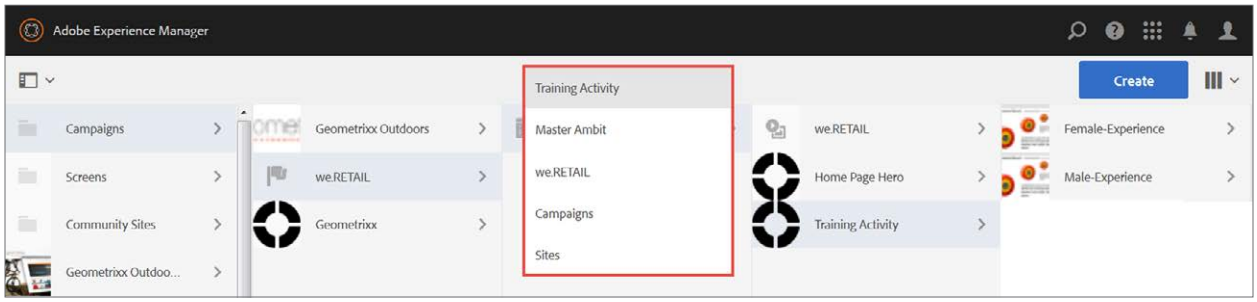
22. Choose a female profile (for example, **Carlene J Avery**), and see how the default image changes to display the image associated with the Female (CH) audience.



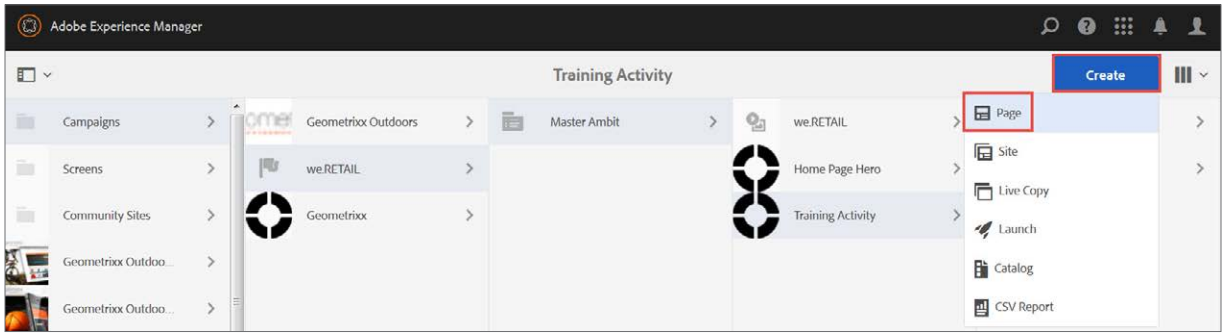
Task 3: Create an importer page and import the page design package

To create an importer page:

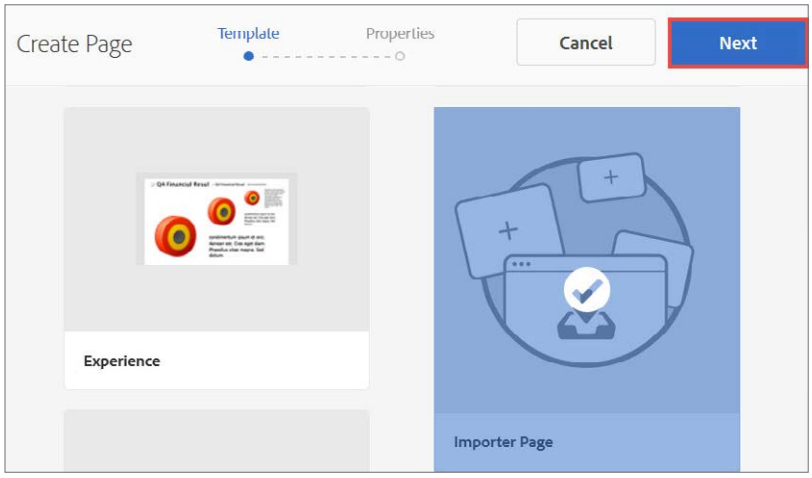
1. Navigate to **Training Activity_<username>** using the **Column View**. Follow the path highlighted in the image:



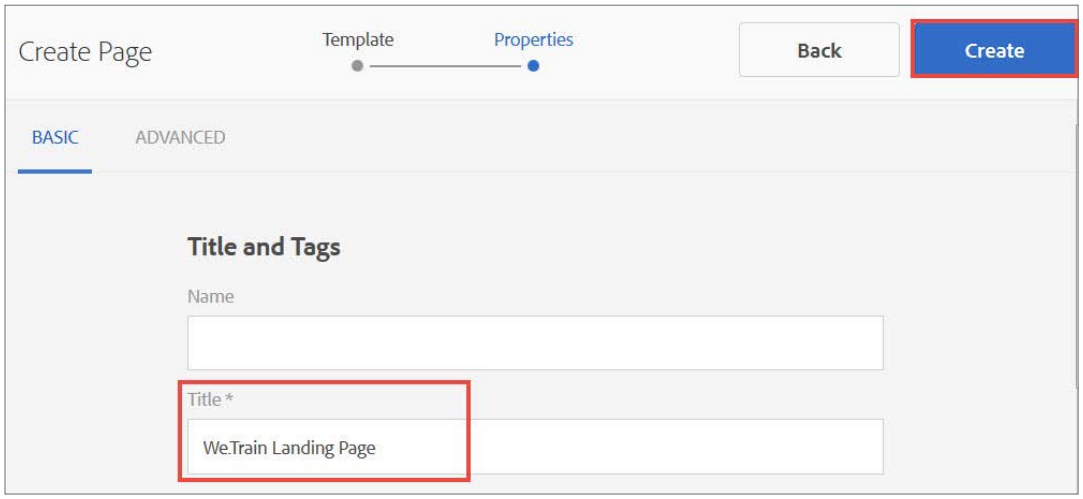
2. Click **Create** > **Page** from the actions bar.



3. Choose the **Importer Page** template and click **Next**.

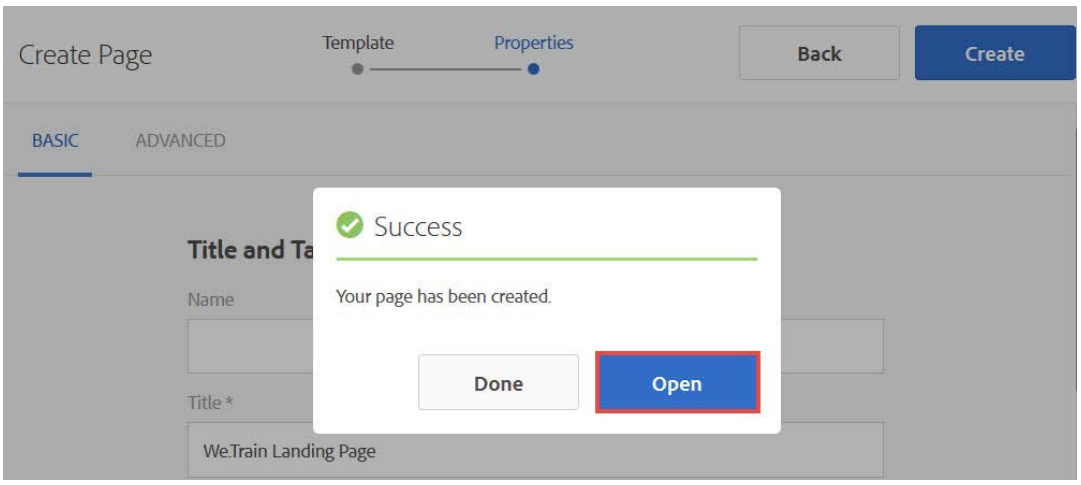


4. Provide **We.Train Landing Page**_<username> as the **Title** and click **Create**.



NOTE: It is important to create your landing page with a unique identifier so that you can identify your landing page amongst those of your peers' landing pages being created in the same environment.

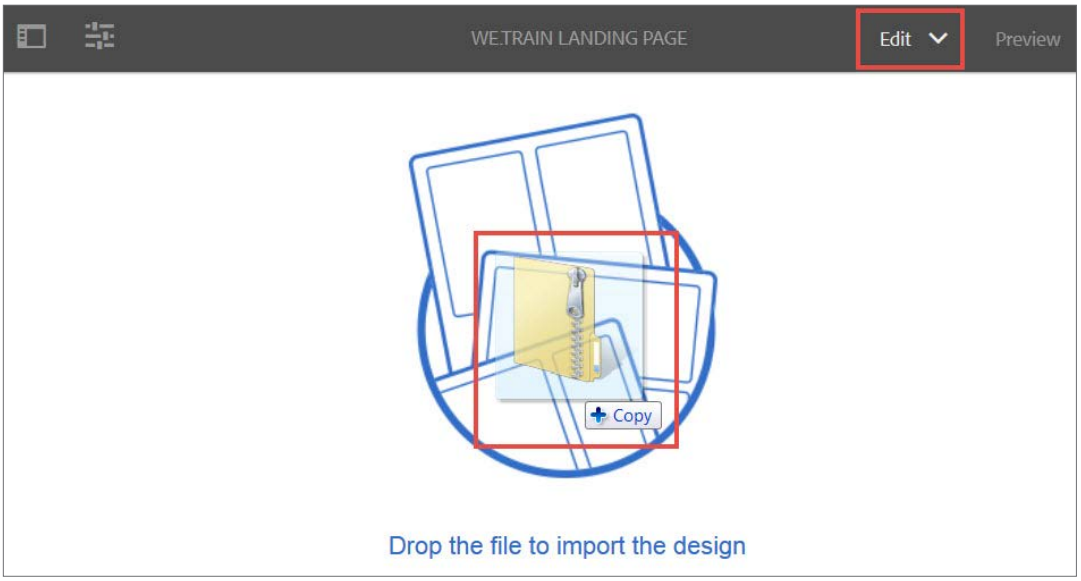
5. Click **Open** to add design to the **We.Train Landing Page_<username>**.



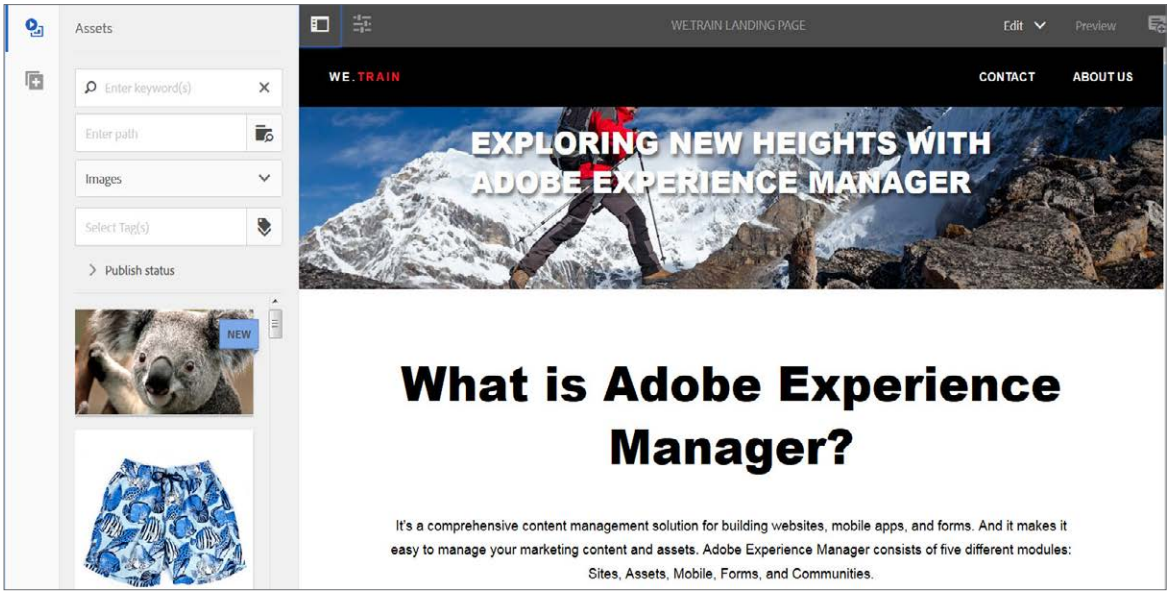
The design package (**We.Train Landing Page**) is available in the USB folder.

Follow these steps to import the design package onto a blank landing page.

6. Open the **We.Train Landing Page_<username>** that you created earlier in **Edit** mode.
7. Drag and drop the design package from the USB content folder onto the **Drop asset here** area.



The design is imported to the blank page. Your HTML landing page is ready to be used for the activity.



Scenario Conclusion

By performing these tasks, you:

- Creating a unique brand experience that maps to different audience segments
- Testing the brand experience by simulating different visitor profile
- Creating landing pages for better user experience and increased customer conversion

Summary

You should now be able to:

- Define personalization and its use in today's world
- Define the tools used for personalization and content targeting
- Define ContextHub
- Explore user profiles in ContextHub
- Explain the process to create activities
- Map audiences with experiences
- Simulate an experience using ContextHub
- Define landing pages
- Create and import landing pages
- Perform various actions of landing pages

Installation

Overview

This chapter provides information and step-by-step instructions to install Adobe Experience Manager. Depending on the type of user, there are multiple ways to install and run Adobe Experience Manager instance.

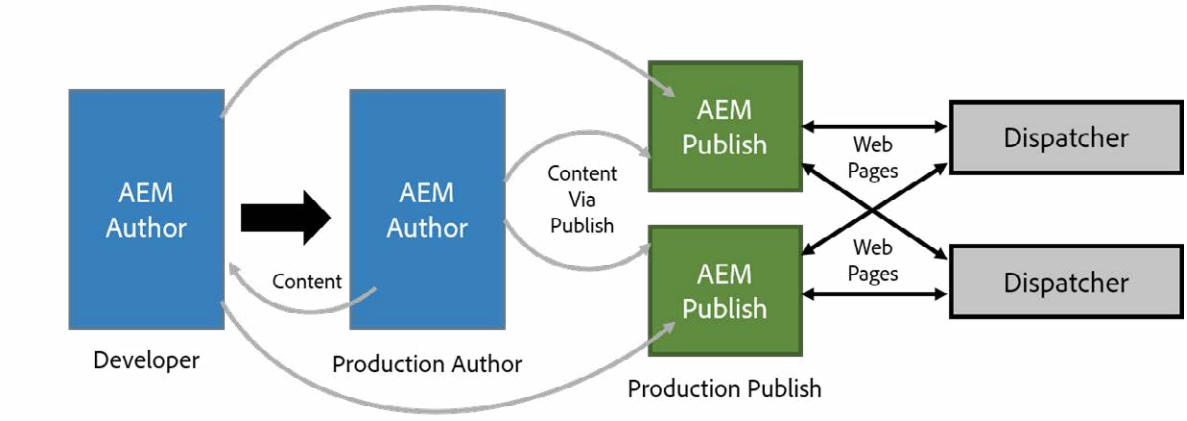
Objectives

By the end of this chapter, you will:

- Install and run the Adobe Experience Manager Author instance.
- Install and run the Adobe Experience Manager Publish instance.

What is Adobe Experience Manager?

Adobe Experience Manager is a web-based client-server system for building, managing, and deploying commercial websites and related services. A number of infrastructure-level and application-level functions are combined into a single integrated package.



Adobe Experience Manager Workflow

In Adobe Experience Manager terminology, an "instance" is a copy of Adobe Experience Manager running on a server. Adobe Experience Manager installations usually involve at least two instances, typically running on separate machines:

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

These instances are identical in terms of installed software. They are differentiated only by their configuration.

In addition, most installations use a Dispatcher:

- Dispatcher: A static web server (Apache httpd, Microsoft IIS, and so on) augmented with the Adobe Experience Manager Dispatcher module. It caches web pages produced by the Publish instance to improve performance.

Prerequisites

To install Adobe Experience Manager, you need:

- Adobe Experience Manager installation and startup JAR file
- Valid Adobe Experience Manager license key properties file
- JDK version 1.7 (http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html)
- Approximately 4 GB of free space per instance
- Approximately 4 GB of RAM

The Adobe Experience Manager installation and startup JAR file is also known as the "quickstart" file. You use the file to install Adobe Experience Manager. Once installed, the file is referred to as the Adobe Experience Manager startup file. During installation, you will notice the JAR file creates a root folder called **crx-quickstart**.

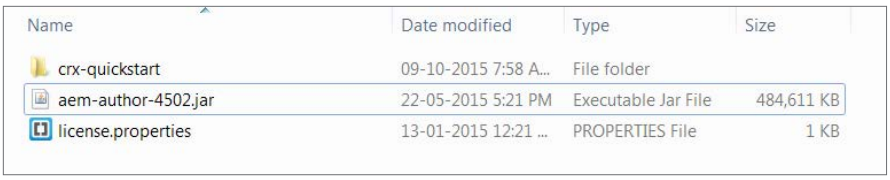
Installing Adobe Experience Manager on Your System

In general, when you want to install Adobe Experience Manager on your system, you would follow this procedure:

1. Create a specific folder structure for your Adobe Experience Manager instance.
   a. Author instance
      For Windows: C:/adobe/AEM/author
      For Mac OS or *x: /opt/adobe/AEM/author OR /Applications/AEM/author
   b. Publish instance
      For Windows: C:/adobe/AEM/publish
      For Mac OS or *x: /opt/adobe/AEM/publish OR /Applications/AEM/publish
2. Add the aem-quickstart-6.2.0.jar file along with the license.properties file to the folder, which you created earlier.
3. Rename the jar file to include the run mode as well as the port number. That is, rename the file to the format:
   `aem-<run mode>-<port number>.jar`
   For example:
   Author instance: `aem-author-4502`
   Publish instance: `aem-publish-4503`

The first time you double-click the jar file, Adobe Experience Manager will be installed on your system, creating a root folder called crx-quickstart, which serves as your repository.

A sample folder structure for an Author instance is shown below.



NOTE: The Adobe Experience Manager quickstart file is renamed for installation purposes. When running for the first time, the quickstart file will notice that it has to install Adobe Experience Manager. By renaming the file, you use a convention of passing instance name (Webpathcontext) and port number through the file name so that no user interaction is needed during the installation process. If no port number is provided in the file name, Adobe Experience Manager will select the first available port from the following list in this specific order: 1) 4502, 2) 8080, 3) 8081, 4) 8082, 5) 8083, 6) 8084, or a random port.

Perform **Task 1: Start an Adobe Experience Manager Author Instance**, from the Lab Activity section.

Perform **Task 2: Start an Adobe Experience Manager Publish Instance**, from the Lab Activity section.

Starting an Adobe Experience Manager Instance

There are two ways to start an Adobe Experience Manager instance—graphical and by command line. The latter is more powerful because the user has the possibility of providing additional performance-tuning parameters to the Java Virtual Machine (JVM).

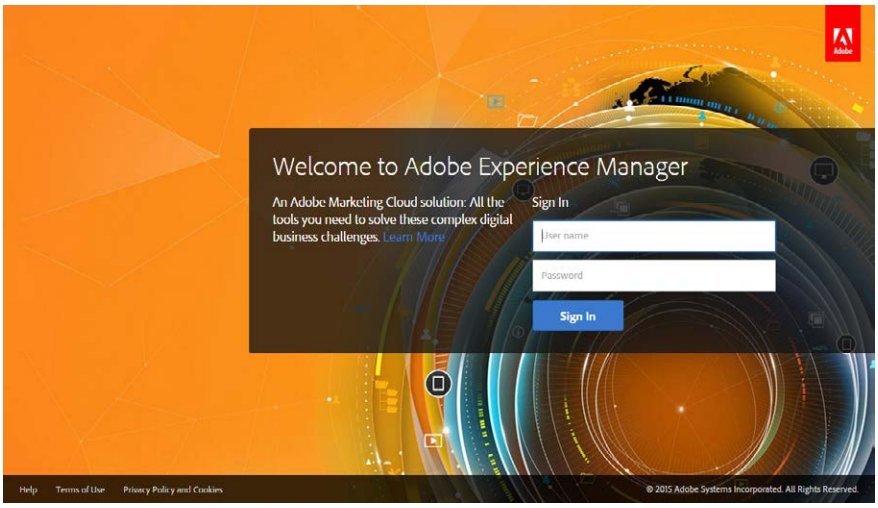
Using the GUI to Start an Adobe Experience Manager Instance

In a Windows or Mac OS environment, you can double-click the **aem-author-4502.jar** file to start an Author instance (or the **aem-publish-4503.jar** file for a Publish instance).

- Installation will take approximately 5-7 minutes, depending on your system's capabilities.
- A dialog box will pop up similar to the following:

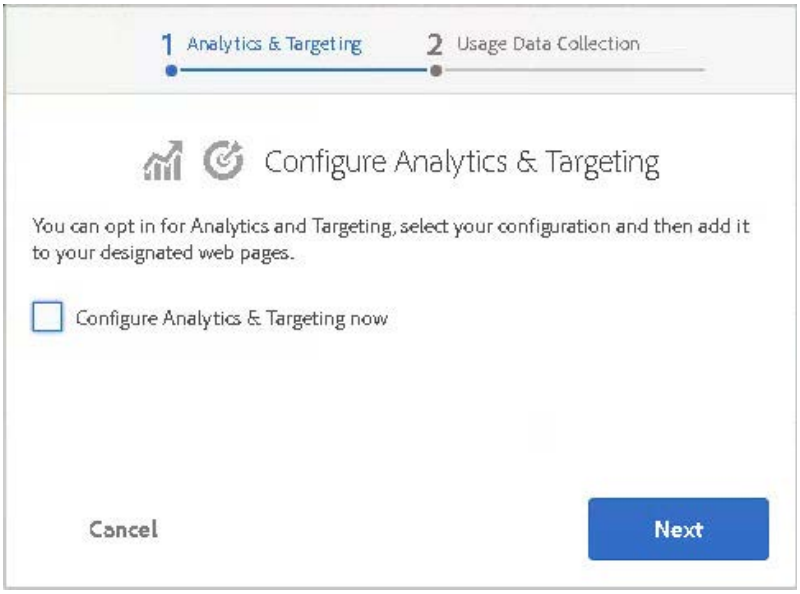
After Adobe Experience Manager starts, your default browser will open automatically, pointing to Adobe Experience Manager's start URL (where the port number is the one you defined on installation).

1. In the Sign In area that displays, enter the default administrator's credentials (**admin/admin**), and then click **Sign In**.

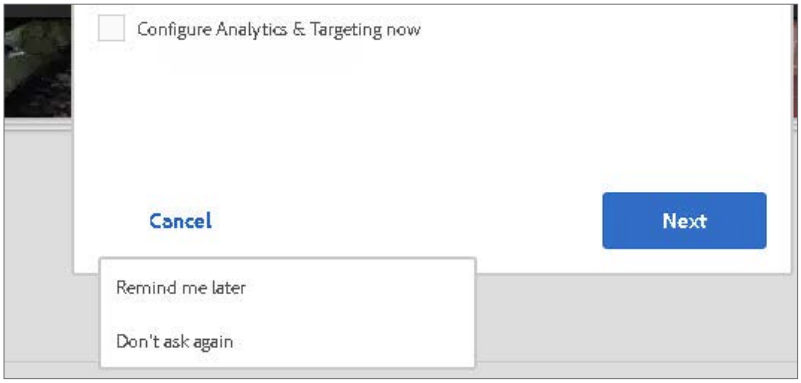


If this is the first time you are logging in, a wizard displays, asking if you want to configure Analytics & Targeting now.

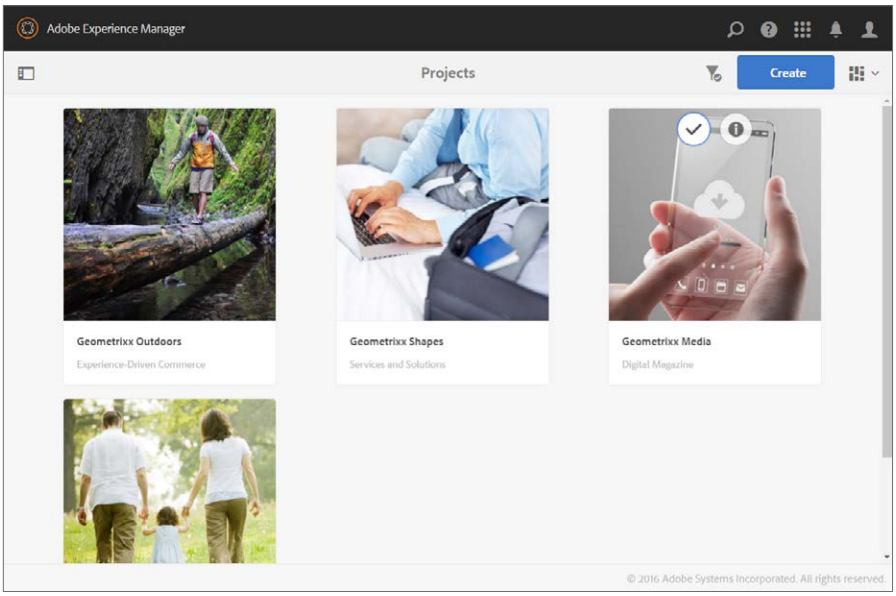
2. Click **Cancel**.



3. A hover menu opens, asking if you want to be reminded to complete the Configure Analytics & Target wizard later, or not ask again (which means it will never open again when you first log in). For this class, click **Don't' ask again**.



4. The Welcome screen displays, with different consoles available for you.



Using the Command Line to Start Adobe Experience Manager Author Instance

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

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

The Adobe Experience Manager quickstart installer will show all available command-line options without starting the server. In addition, you need to tune the JVM used for running Adobe Experience Manager. Tuning the JVM is an important and delicate task and requires a more realistic environment in terms of resources (hardware, operating system, and so on) and workload (content, requests, and so on). For now, it will be enough to know that you can start your instance (Author or Publish) using the following parameters:

-Xms --> assigns the initial heap size

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

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

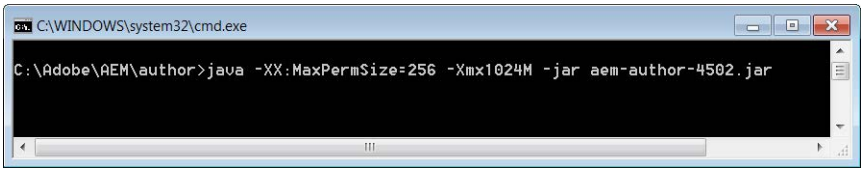
Default value	64 MB for a JVM running on 32-bit machines, or 83 MB for 64-bit machines
Recommended	Specific to physical memory available and expected traffic, but should be equal or greater than the initial size. To run Adobe Experience Manager, it is recommended to allocate at least 1024 MB of heap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

-XX:MaxPermSize --> assigns the heap to hold reflective data of the VM (for example, Java objects)

Default value	32 MB for a JVM running as a client, or 64 MB when running as a server.
Recommended	The 'PermSize' should be set to at least 128 MB for 'normal-sized' Web apps or 256 MB for larger Web apps with significant Java activity.
Syntax	-XX:MaxPermSize=128m (sets the initial perm gen size to 128 MB).

NOTE:-XX:MaxPermSize is no longer an option in JDK 1.8.

You can now install and start Adobe Experience Manager from the command line together with increasing the Java heap and perm gen size, which will improve performance. Refer to the following image for an example of the command line.



You can control the way Adobe Experience Manager is installed by defining properties via file name.

Perform **Task 3: Start and install Adobe Experience Manager using command line**, from the Lab Activity section.

Using the Command Line to Start Adobe Experience Manager Publish Instance

Enter the following command to install the publish install:

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

Lab Activity

Overview

Unlike many other applications, you can install Adobe Experience Manager using a quickstart, self-extracting JAR file. When you double-click the JAR file for the first time, everything you need is automatically extracted and installed.

In some scenarios, you need to start the Adobe Experience Manager instance using the command line, where you provide the argument during the instance start-up.

To begin, you need to configure and install two servers:

1. **Author** on localhost port number 4502
2. **Publish** on localhost port number 4503

Content creators and developers use Author server to create and manage the content. Visitors access the website and interact with it on the Publish instance.

NOTE: The port numbers would differ in a VILT environment.

Prerequisites

You need the following to complete the tasks in this module:

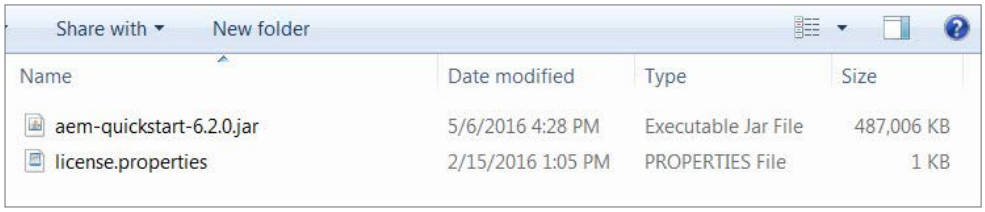
1. Adobe Experience Manager quickstart JAR file
2. license.properties file

NOTE: In VILT, the installation would have already been performed for you.

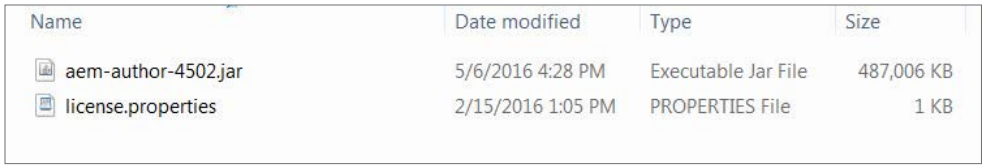
Steps

Task 1: Start an Adobe Experience Manager Author instance

1. Create a folder structure on your file system where you will store, install, and start Adobe Experience Manager. For example:
   a. Windows: C:/adobe/AEM/author
   b. MacOS X: /Applications/adobe/AEM/author or *x: /opt/adobe/AEM/author
2. Copy the **aem-quickstart-6.2.0.jar** and **license.properties** files from the USB contents.



3. Rename the **aem-quickstart-6.2.0.jar** file to **aem-author-4502.jar**:
   a. aem = Application
   b. author = Web Content Management (WCM) mode it will run in (in this case, Author)
   c. 4502 = Port it will run in (any available port is acceptable).



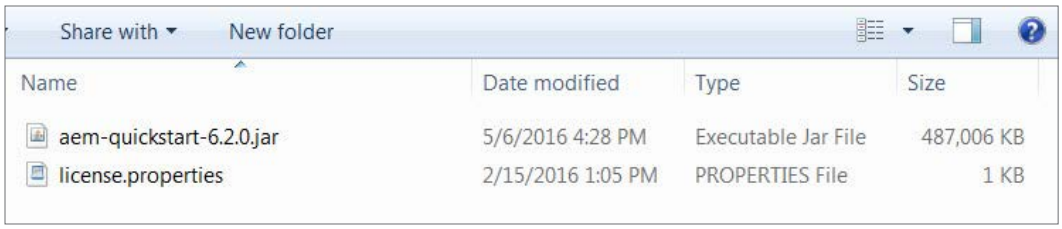
4. In a Windows or MacOS X environment, double-click the aem-author-4502.jar file. Installation will take approximately 5–7 minutes depending on your system's capabilities.

After Adobe Experience Manager Author instance has started successfully, the start-up screen will change to something similar to the following:

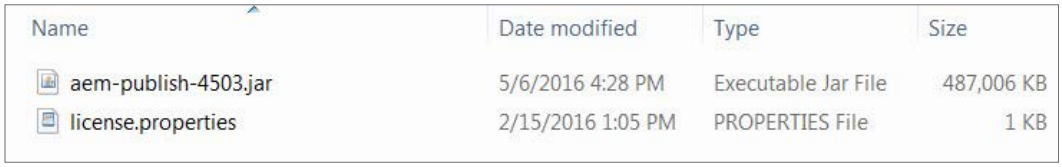
In addition, after Adobe Experience Manager starts, your default browser will automatically open to Adobe Experience Manager's start URL (where the port number is the one you defined on installation);. For example: http://localhost:4502

Task 2: Start an Adobe Experience Manager Publish instance

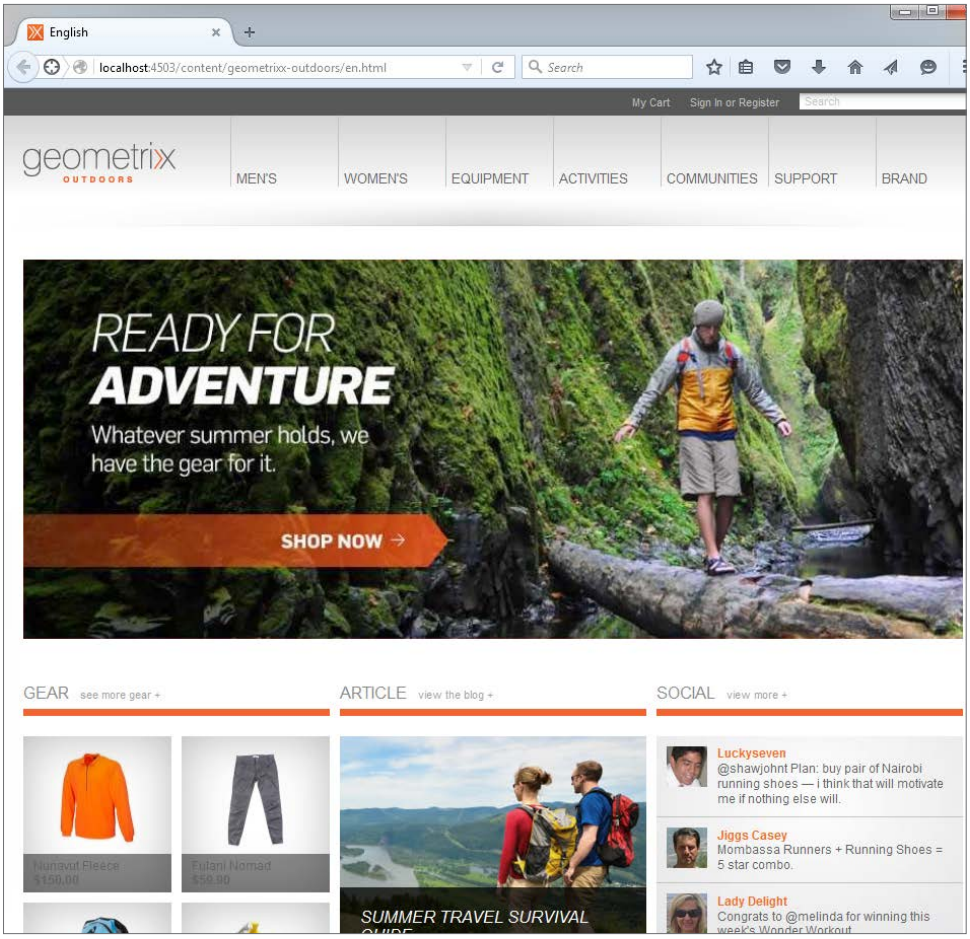
1. Create a folder structure on your file system where you will store, install, and start the Adobe Experience Manager. For example:
   a. Windows: C:/adobe/AEM/publish
   b. MacOS X: /Applications/adobe/AEM/publish or *x: /opt/adobe/AEM/publish
2. Copy the `aem-quickstart-6.2.0` JAR and `license.properties` files from USB contents.



3. Rename the `aem-quickstart-6.2.0.jar` file to `aem-author-4503.jar`
   a. aem = Application
   b. publish = WCM mode it will run in
   c. 4503 = Port it will run in (any available port is acceptable)



4. Double-click the **aem-publish-4503.jar** file to start the Publish instance.



After Adobe Experience Manager Publish instance has started successfully, the start-up screen will change to something similar to the website shown.

In addition, the Adobe Experience Manager login page opens from your default browser (where the port number is the one you defined on installation). For example: http://localhost:4503

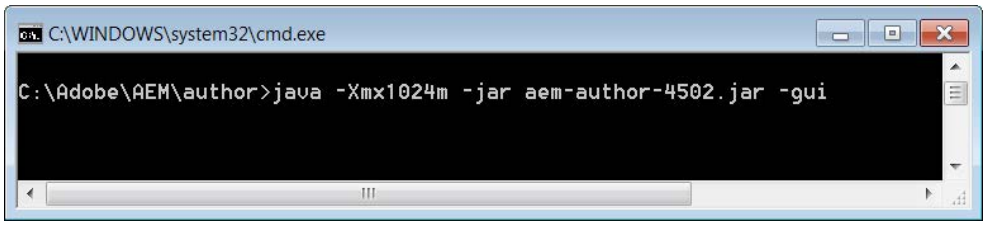
The following screen appears once the Publish instance is up and running. You have now successfully installed and started Adobe Experience Manager Author and Publish instances on localhost. To start Adobe Experience Manager in the future, double-click the renamed aem-quickstart-6.2.0.jar file. For example: aem-author-4502.jar

Task 3: Start and install Adobe Experience Manager using command line

You already have an author instance and a publish instance running. Perform this task only when necessary or as an add-on exercise to try out beyond this class.

This is a powerful method because the user can provide additional performance-tuning parameters to the Java Virtual Machine (JVM). On Windows, MacOS X, or *x, you can install or start Adobe Experience Manager from the command line, while increasing the Java heap size, which improves performance.

A typical command line start will have the following:




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

In your command prompt, navigate to the **Adobe\AEM\author** directory, and use the following command to install Adobe Experience Manager, without installing the Geometrixx sites:

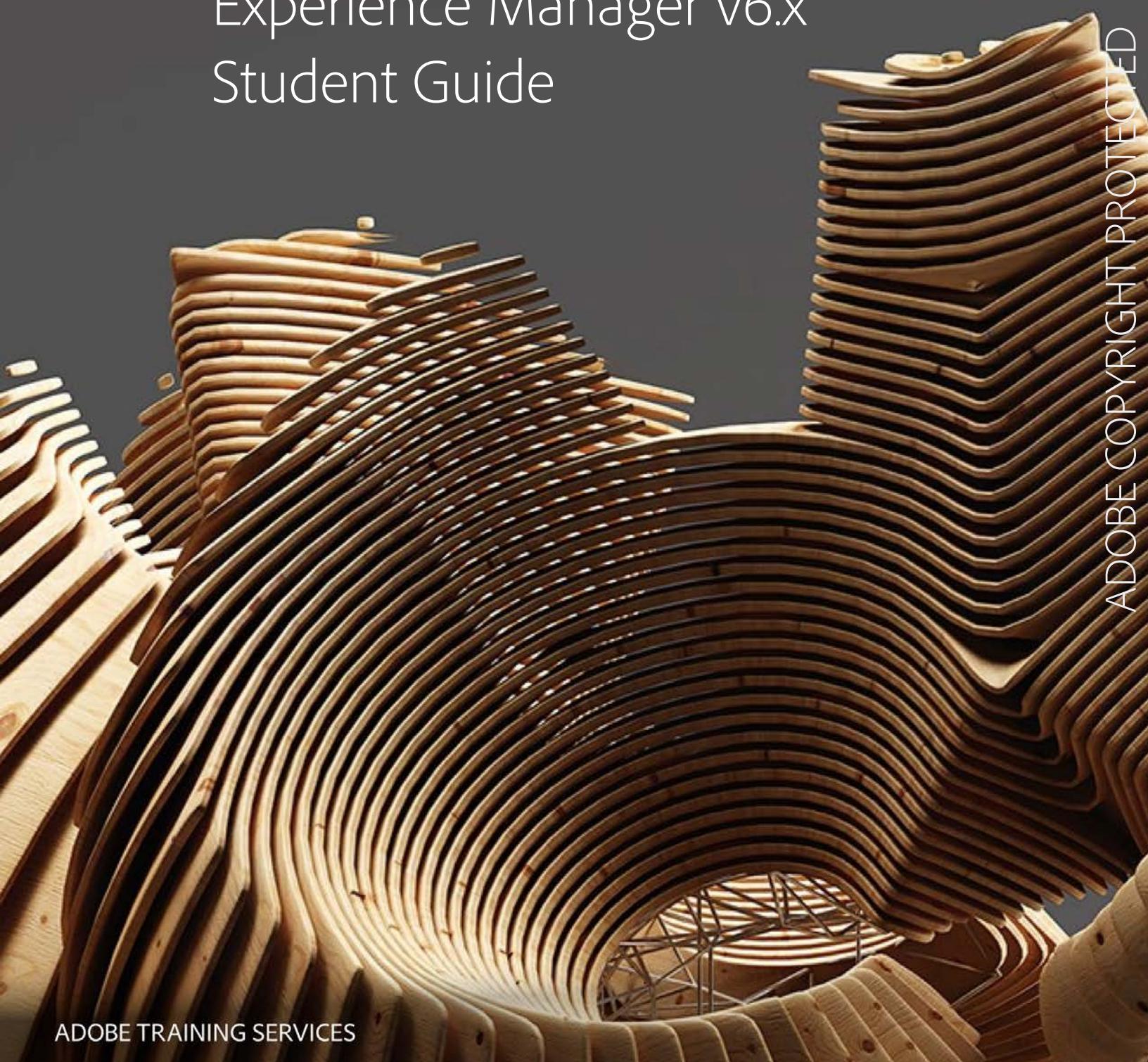
```
java -jar aem-author-p4502.jar -r author, nosamplecontent -gui
```

Summary

You should now be able to:

- Install and run the Adobe Experience Manager Author instance
- Install and run the Adobe Experience Manager Publish instance

Develop Websites and Components using Adobe Experience Manager v6.x Student Guide



ADOBE TRAINING SERVICES

Develop Websites and Components using Adobe Experience Manager v6.x

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

September 13, 2016

Table of Contents

1 Architecture Stack

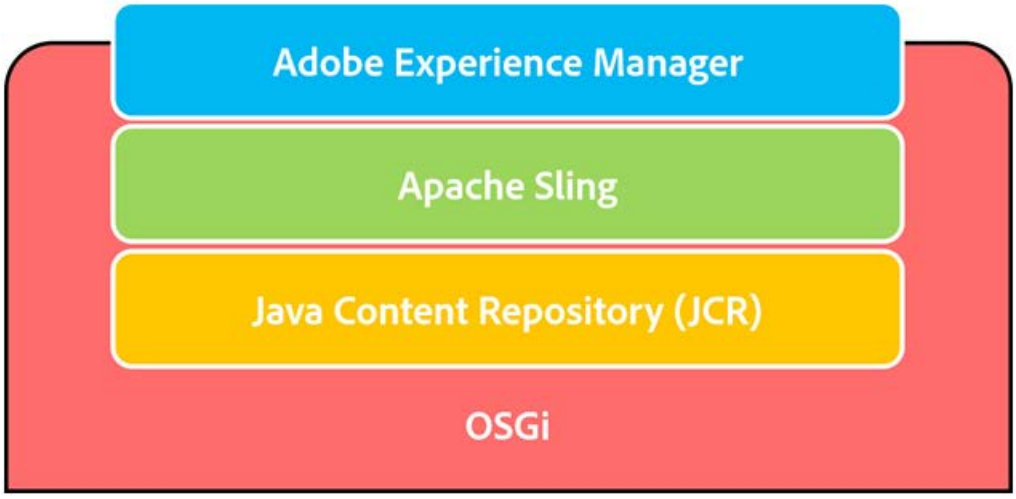
1.1 What is Adobe Experience Manager?

- Adobe Experience Manager is a web-based client-server system for building, managing, and deploying commercial websites and related services.
- A number of infrastructure-level and application-level functions are combined into a single integrated package.

1.2 Basics of the Architecture Stack

Adobe Experience Manager is a Java web application, and is based on technologies such as Open Service Gateway Initiative (OSGi), Java Content Repository (JCR), and Apache Sling.

The following diagram is a high-level view of the architecture stack.



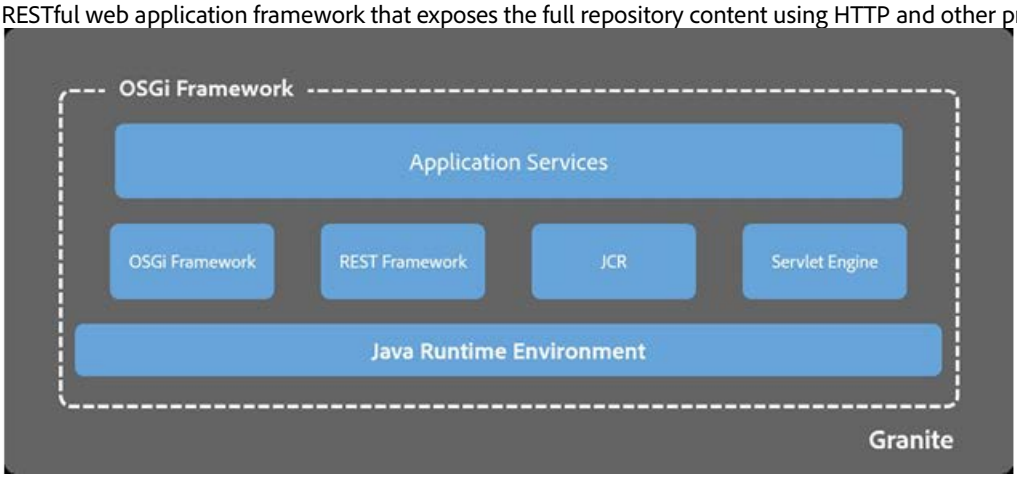
1.3 Introduction to the Granite Platform

Adobe Experience Manager runs on the Granite platform within an OSGi framework. This platform provides various components such as:

- An application launcher
- An OSGi framework into which everything is deployed
- A number of OSGi compendium services to support building applications
- A comprehensive Logging Framework providing various logging APIs
- The CRX Repository implementation of the JCR API specification
- The Apache Sling framework

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

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



Apache Felix is an open source implementation of the OSGi that the Adobe Experience Manager framework makes use of. It provides a dynamic runtime environment, where code and content bundles can be loaded, unloaded, and reconfigured at runtime.

1.4 Introduction to the Java Content Repository

The JCR, specifically JSR-283, is a database that supports structured and unstructured content, versioning, and observation. All data pertaining to Adobe Experience Manager such as HTML, CSS, JS/Java, Images, and Videos are stored in the JCR object database. It is built with Apache Jackrabbit Oak. The Adobe implementation of JSR-283 is the Content Repository eXtreme (CRX), and the version of CRX that comes with Adobe Experience Manager supports JCR 2.x.
In addition to CRX, Adobe Experience Manager can also work with other JCR repositories such as Apache Jackrabbit and with a number of non-JCR data stores through connectors. As Adobe Experience Manager is built on top of this standard, it is capable of pulling content not just from its built-in CRX repository, but from any JCR-compliant source, such as a third-party repository (for example, Jackrabbit) or a connector that exposes legacy storage through JCR.

Advantages of using JCR

1. JCR provides a generic application data store for structured and unstructured content. While file systems provide excellent storage for unstructured, hierarchical content, and databases provide storage for structured data, JCR provides the best of both data storage architectures.
2. It supports namespaces. Namespaces prevent naming collisions among items and node types that come from different sources and application domains. JCR namespaces are defined with a prefix, delimited by a single colon (:); for example, jcr:title.

1.5 Introduction to Apache Sling

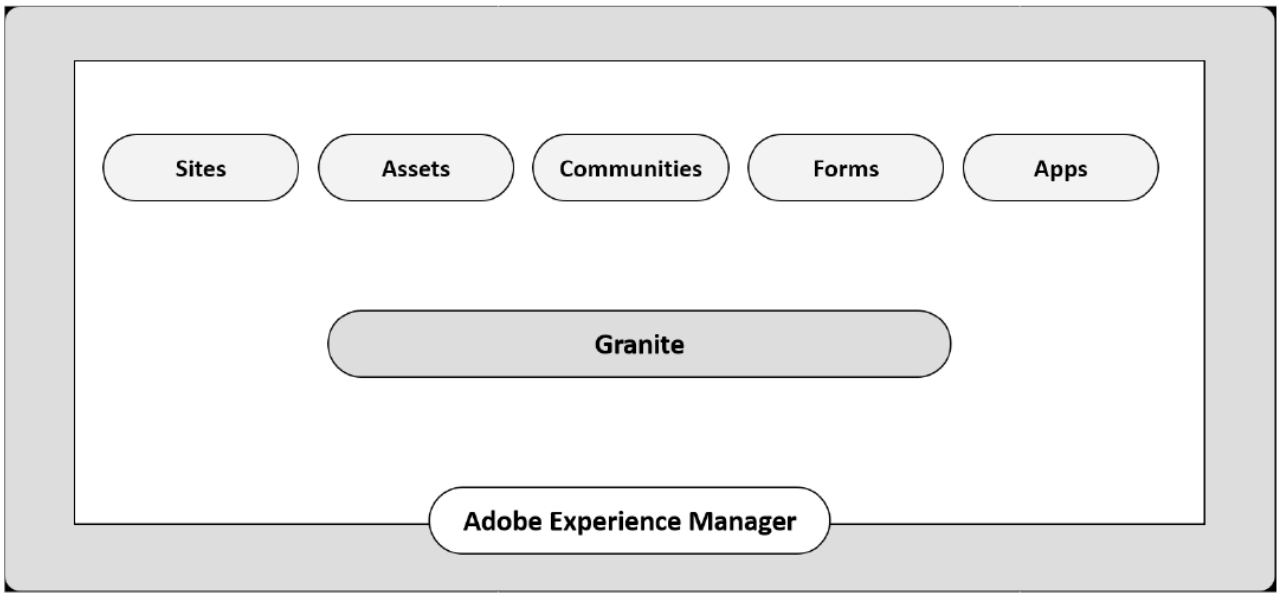
Apache Sling is a web application framework for content-centric applications, and uses a Java Content Repository, such as Apache Jackrabbit or CRX to store and manage content.

- It is based on REST principles
- Its applications are built as a series of OSGi bundles
- It is resource-oriented, and usually map into JCR nodes

The embedded Apache Felix OSGi framework and console provides a dynamic runtime environment, where code and content bundles can be loaded, unloaded, and reconfigured at runtime. Sling makes it easy to implement simple applications, while providing an enterprise-level framework for more complex applications.
Sling assumes that 'Everything is a Resource'. That is, Sling is resource oriented, where the resources have URLs, and each resource is mapped into a JCR node. A request URL is first resolved to a resource, and then based on the resource, it selects the Servlet or script to handle that request. Servlets and scripts are handled uniformly in that they are represented as resources themselves and are accessible by a resource path. This means that every script, Servlet, filter, error handler, and so on is available from the ResourceResolver just like normal content—providing data to be rendered on request.

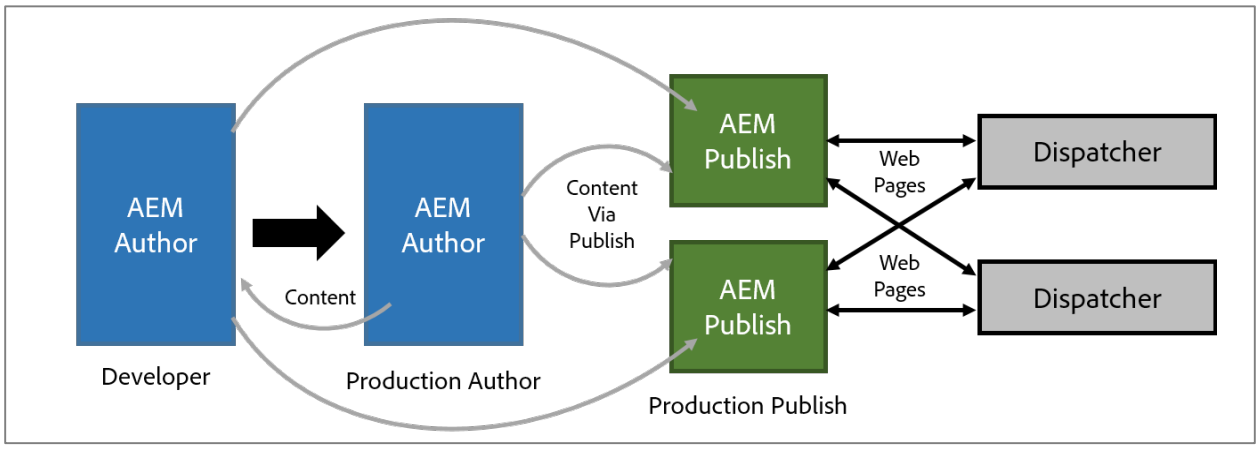
1.6 The Functional Building Blocks of Adobe Experience Manager

The Adobe Experience Manager application modules such as Sites, Assets, Communities, Forms, and Apps, sit on top of the Adobe Experience Manager shared framework, a framework known as Granite. This framework includes all the application layer functionality such as mobile functionality, multisite manager, taxonomy management, and workflow. These functionalities are shared among all the application modules. In addition to these functionalities, these Adobe Experience Manager applications share the same infrastructure and UI framework, and are very tightly integrated with each other. All of this is encapsulated within the OSGi container.



2 Installation

2.1 Adobe Experience Manager Workflow



Adobe Experience Manager Workflow

In Adobe Experience Manager terminology, an "instance" is a copy of Adobe Experience Manager running on a server. Adobe Experience Manager installations usually involve at least two instances, typically running on separate machines:

- Author: An Adobe Experience Manager instance used to create, upload, and edit content, and administer the website. After content is ready to go live, it is replicated to the Publish Instance.
- Publish: An Adobe Experience Manager instance that serves the published content to the public.
- **NOTE:** In a basic sense, the author and publish instances are the same software stack but two different run modes.

These instances are identical in terms of installed software. They are differentiated only by their configuration.

In addition, most installations use a Dispatcher:

- Dispatcher: A static web server (Apache httpd, Microsoft IIS, and so on) augmented with the Adobe Experience Manager Dispatcher module. It caches webpages produced by the Publish instance to improve performance.

2.2 Prerequisites

To install Adobe Experience Manager, you need:

- Adobe Experience Manager installation and startup JAR file
- Valid Adobe Experience Manager license key properties file
- JDK version 1.8 (http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)
- Approximately 4 GB of free space per instance
- Approximately 4 GB of RAM (at the **very minimum!)**

The Adobe Experience Manager installation and startup JAR file is also known as the "quickstart" file. You use the file to install Adobe Experience Manager. Once installed, the file is referred to as the Adobe Experience Manager startup file. During installation, you will notice the JAR file creates a root folder called **crx-quickstart**.

You will also need to set environment variables as part of setting up your JDK 1.8.

2.3 Installing Adobe Experience Manager on Your System

In general, when you want to install Adobe Experience Manager on your system, you would follow this procedure:

1. Create a specific folder structure for your Adobe Experience Manager instance.

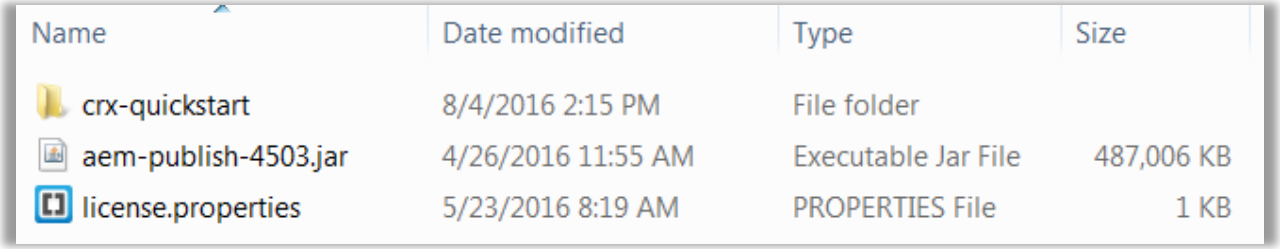
    a. Author instance
    For Windows: C:/adobe/AEM/author
    For Mac OS or *x: /opt/adobe/AEM/author OR /Applications/AEM/author

    b. Publish instance
    For Windows: C:/adobe/AEM/publish
    For Mac OS or *x: /opt/adobe/AEM/publish OR /Applications/AEM/publish

3. Add the **AEM Quickstart JAR** file along with the license.properties file to the folder, which you created earlier.
4. Rename the jar file to include the run mode as well as the port number. That is, rename the file to the format:
    aem-<run mode>-<port number>.jar
    For example,
    Author instance: `aem-author-4502`
    Publish instance: `aem-publish-4503`

The first time you double-click the jar file, Adobe Experience Manager will be installed on your system, creating a root folder called `crx-quickstart`, which serves as your repository.

A sample folder structure for an Author instance is shown below.



NOTE: The Adobe Experience Manager quickstart file is renamed for installation purposes. When running for the first time, the quickstart file will notice that it has to install Adobe Experience Manager. By renaming the file, you use a convention of passing instance name (Webpathcontext) and port number through the file name so that no user interaction is needed during the installation process. If no port number is provided in the file name, Adobe Experience Manager will select the first available port from the following list in this specific order: 1) 4502, 2) 8080, 3) 8081, 4) 8082, 5) 8083, 6) 8084, or a random port.

Perform Task - Start an Adobe Experience Manager Author Instance, from the Lab Activity section.

Perform Task - Start an Adobe Experience Manager Publish Instance, from the Lab Activity section.

2.4 Starting an Adobe Experience Manager Instance

There are many ways of starting an Adobe Experience Manager instance, two of which are—graphical and by command line. The latter is more powerful because the user has the possibility of providing additional performance-tuning parameters to the Java Virtual Machine (JVM).

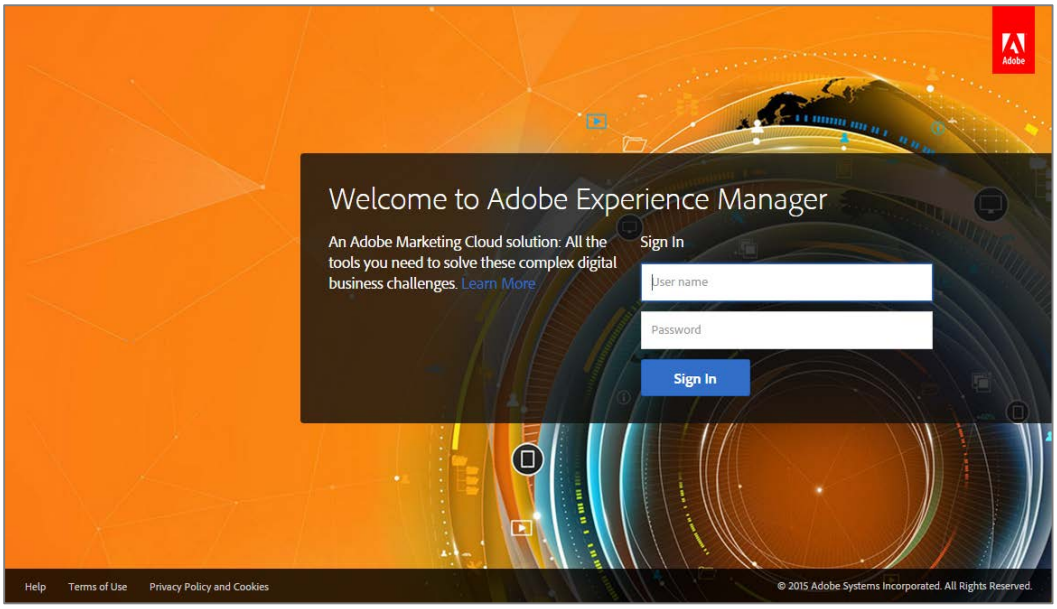
2.4.1 Using the JAR FILE to Start an Adobe Experience Manager Instance

In a Windows or Mac OS environment, you can double-click the `aem-author-4502.jar` file to start an Author instance (or the `aem-publish-4503.jar` file for a Publish instance).

- Installation will take approximately 5-7 minutes, depending on your system's capabilities.
- A dialog box will pop up similar to the following (this is known as the GUI):

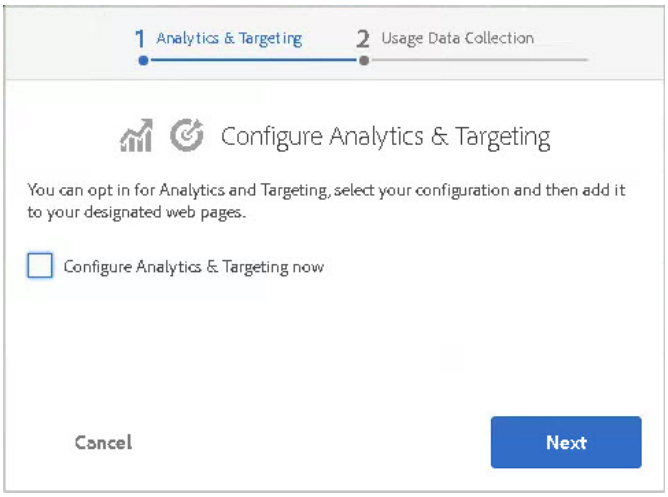
After Adobe Experience Manager starts, your default browser will open automatically, pointing to Adobe Experience Manager's start URL (where the port number is the one you defined on installation).

1. In the Sign In area that displays, enter the default administrator's credentials (`admin/admin`), and then click **Sign In**.

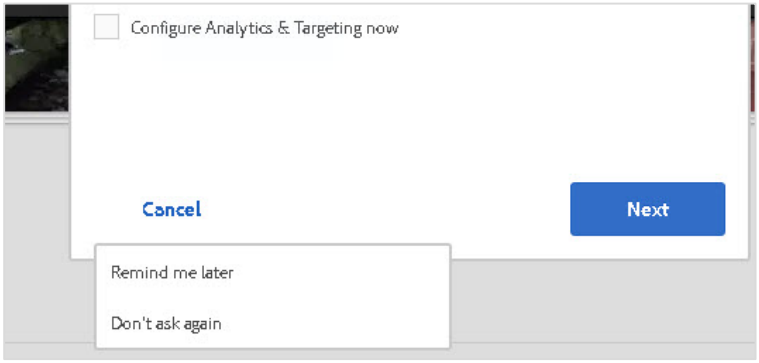


If this is the first time you are logging in, a wizard displays, asking if you want to configure Analytics & Targeting now.

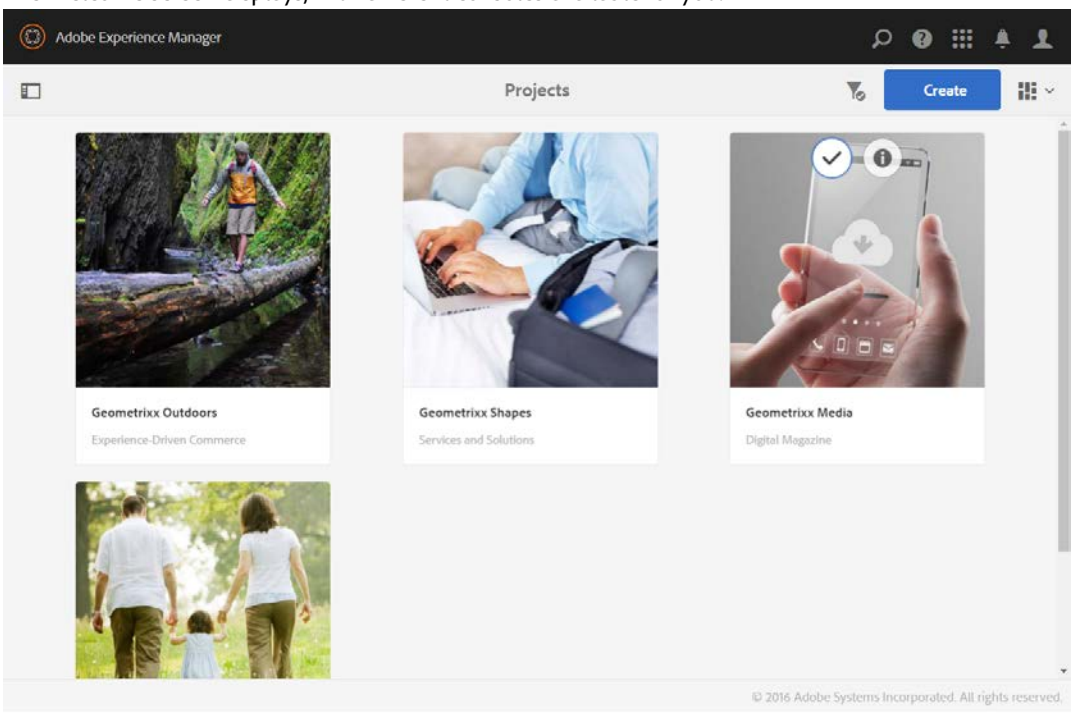
2. Click **Cancel**.



3. A hover menu opens, asking if you want to be reminded to complete the Configure Analytics & Target wizard later, or not ask again (which means it will never open again when you first log in). For this class, click Don't' ask again.



4. The Welcome screen displays, with different consoles available for you.



2.4.2 Using the Command Line to Start Adobe Experience Manager Author Instance

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

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

The Adobe Experience Manager quickstart installer will show all available command-line options without starting the server. In addition, you need to tune the JVM used for running Adobe Experience Manager. Tuning the JVM is an important and delicate task and requires a more realistic environment in terms of resources (hardware, operating system, and so on) and workload (content, requests, and so on). For now, it will be enough to know that you can start your instance (Author or Publish) using the following parameters:

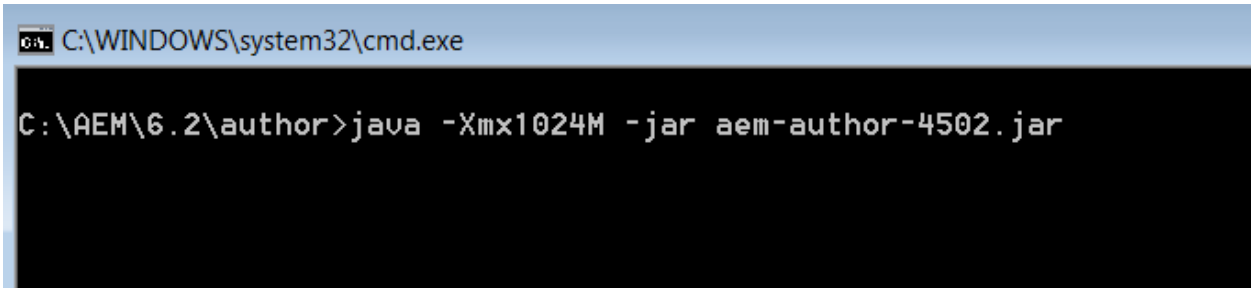
-Xms --> assigns the initial heap size

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

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

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

You can now install and start Adobe Experience Manager from the command line together with increasing the Java heap size, which will improve performance. See the image below for an example of the command line.



You can control the way Adobe Experience Manager is installed by defining properties via file name.

Perform Task - Start and install Adobe Experience Manager using command line, from the Lab Activity section.

2.4.3 Using the Command Line to Start Adobe Experience Manager Publish Instance

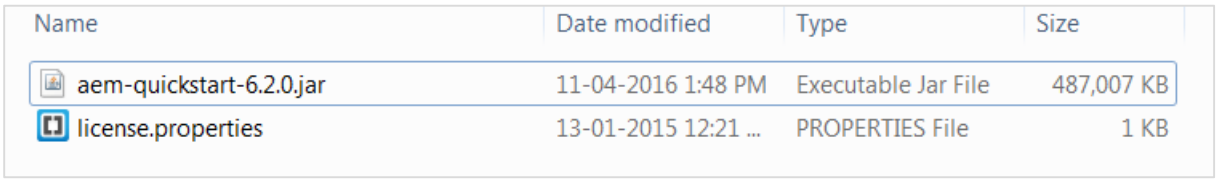
In your command prompt, navigate to the directory **/adobe/AEM/publish**, and enter the following command to install the publish instance:

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

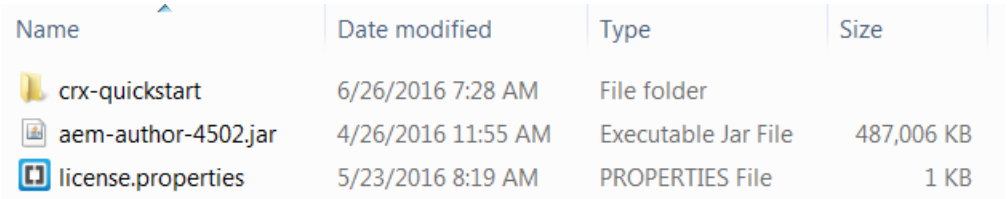
2.5 Lab Activity

Task - Start an Adobe Experience Manager Author instance

1. Create a folder structure on your file system where you will store, install, and start Adobe Experience Manager. For example:
    a. Windows: C:/adobe/AEM/author
    b. MacOS X: /Applications/adobe/AEM/author or *x: /opt/adobe/AEM/author
2. Copy the `aem-quickstart-6.2.0.jar` and `license.properties` files from USB contents.



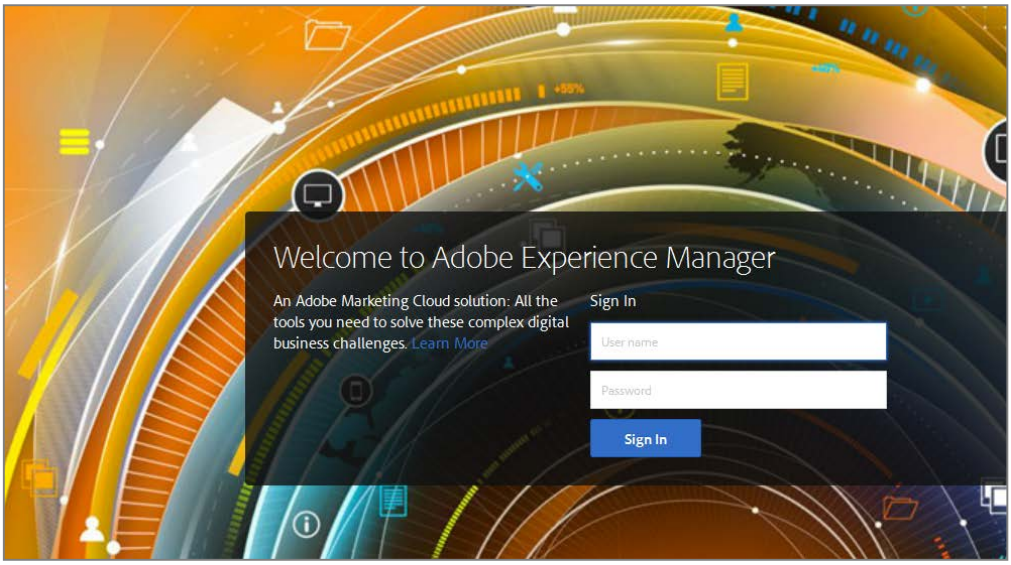
3. Rename the `aem-quickstart-6.2.0.jar` file to `aem-author-4502.jar`:
    a. aem = Application
    b. author = Web Content Management (WCM) mode it will run in (in this case, Author)
    c. 4502 = Port it will run in (any available port is acceptable)



4. In a Windows or MacOS X environment, double-click the `aem-author-4502.jar` file. Installation will take approximately 5–7 minutes depending on your system's capabilities.

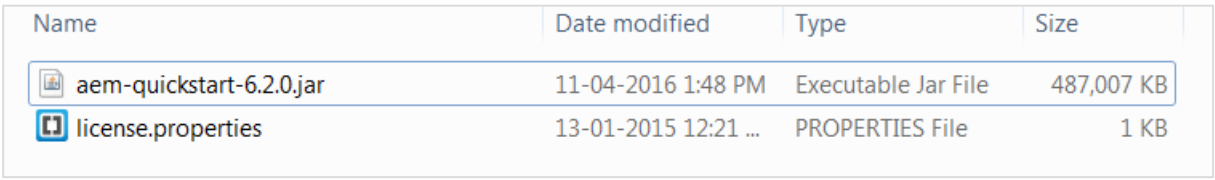
After Adobe Experience Manager Author instance has started successfully, the start-up screen will change to something similar to the following:

In addition, after Adobe Experience Manager starts, your default browser will automatically open to Adobe Experience Manager's start URL (where the port number is the one you defined on installation); for example: http://localhost:4502



Task - Start an Adobe Experience Manager Publish instance

1. Create a folder structure on your file system where you will store, install, and start the Adobe Experience Manager. For example:
    a. Windows: C:/adobe/AEM/publish
    b. MacOS X: /Applications/adobe/AEM/publish or *x: /opt/adobe/AEM/publish
2. Copy the `aem-quickstart-6.2.0` JAR and `license.properties` files from USB contents.
3. Rename the `aem-quickstart-6.2.0.jar` file to `aem-publish-4503.jar`
    a. aem = Application
    b. publish = WCM mode it will run in
    c. 4503 = Port it will run in (any available port is acceptable)

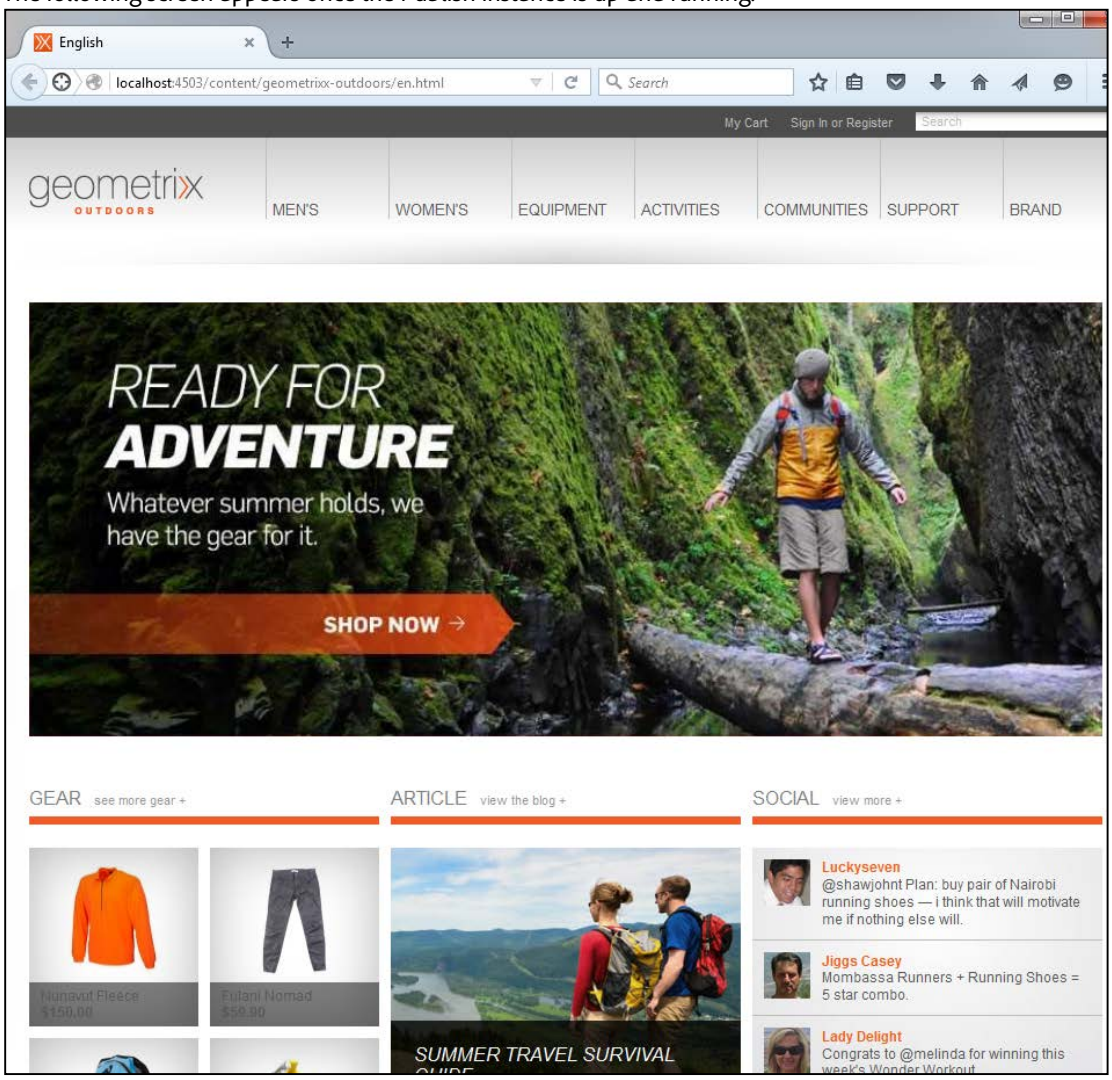


4. Double-click the `aem-publish-4503.jar` file to start the Publish instance.

After Adobe Experience Manager Publish instance has started successfully, the start-up screen will change to something similar to the following:

In addition, the Adobe Experience Manager login page opens from your default browser (where the port number is the one you defined on installation); for example, http://localhost:4503

The following screen appears once the Publish instance is up and running.



You have now successfully installed and started Adobe Experience Manager Author and Publish instances on localhost. To start Adobe Experience Manager in the future, double-click the renamed `aem-quickstart-6.2.0.jar` file; for example, `aem-author-4502.jar`.

Task - Start and install Adobe Experience Manager using command line [optional]

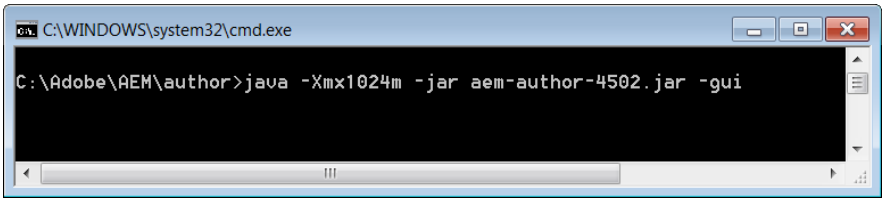
You already have an author instance and a publish instance running. Perform this task only when necessary or as an add-on exercise to try out beyond this class.

This is a powerful method because the user can provide additional performance-tuning parameters to the Java Virtual Machine (JVM). On Windows, MacOS X, or *x, you can install or start Adobe Experience Manager from the command line, while increasing the Java heap size, which improves performance.

A typical command line start will have the following:

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

This example below starts AEM author runmode with a specific memory allocation to the JVM and the GUI window "on":



In your command prompt, navigate to the Adobe\AEM\author directory, and use the following command to install Adobe Experience Manager, without installing the Geometrixx sites:

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

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

Summary

You should now be able to:

- install and run the Adobe Experience Manager Author instance
- install and run the Adobe Experience Manager Publish instance

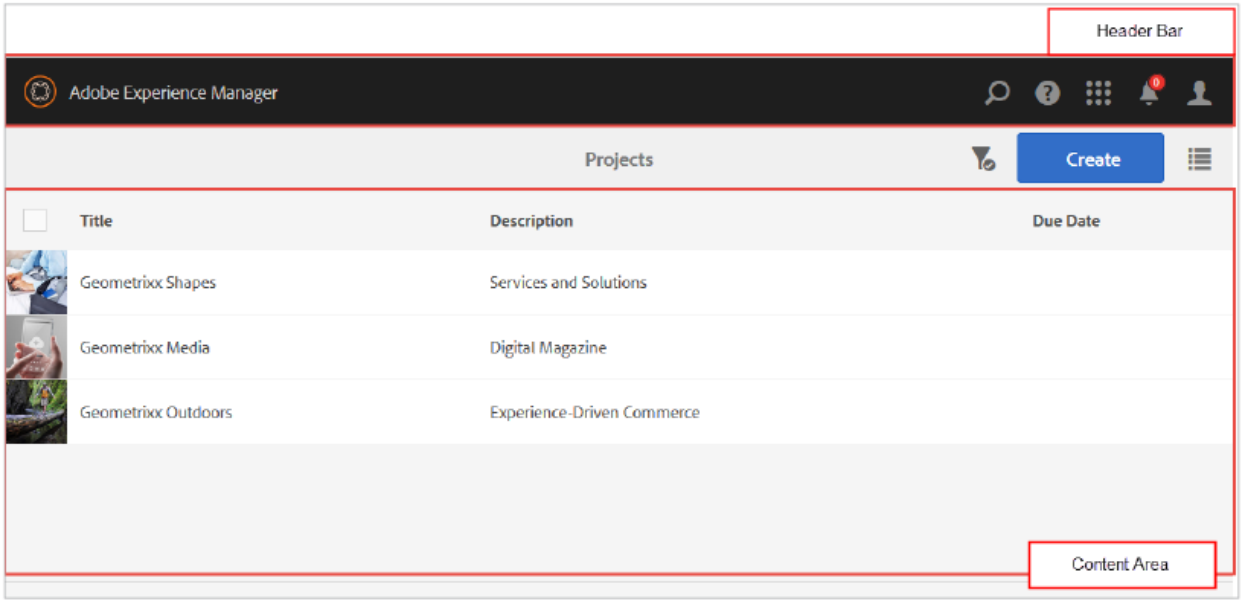
3 Authoring Basics

3.1 Introduction to Touch UI

Adobe Experience Manager User Interface (UI) combines advantages of web interface with fluidity and responsiveness that is usually associated with desktop applications.

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

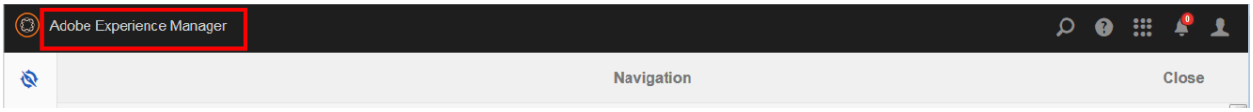
Let's familiarize ourselves with the key areas of Adobe Experience Manager Touch UI. When you load the application, you are presented with a screen that includes the header bar, and the content area.



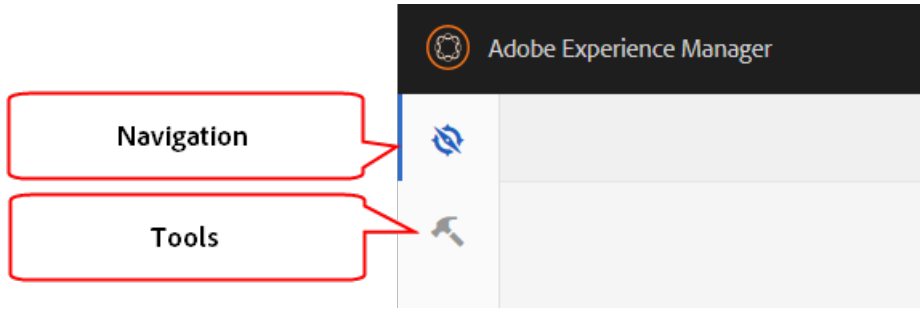
The Header bar has the following elements available globally:

Navigating to Consoles and Tools

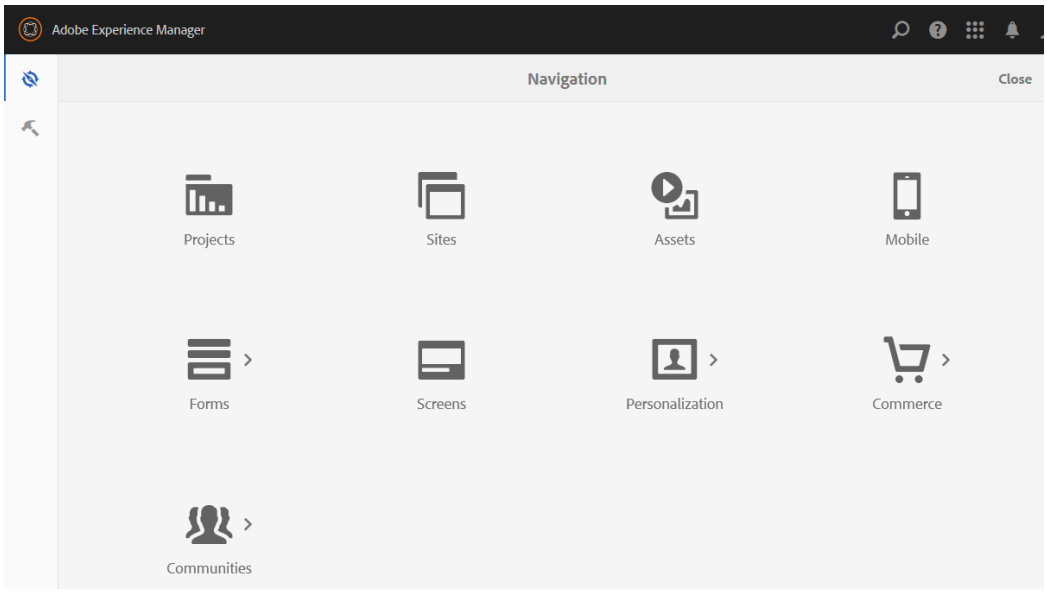
Two major consoles that you need to access during this training are the **consoles** and the **tools**. You can access these links by clicking **Adobe Experience Manager** in the upper-left corner of the screen.



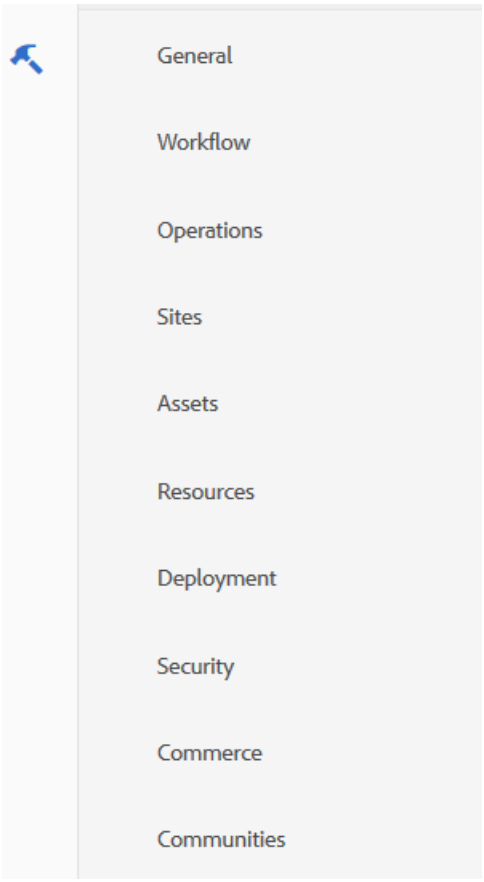
You are presented with the following options:



- **Navigation**—This option privides access to consoles such as Projects, Sites, Assets, Mobiles, and so on.



- **Tools**—This option provides access to a number of specialized tools and consoles that help you administer your websites, digital assets, and other aspects of your content repository.



3.2 Creating and Editing Pages

A page in Adobe Experience Manager is similar to a web page, and contains components such as text, image, videos, and many more. These pages are created from templates that define the structure of the page—including where each component has to be placed.

3.2.1 Creating Pages

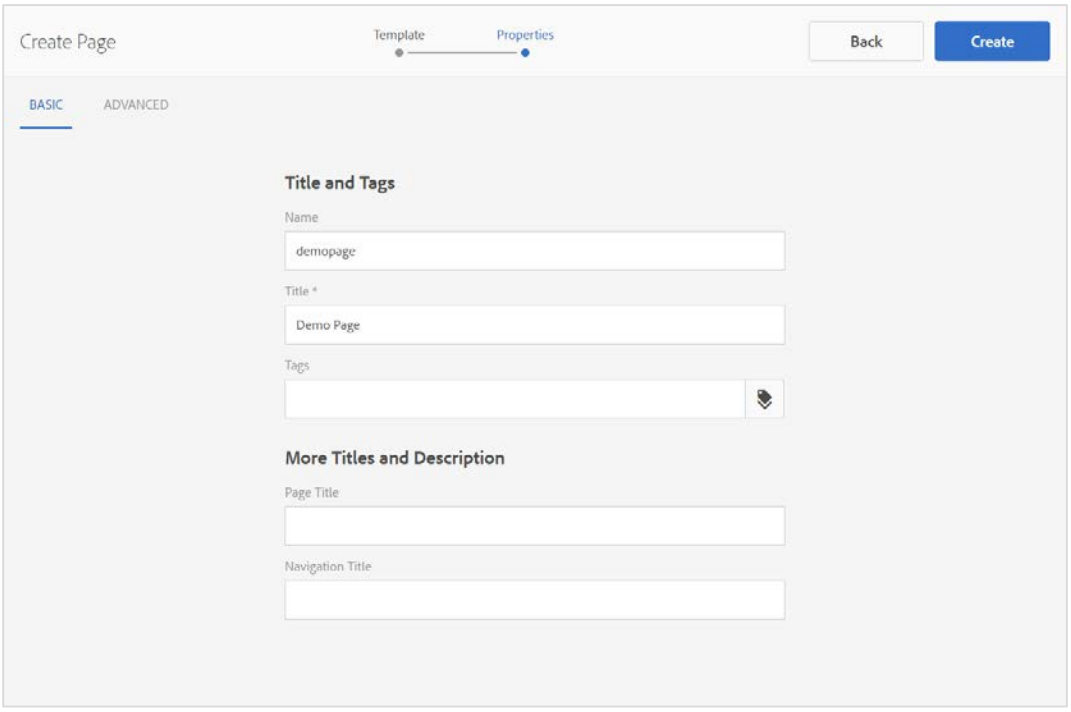
To organize website within Adobe Experience Manager, you need to first create and name your content pages, which allows for easy accessibility.

Before you add content, you must create a page within a site. Two key fields are used while creating a page:

- **Title**: This is displayed to the user in the console and shown at the top of the content page when editing. This field is mandatory.
- **Name**: This is used to generate the URI. User input for this field is optional. If not specified, the name is derived from the title.
  - Name of the page should be in lowercase, as it's a recommended practice for URLs (some web servers are case-sensitive while others aren't).
  - NOTE: Only the following characters are allowed in the Name field: 'a' through 'z', 'A' through 'Z', '0' through '9', '-'

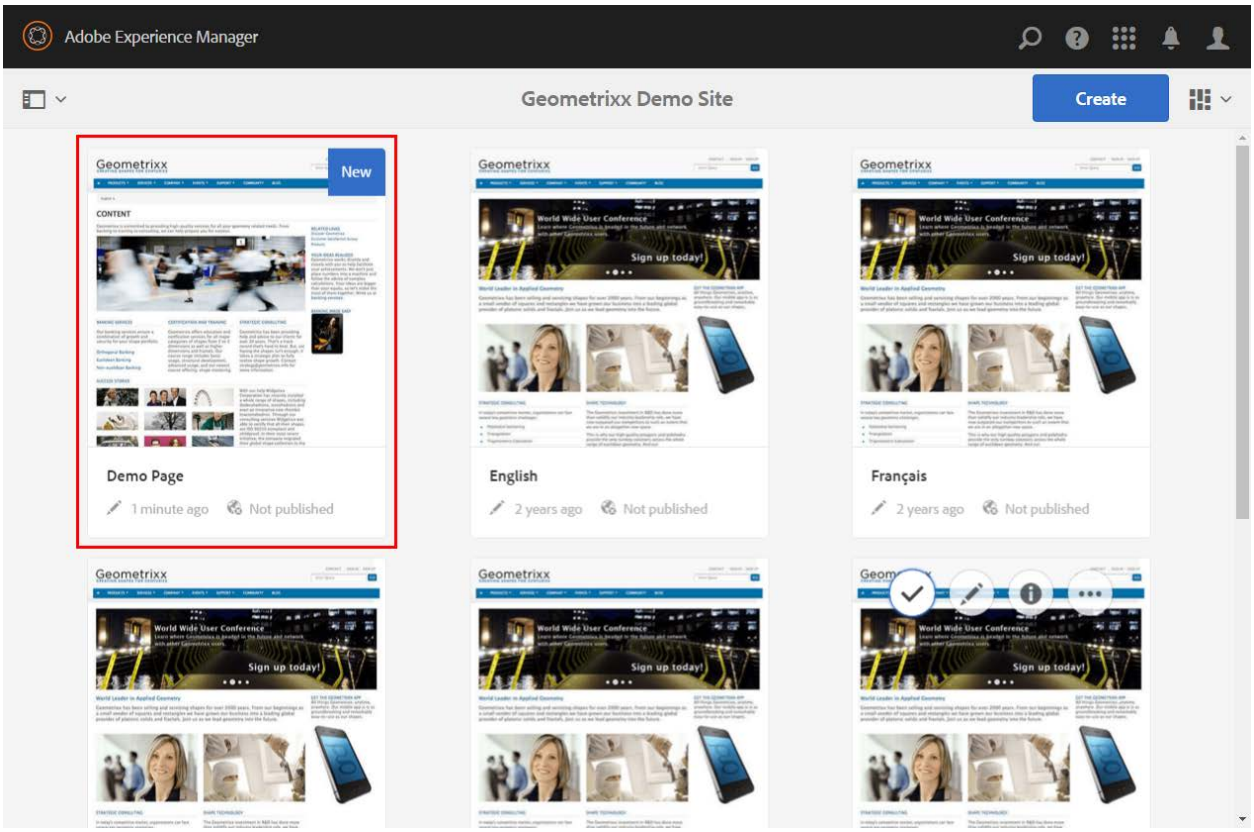
Steps to create a page:

1. Log in to Adobe Experience Manager, and navigate to the Sites console.
2. While in the Column view, select the site for which you want to create a page, and click **Create** > **Page**.

3. Click **Create** > **Page** from the actions bar.
4. In the **Template** wizard, select the **Geometrixx Content Page** template, and then click **Next** to proceed.
5. In the **Properties** wizard, enter these values for the following fields:
   - Name – demopage
   - Title – Demo Page
   - 

6. Click **Create** to complete the process.
7. Click **Done** from the **Page Created** dialog box.

The new page will appear as a child of the English page under the Geometrixx Demo Site. You can create subpages of any page using the same process.



3.2.2 Editing Pages

Once the page is created, you can edit the content of the page. Content is added using components (content placeholders), which you can drag and drop onto the page. You can add, edit and delete components from the page.

Let's edit the page and add content using components such as Text, Image, and Text & Image on the page you just created.

Steps to add text to the page:

1. Select the page that you just created using the selection mode and click **Edit** from the actions bar. Make sure the page is opened in **Edit** mode by looking in the upper-right and seeing Edit listed.
2. To add **Text** component to the page, click the **Toggle Side Panel** icon.
3. Click **Components** in the side panel.
4. In the **Filter** field, enter **Text** (to search for the Text component) and press **Enter**. The search yields three results, which all contain the word, text.
5. Drag and drop the **Text** component into the **Drag components here** area. The **Drag components here** area changes to the **Text** area.
6. Click the **Text** component, and then click **Edit** (pencil icon) from the component toolbar. A text editor opens in the same tab.
7. Add sample text and click **Done** (checkmark) to save the changes. The text is added to the page.

To add an image to the page:

8. In **Demo Page**, double-click **Drag components here** and select **Image** from the list of components.

9. Choose any image from the **Assets** tab and drag and drop the image onto the **Image** component. The image will be added to the page.

4 Developer Tools

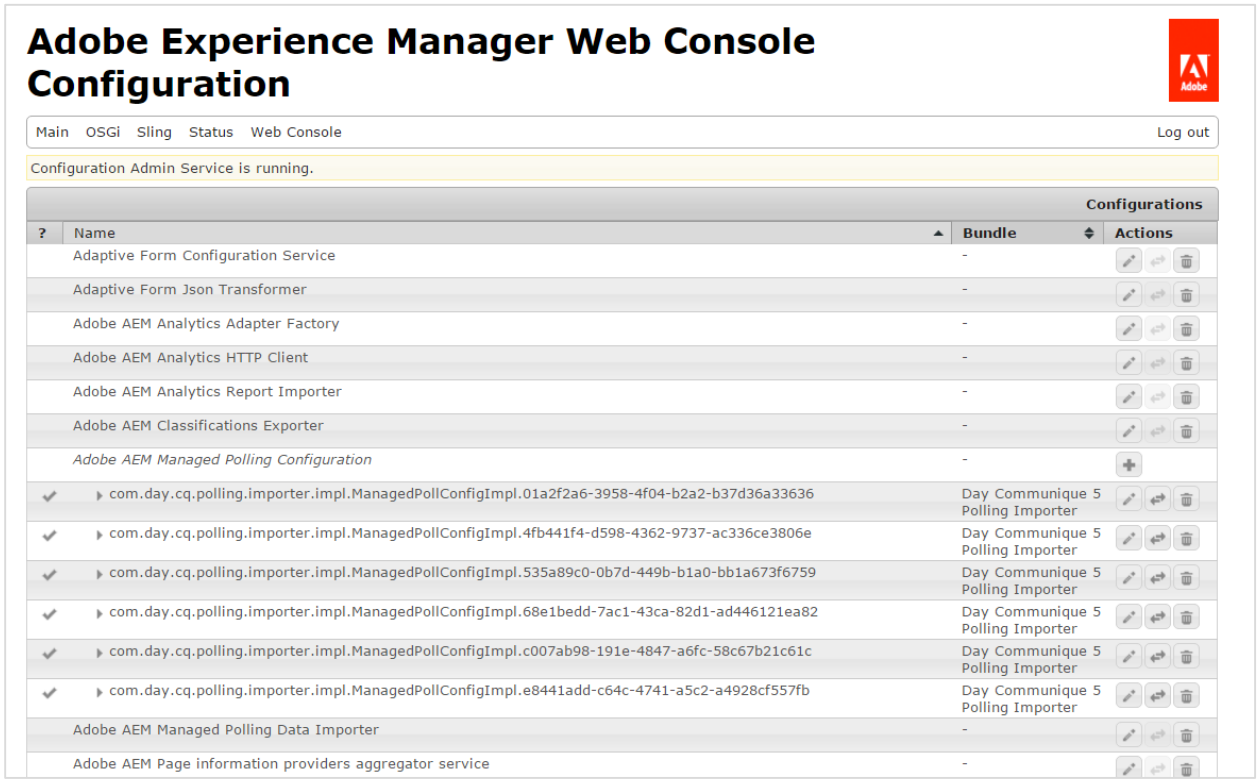
4.1 The Web Console

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

You can access the console through this link: http://localhost:4502/system/console

The Web Console has the following major selections under the OSGi tab.

- **Bundles**—used for installing and managing bundles.
- **Components**—used for managing and controlling the status of components required for Adobe Experience Manager.
- **Configurations**—used for configuring OSGi bundles, and is the underlying mechanism for configuring Adobe Experience Manager parameters.



4.2 CRXDE Lite

CRXDE Lite is embedded into Adobe Experience Manager and allows you to perform common development and administration tasks in the browser. As it is embedded in the server and is always available, CRXDE Lite is often the

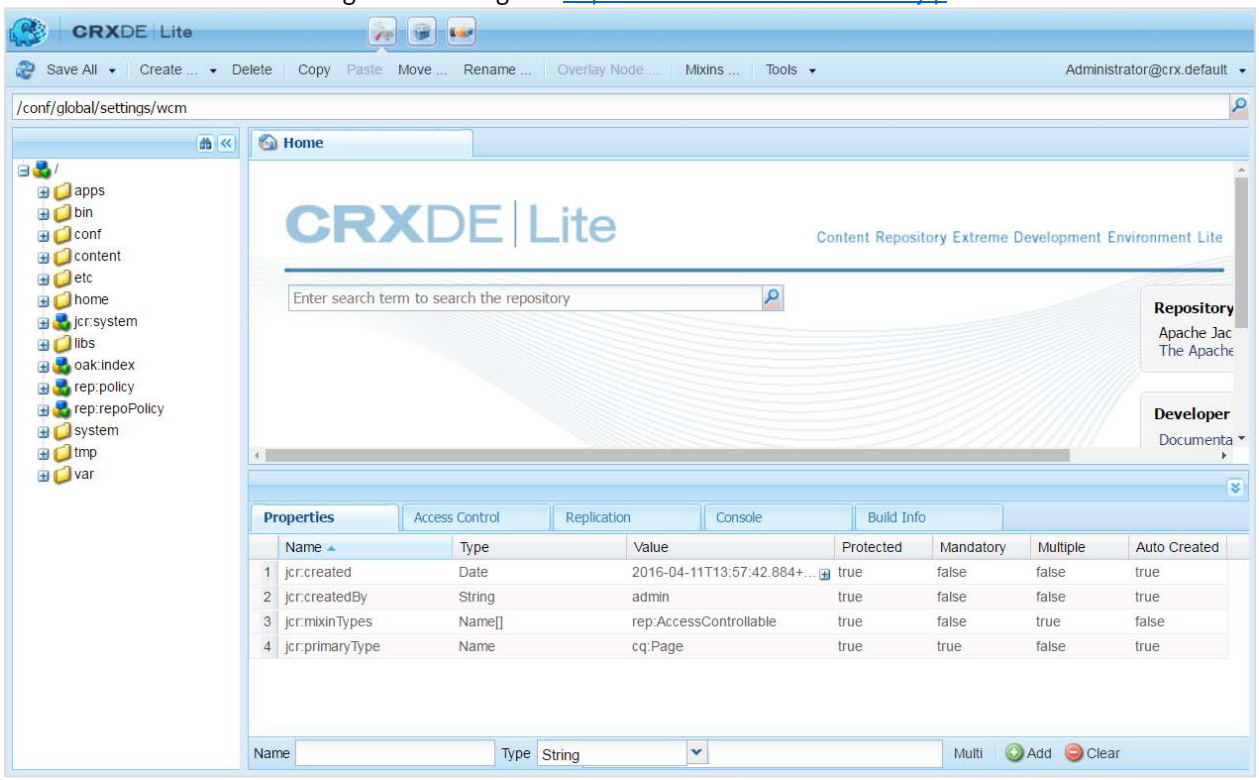
preferred tool for administrators and developers for working with nodes and properties in the JCR/CRX repository. It gives quick and direct access to the repository for observation, configuration, and development.

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

- **Explorer pane**: Displays a tree of all the nodes in the repository. You can perform the following actions on a node in the tree:
  - Select the node and view its properties in the Properties tab. Examine all the JCR properties of different nodes.
  - Right-click the node and perform an action on it, such as renaming the node, creating a new node, creating a folder, creating a file, and so on. You can see some of the common types of folders and nodes that will be used in the training class.
- **Edit pane**: Double-click a file in the Explorer pane to display its content.For example, a .jsp or a .html file. You can then modify it and save the changes.
- **Properties tab**: Displays the properties of the node that you selected. You can add new properties or delete existing ones.

NOTE: You will use this interface frequently in the training. It is recommended to use this interface when there is no direct access to the Adobe Experience Manager/CRX server.

You can access this console through the following link: http://localhost:4502/crx/de/index.jsp



It is recommended that you use Eclipse or Brackets for application development. CRXDE Lite can be used as a window to the content repository.

4.3 Working with Packages

Throughout this training, you will be working with packages; either by creating your own or installing an available package. These packages enable you to import and export repository content. A package may contain:

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

You would commonly use packages for any of the following:

- To install new functionality
- To transfer content between instances
- To back up repository content
- To export content to the local file system

You can work with packages in two ways:

- **Package Manager**—can be used to import or export content, transfer content between instances, and back up repository content. Using filters, you can create a package to contain page content, or project-related content. You can access this console through the following link: http://localhost:4502/crx/packmgr/index.jsp. With the Package Manager, you can perform the following common tasks:
  - Create, build, and download content packages
  - Upload and install packages
  - Modify existing packages
  - View package information
- **Package Share**—a centralized server that contains both public and private packages available across all instances. Public packages may include hotfixes, new functionality, documentation, and so on.

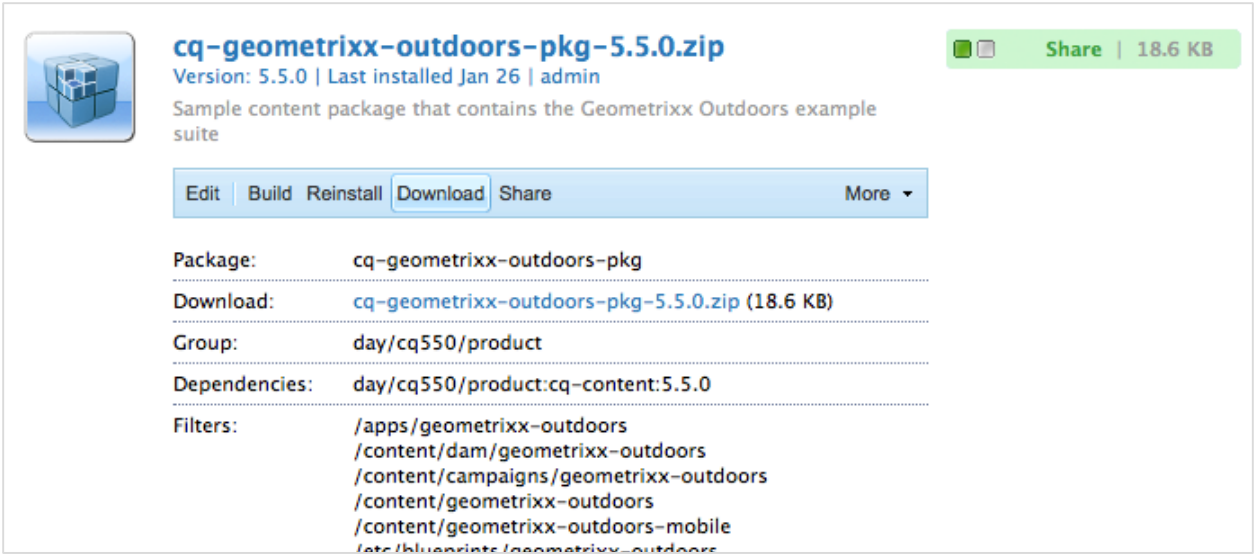
4.3.1 Creating and Building New Packages

Packages are created by applying filters and rules that will extract content from the repository. This can be done using the Package Manager available in Adobe Experience Manager. Once you define a package, you can build it. Building a package results in a zip file being created that you can download to your local file system. You can test the contents of the package before building it. There are many options available for a package. Some of them are:

- Rebuild—this option is used if there is a change in the repository content.
- Edit—this option is used to edit filters or rules applied to the package
- Test—this option is used to perform a dry run of the installation.
- Rewrap—this option is used to recreate the package with additional information such as thumbnails, and icons. An example of when this is required is when you might have to share this package on Package Share.

4.3.2 Downloading packages to your file system

Packages are in the form of a zip file that you can download to your file system. To download a package, select the package from the list and click the download link available when the package details are expanded. Once downloaded, you can unzip them to retrieve the contents of the package.



Typically, a content package would contain the following folders:

- **jcr_root:** This folder represents the root node of the repository, and contains the actual content of the package.
- **META-INF**: This folder contains metadata regarding node definitions and also contains the filter.xml file, which gives directions to FileVault about which paths to include.

4.3.3 Using the Package Share

Package Share is a centralized server where public and private packages are made available. These packages may be hotfixes, feature sets, updates, or Adobe Experience Manager content generated by other users. You can search, download, and install any package either to your instance or to your local file system.

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

- Adobe packages provided by Adobe
- Shared packages provided by others companies and made public by Adobe
- Your company packages that are private

You can access the Package Share with this link: http://localhost:4502/crx/packageshare/index.html

NOTE:

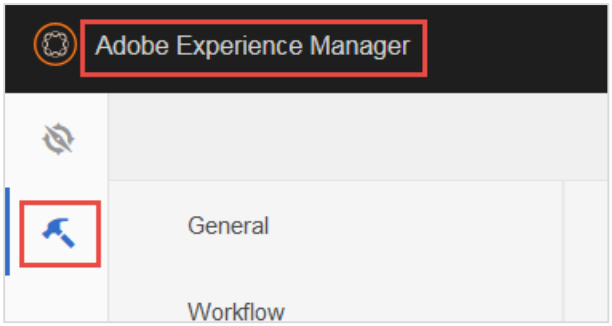
- You need to have an Adobe ID to use the Package Share. For more information, refer to the online documentation on Package Share.
- You can directly access Package Share (without using CRXDE Lite) through the following link: https://www.adobeaemcloud.com/content/packageshare.html

4.4 Lab Activity – I

Task - Install a package in Adobe Experience Manager

1. Navigate to the **Projects** console, or click the following link: http://localhost:4502

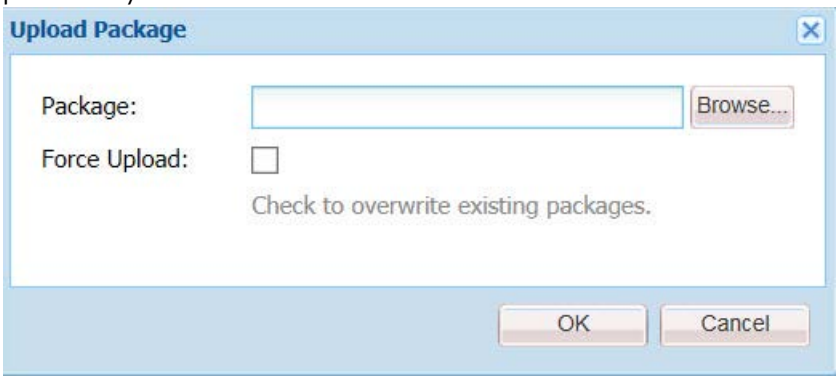
2. Click **Adobe Experience Manager** in the upper left corner of the screen, and then click the **Tools** icon.



3. Under the **Deployment** section, select **Packages**. This opens the Package Manager of CRXDE Lite. As an alternative, click the following link to open the page directly: http://localhost:4502/crx/packmgr/index.jsp

4. Click Upload Package.

5. In the **Upload Package** dialog box, click **Browse**, and select the **SamplePackage.zip** package from the task files provided to you. Click **OK**.

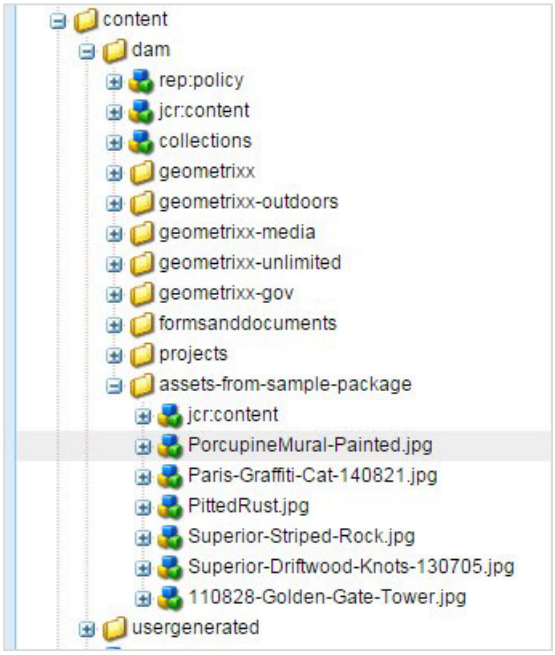


6. After the package is uploaded, click **Install**.

7. In the **Install Package** dialog box, ignore the Advanced Settings area and click **Install**.

8. Check the **Activity Log**. You can see the content that was added from the package.

9. Check the **/content/dam** folder in CRXDE Lite to see the changes reflected there. The contents of the package are added to the repository.

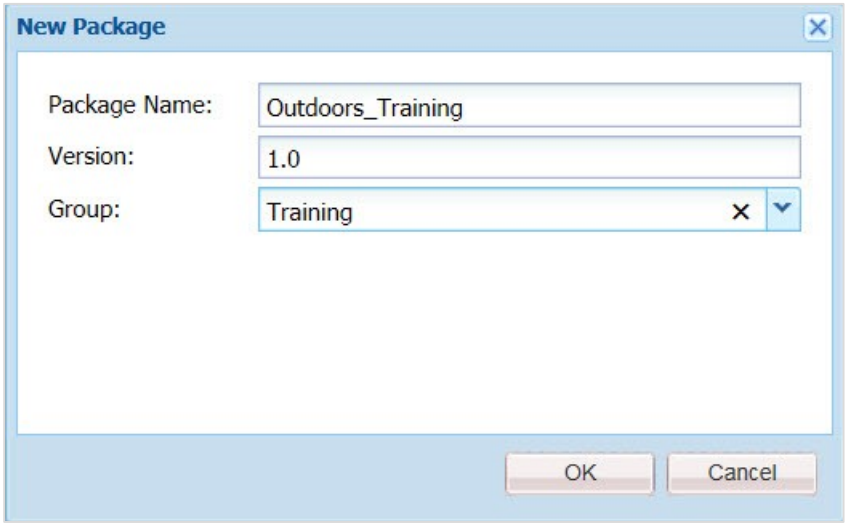


Task - Create, Build, and Download a Package

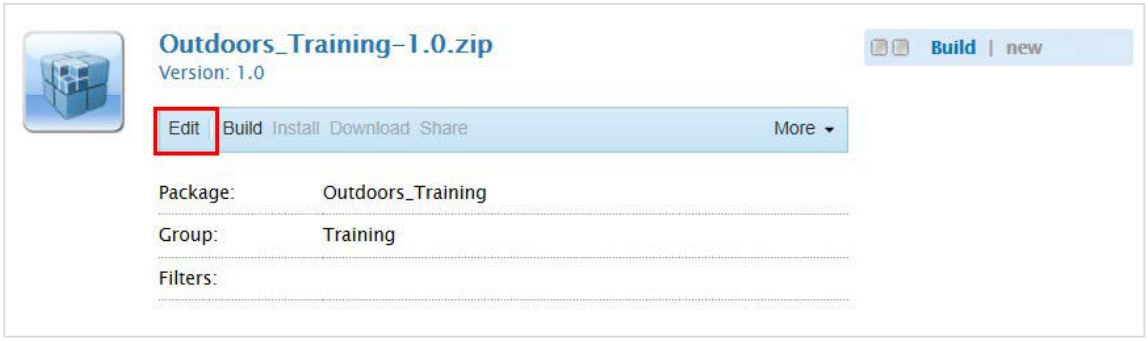
1. Navigate to the **Projects** console or click the following link: http://localhost:4502
2. Click **Adobe Experience Manager** in the upper right corner of the screen, and then click the **Tools** icon.
3. Under the **Deployment** section, select **Packages**. This opens the Package Manager of CRXDE Lite. As an alternative, click the following link to open the page directly: http://localhost:4502/crx/packmgr/index.jsp
4. Click Create Package.

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

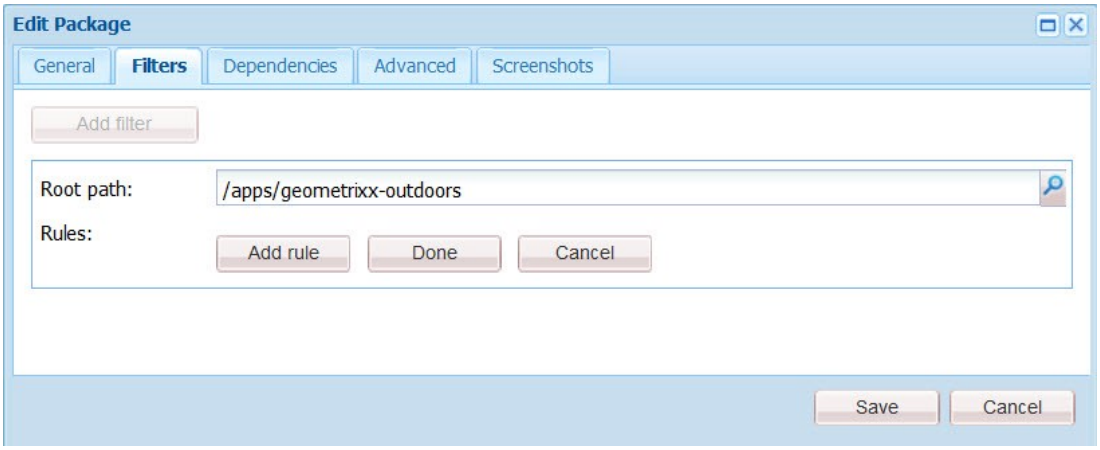
Property	Value
Package Name	Outdoors_Training
Version	1.0
Group	Training



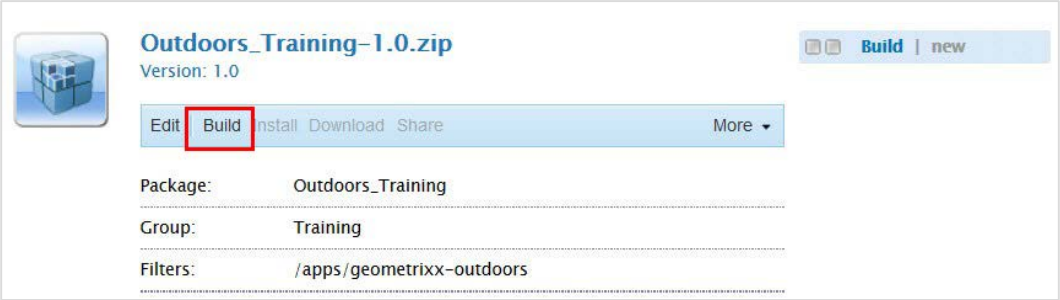
6. Click **OK**.
7. Click **Edit** on the newly created package.



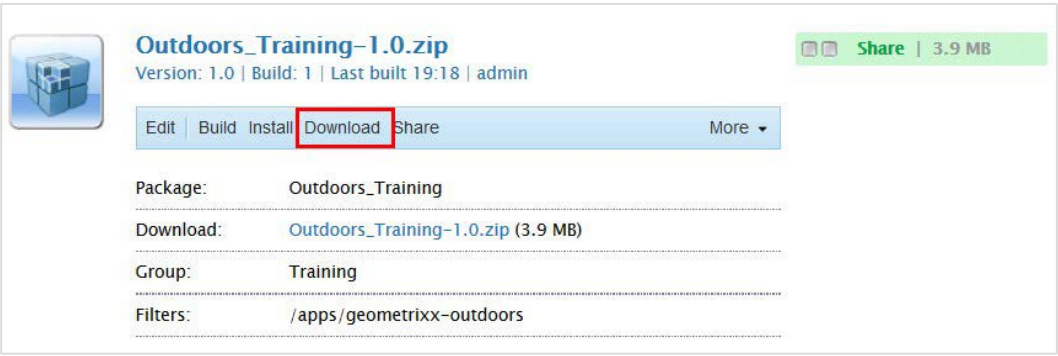
8. To add filters to the package, click the **Filters** tab, and click **Add Filter**.
9. For the **Root** path, browse and select the **geometrixx-outdoors** project under **apps**, and click **OK.**
10. Click **Done**, and then click **Save**.



11. Click **Build** to build the package, and then click **Build** again in the confirmation dialog box.



12. The package is now ready for download. Click the **Download** link to download the package.



4.5 Working with Brackets

Brackets is developed by Adobe, and is an open source code editor for HTML, CSS, and JavaScript. You can download the latest version of Brackets from: http://brackets.io/

Features of Brackets:

- Live preview
- In-line editors
- Pre-processor support
- A collection of add-on extensions

Available Plugins for Adobe Experience Manager Developers:

- AEM Brackets Extension
- Extract for Brackets

4.5.1 Installing the AEM Brackets Extension

You can install the AEM Brackets Extension within Brackets, using the Extension Manager. Brackets with the HTL extension is the preferred tool for front-end developers building components with HTL.

Features of AEM Brackets Extension:

- Automatic, bidirectional synchronization with the content repository
- Full content-package synchronization of the project.
- HTL syntax highlighting
- HTL code completion

HTL is a template language developed by Adobe, and was introduced with Adobe Experience Manager 6.0. It simplifies component development, making it more secure.

For more information on this extension, refer to https://docs.adobe.com/content/docs/en/dev-tools/aem-brackets.html

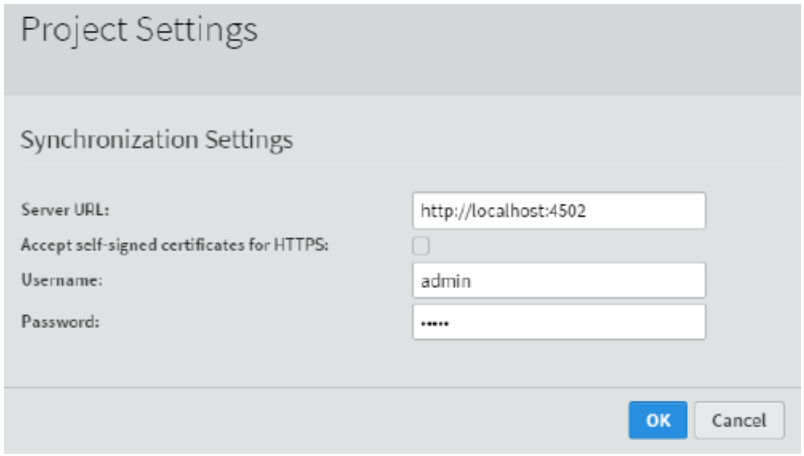
4.5.2 Configuring Your Project

You have already learned how to create a package using the Package Manager. You can import the same package to Brackets and begin working on it, or if you already have a project package, you need to ensure that it has the following:

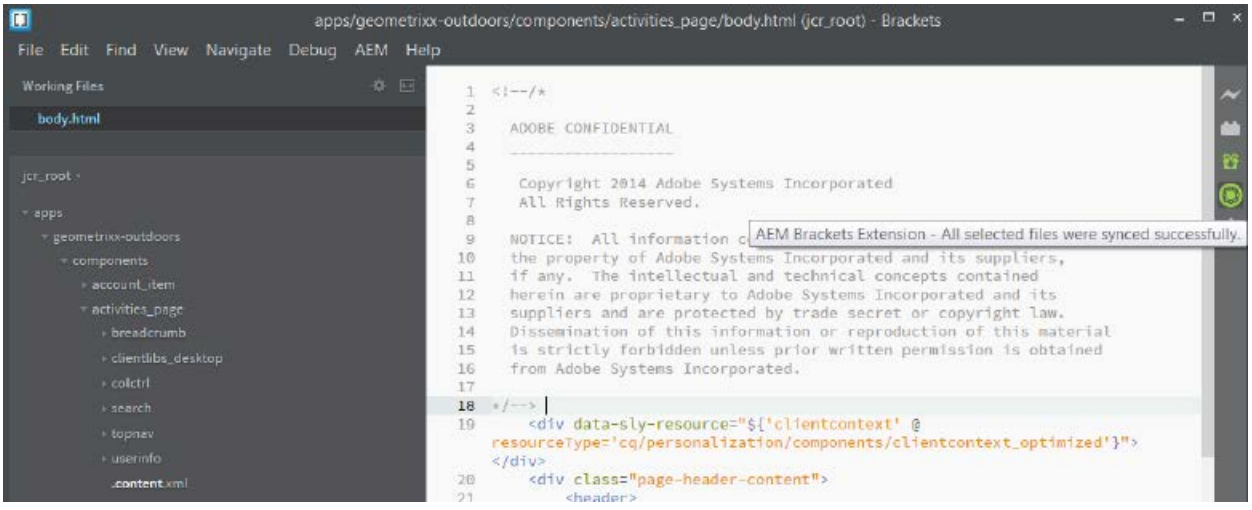
- A **jcr_root** folder
- A filter.xml file

Here is how you will work on a project:

- If the package is still in a zipped format, unzip it and extract its contents.
- Open the **jcr_root** folder of the project package.
- Configure the project settings to connect to the Adobe Experience Manager server.



Your project is now ready for development. Anytime you save your changes, it will be automatically synchronized with the Adobe Experience Manager content repository. This is indicated by a green icon in the toolbar on the right pane.



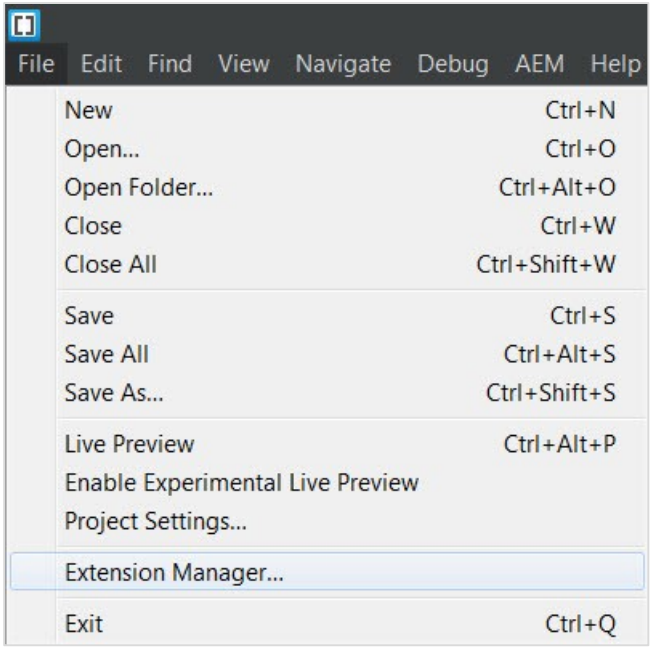
Either you can use the Live Preview to review your changes, or you can do it through the Adobe Experience Manager browser instance.

4.6 Lab Activity – II [Optional]

Task - Install AEM Bracket Extension

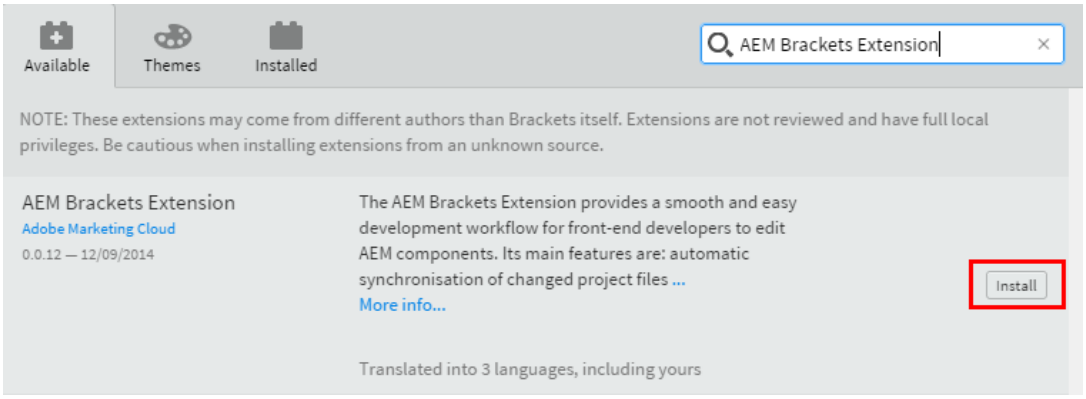
You can install Brackets with or without an internet connection.

1. If you have an Internet connection, download the latest version of Brackets from: http://brackets.io/
   OR
   Obtain the installation files from the USB contents.
2. Install Brackets and open it.
3. Click File > Extension Manager...



4. On the **Available** tab, search for **AEM Brackets Extension**. You can also drag the extension from the USB folder to the area displayed at the bottom of the pop-up window.

5. Click **Install**.



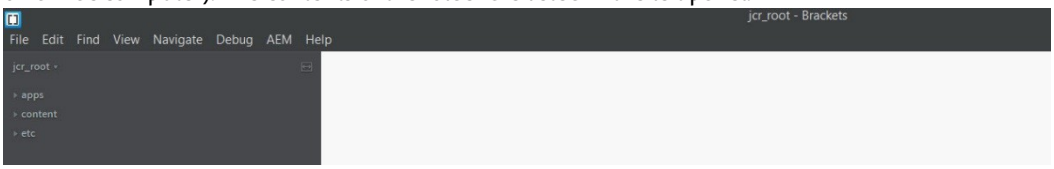
6. After a successful installation, a success message appears on the screen.
7. Restart Brackets.

Task – Make changes to the repository using Brackets

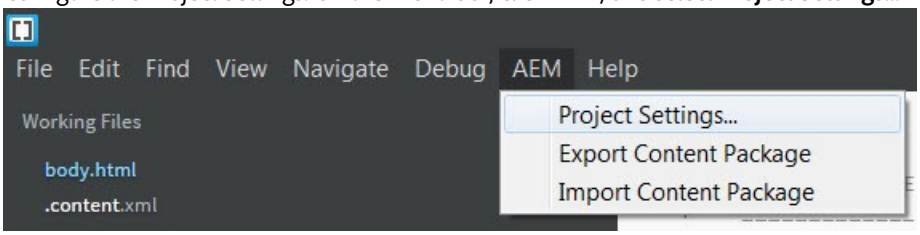
The goal of this task is to edit the contents of the Geometrixx Outdoors project through Brackets. After completing this task, you can update the package content and share it with the team members so that they also see the same content in their Adobe Experience Manager instance.

You have already created and downloaded a package in the previous task. You will be configuring and editing this package in Brackets. You will see the changes saved by browsing the updated content in web browser.

1. Navigate to the downloaded zipped package, and unzip the file.
2. Open Brackets.
3. Navigate to **File** > **Open Folder...**
4. Locate the **jcr_root** folder of the unzipped Geometrixx-Outdoors package, and click **Select Folder** (or click Open on a Mac computer). The contents of the folder are listed in the left panel.

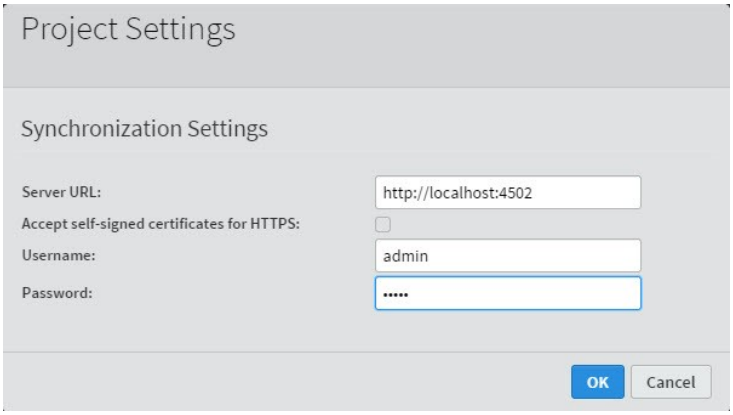


5. Configure the Project Settings. On the menu bar, click **AEM**, and select **Project Settings...**



6. In the **Project Settings** dialog box, enter the following details, and click **OK**.

Label	Value
ServerURL	http://localhost:4502
Username	admin
Password	admin



7. These details are provided to Brackets, so that it can interact with Adobe Experience Manager instance. In a real-life scenario, your URL and credentials might differ.

8. In the left panel, expand the apps node and navigate to **apps/geometrixx-outdoors/components/activities_page/body.html**.

9. When you select **body.html**, the file is opened in the right panel. Add the following code in the file at line 33.

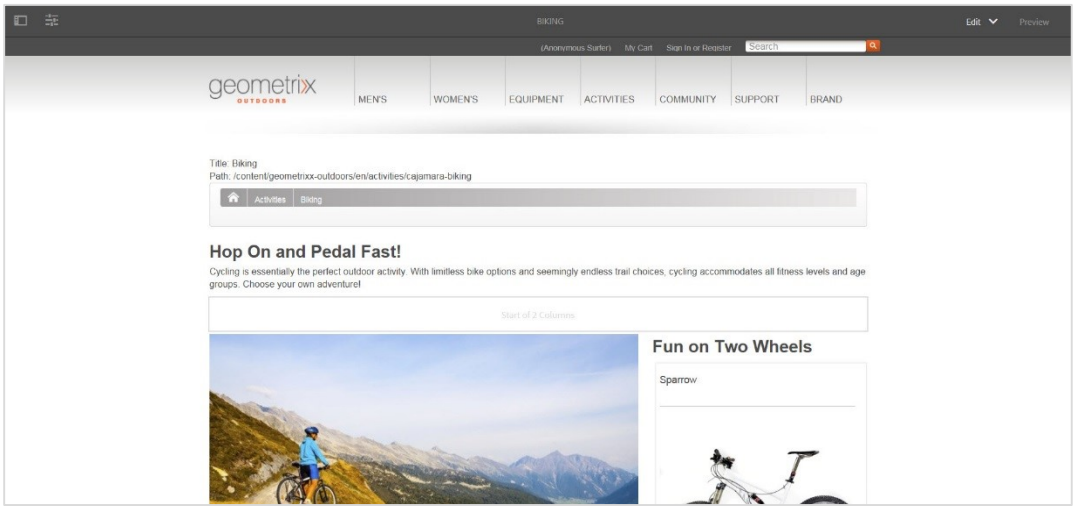
```
Title: ${currentPage.title} <br>
Path: ${currentPage.path}
```

The inserted code is shown in the figure below.

```
resourceType='cq/personalization/components/clientcontext_optimized'}"></div>
    <div class="page-header-content">
        <header>
            <div class="page-systemnav">
                <div class="sys-nav-wrapper">
                    <div data-sly-resource="${'search' @ resourceType='geometrixx-outdoors/components/activities_page/search'}" class="search"></div>
                    <div data-sly-resource="${'userinfo' @ resourceType='geometrixx-outdoors/components/activities_page/userinfo'}" class="userinfo"></div>
                </div>
            </div>
            <div data-sly-resource="${'topnav' @ resourceType='geometrixx-outdoors/components/activities_page/topnav'}" class="topnav"></div>
        </header>
    </div>
    <div id="main" class="page-main">
        <div class="page-content">
        Title: ${currentPage.title}<br>
        Path: ${currentPage.path}
            <div data-sly-resource="${'breadcrumb' @ resourceType='geometrixx-outdoors/components/activities_page/breadcrumb'}" class="breadcrumb"></div>
            <div data-sly-resource="${'par' @ resourceType='wcm/foundation/components/parsys'}"></div>
        </div>
        <div class="page-footer" sly-use.footer="footer.js">
            <footer>
                <nav>
                    <div data-sly-resource="${'toolbar' @ resourceType='foundation/components/toolbar'}" class="toolbar"></div>
                </nav>
                <p class="copyright">&copy; ${"{0} Geometrixx Outdoors. All rights reserved." @ i18n, format=[footer.currentYear]}</p>
```

10. Save your changes. This would automatically synchronize the file with Adobe Experience Manager. To test the synchronization, open the Bikers page in Geometrixx Activities. Alternatively, click the following link: http://localhost:4502/editor.html/content/geometrixx-outdoors/en/activities/cajamara-biking.html

    You will see the changes reflected in the page.

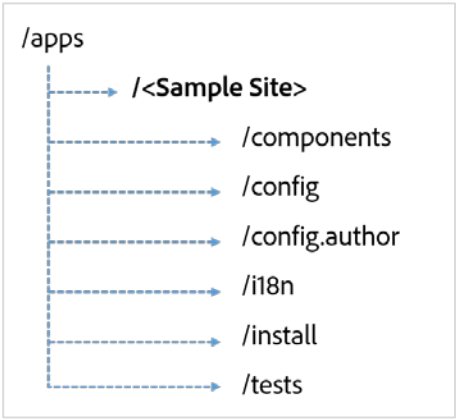


4.7 Best Practices in Development

Adobe has come up with a new reference site called we.Retail that replaces the Geometrixx sites. This new site is built using the Maven Archetype 10. This is the latest archetype available by Adobe and makes use of all the development best practices.

What is a Maven Archetype?
Archetype is a Maven project templating toolkit. It is defined as an original pattern or model from which all other things of the same kind are made.

When a project is created using the Archetype 10, it ideally creates the following folder structure in CRXDE Lite:



- **components** – this folder would contain all custom components such as title component, navigation component, dialog boxes, and so on.
- **config** – this folder would contain all new and custom configuration files.
- **config.author**—this folder contains all configuration files for the author runmode.
- **templates** – this folder would contain all new and custom page templates.
- **src** – this folder would contain all source codes for creating bundles.
- **install** – this folder would contain all compiled OSGi bundles.

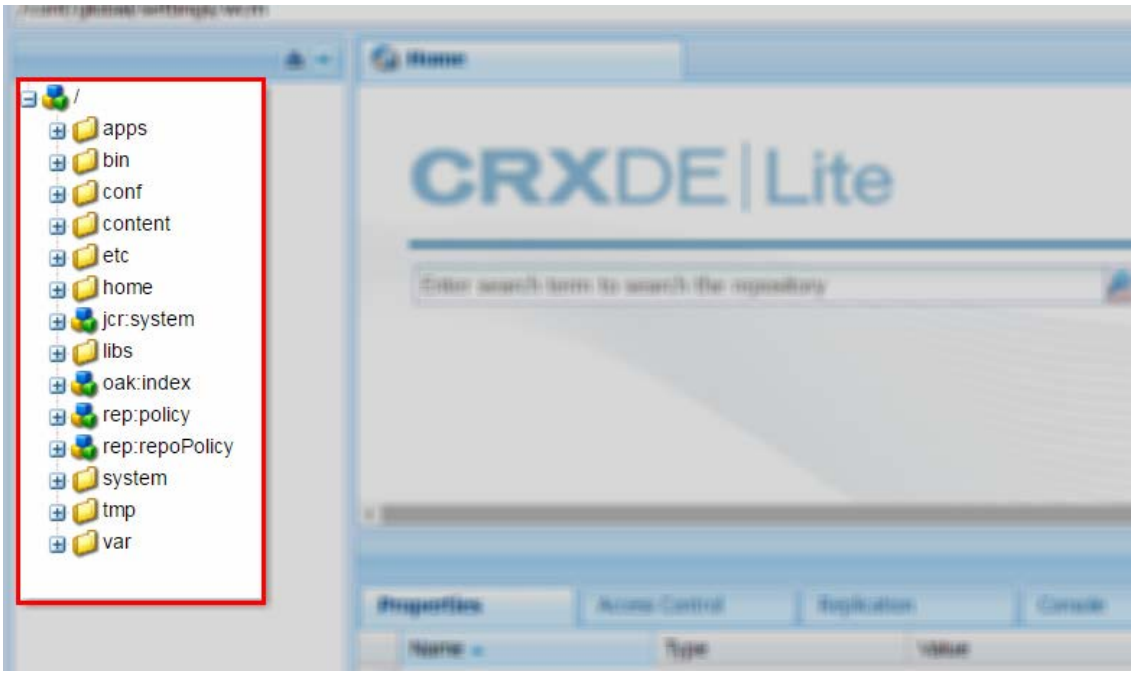
With this course, you would create a site that resembles the we.Retail site and thereby adheres to the best practices.

5 Introduction to Content Rendering

5.1 Folder Structure of the Repository

You have already seen how a project created using the Archetype 10 is structured. We will create a similar structure for our site. It is good practice to set up a specific folder structure that clearly defines and organizes the various elements of your site into its respective folders. These elements include your templates, components, OSGi bundles, and static files. You should create all custom projects under the /apps folder in CRXDE Lite.

Before we begin building our site, let's have a quick look at the folder structure in CRXDE Lite.



Here's what each of the folders contain:

- **/apps**—all custom templates, components, and any other definitions related to your site are stored here. When you inherit foundation components, it is done from the libs folder. Always copy the components from the libs folder to the apps folder, and then modify it in the apps folder. By doing this, no code will be lost when you upgrade Adobe Experience Manager. You just need to update the apps folder.

  Best practice is to use a feature known as Sling Resource Merger, and just duplicate the required /libs folder structure (empty folders) under /apps. Make modifications to the node or property, wherever the changes are required.

- **/libs**—all libraries and definitions that belong to Adobe Experience Manager code are stored here. It includes out-of-the-box components, templates, and other Adobe Experience Manager features such as search or replication. It is also referred to as the foundation components. **Avoid making any modifications to any of the components in libs**.

- **/content**—all content of your website is stored in this folder.

- **/conf**—contains all configurations for your site. Since Adobe Experience Manager 6.2, this folder is used to store the dynamic templates and policies for your site.

- **/etc**—contains all resources related to utilities and tools.
- **/home**—contains all information related to users and groups.
- **/tmp**—a temporary working area.
- **/var**—contains files that change and are updated by the system, such as audit logs, statistics, and event handling.
- **/oak:index**—contains Jackrabbit Oak index definitions. Each node specifies the details of one index. Standard indexes for the AEM application are visible and additional custom indexes can be created.

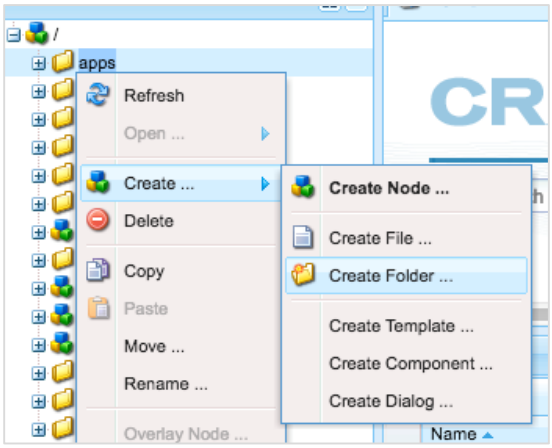
5.2 Lab Activity - I

In one of the earlier chapters, we touched on Apache Sling, the web framework. Now let's go a bit deeper and see how AEM uses Apache Sling to resolve a resource and then map that resource to a rendering script.

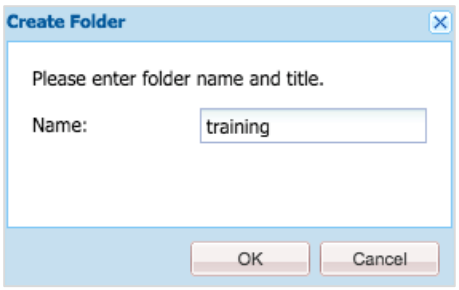
Task – Create the project structure in /apps

1. Navigate to CRXDE Lite. http://localhost:4502/crx/de. Or from any console, type "**/**" and enter "**crx**" into the OmniSearch toolbar and click on **CRXDE Lite**.

2. Right-click on **/apps** and select **Create…**> **Create Folder**.

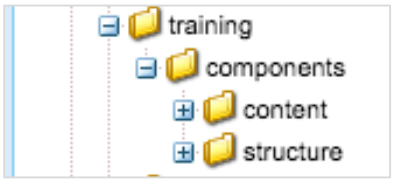


3. Enter **training** as the name of the folder. Click **OK** and then click **Save All** in the upper-left.

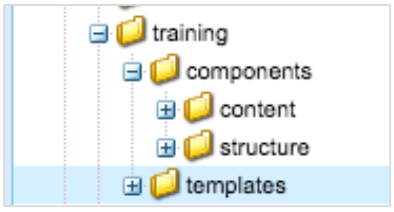


Note: Node names are a restricted set. No spaces or special characters. Also, do not use "_" (underscore). By convention, node names begin with a lowercase letter.

4. Right-click on **/apps/training** and select **Create…**> **Create Folder**. Enter "**components**" as the folder name.
5. Save your changes.
6. Now use what you have learned to create two child folders of **/apps/training/components**, named "**content**" and "**structure**". Save your changes.

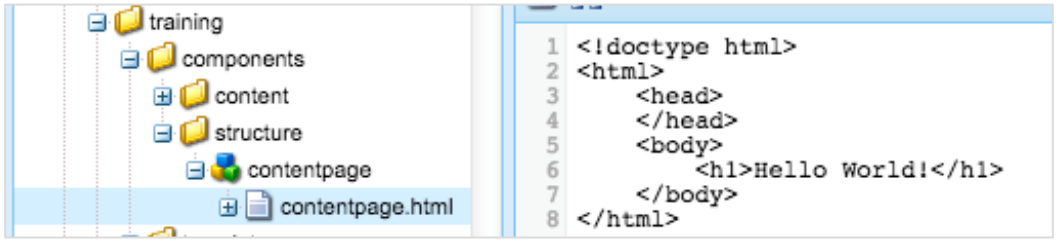


7. To finish out our project structure, right-click on **/apps/training** and create a folder named **templates**. Save your changes.



5.3 Page Rendering Components

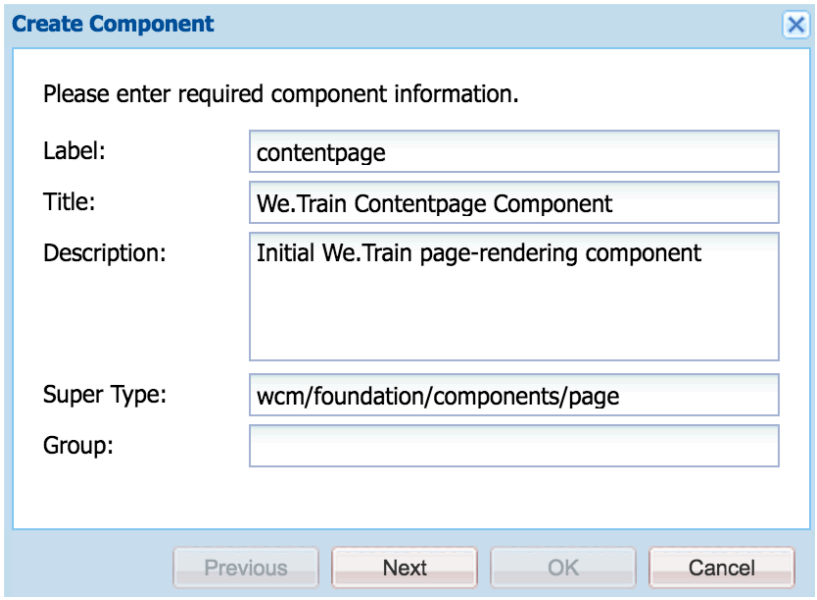
- Components are modular, reusable units that implement specific functionality or logic to render the content of your website.
- A component can be described as a collection of scripts (for example, HTL files, JSPs, Java servlets, and so on) that completely realize a specific function.
- Every component has a default script file that is identical to the name of the component. If needed, you can remove this script and create your own.
- The template sling:resourceType property must match the path to a page rendering component.

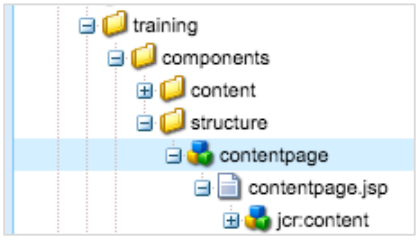


5.4 Lab Activity – II

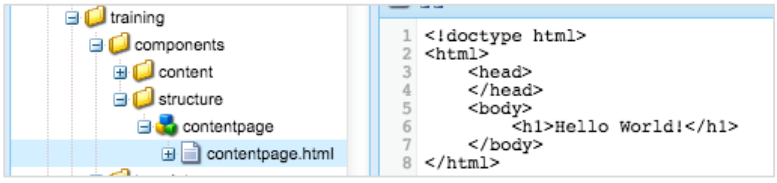
Task – Create a page-rendering component

1. Right-click /apps/training/components/structure and select Create…> Create Component.
2. Fill in the following values:

- **Label**: contentpage
- **Title**: WE.TRAIN Contentpage Component
- **Description**: Initial WE.TRAIN page-rendering Component
- **SuperType**: wcm/foundation/components/page
- 

3. Click **Next**.
4. Click **OK**, and save your changes.
5. 

6. Right-click on contentpage.jsp to rename to contentpage.html.
7. Double-click on **contentpage.html** to open the script for editing. Replace all sample code and comments with the code from the USB Contents.



8. Save your changes.

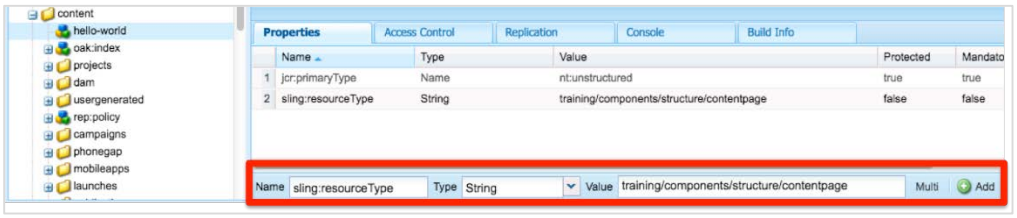
Task – Create content to be rendered

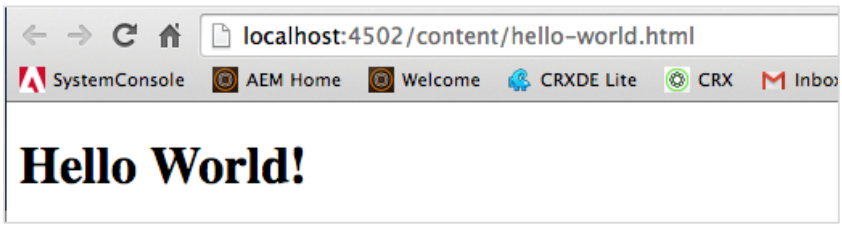
Now that we have a component that will render content, we need some actual content to render.

1. Right-click on **/content** and select **Create…**> **Create Node**.
2. Create a node named "**hello-world**" of type "**nt:unstructured**".
3. Save your changes.
4. Add a property with the following attributes, and click **Add**.

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

5. Notice that the sling:resourceType property value is the path to the page-rendering component that we just defined.



6. **Pro Tip:** By convention, property names are camel case, excluding the namespace prefix. Camel case means that the first word in the property name starts with a lowercase letter and each additional word in the property name starts with an uppercase letter.
7. Save your changes.
8. Open http://localhost:4502/content/hello-world.html in the browser.
9. 

What was the sequence of events that caused the Hello World text to display?

5.5 Understanding the Sling Resolution Process

- Sling is resource oriented.
- All resources are maintained in the form of a virtual tree.
- A resource is usually mapped to a JCR node, but can also be mapped to a file system or database.
- Common properties that a resource can have: Path, Name, Resource Type

URL ➔ Resource = JCR Structure

5.6 Basic Steps of Processing Requests

- Each content item in the JCR repository is exposed as an HTTP resource.
- After the content is determined, the script or servlet to be used to handle the request is determined through:
  - Properties of the content item itself
  - The HTTP method used to make the request
  - A simple naming convention within the URL that provides secondary information
- For every URL request, the following steps are performed to get it resolved:
  - Decompose the URL
  - Search for a node indicated by the URL
  - Resolve the Resource
  - Resolve the rendering script/servlet
  - Create rendering chain
  - Invoke rendering chain

5.7 Decomposing the URL

Consider the following example:
URL : `http://myhost/tools/spy.printable.a4.html/a/b?x=12`

The above URL can be decomposed into the following components:

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

Where,

- **Protocol:** Hypertext transfer protocol (HTTP)
- **Host:** Name of the website
- **Content path:** Path specifying the content to be rendered.
- **Selector(s):** Used for alternative methods of rendering the content
- **Extension:** Content format; also specifies the script to be used for rendering.
- **Suffix:** Can be used to specify additional information
- **Param(s):** Any parameters required for dynamic content

5.8 Resolving Requests to Resources

Let's take a look at a couple of examples:

Example 1:
URL : `http://myhost/tools/spy.html`

Steps performed to map request:

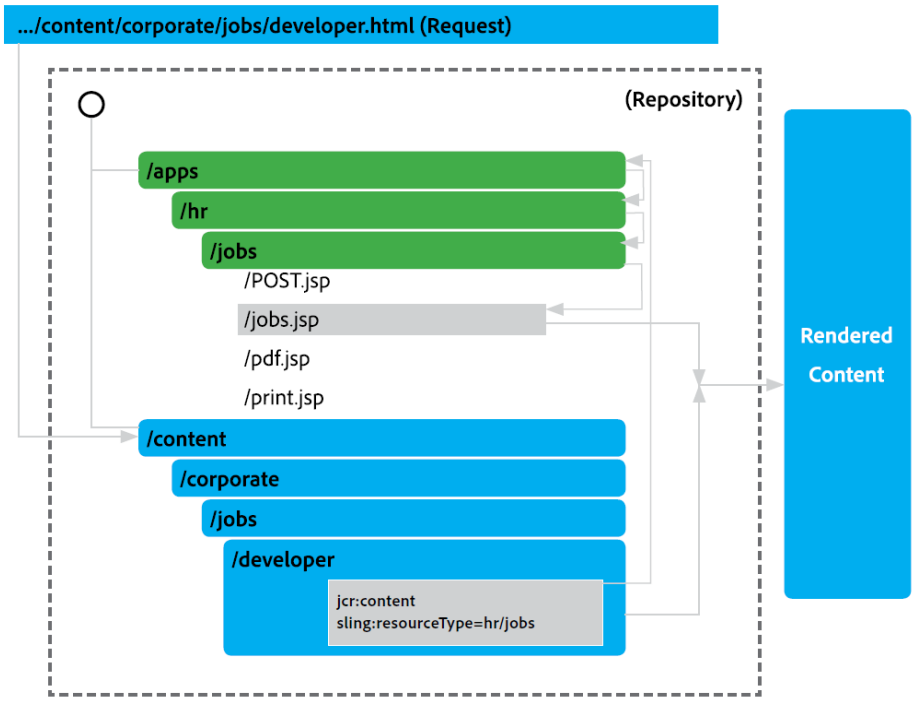
1. Sling searches for the node spy.html
2. If no node is found at that location, the extension is dropped and the search is repeated. For example, spy.
3. If no node is found, then Sling will return the http code 404 (Not Found). NOTE: Sometimes, Sling can resolve to a servlet.
4. If a node is found, then the sling resource type for that node is extracted, and used to locate the script to be used for rendering the content.

Example 2:

Once the URL is decomposed, the content node is located from the content path. This node is identified as the resource, and performs the following steps to map to the request.

Consider the URL request: http://myhost/content/corporate/jobs/developer.html

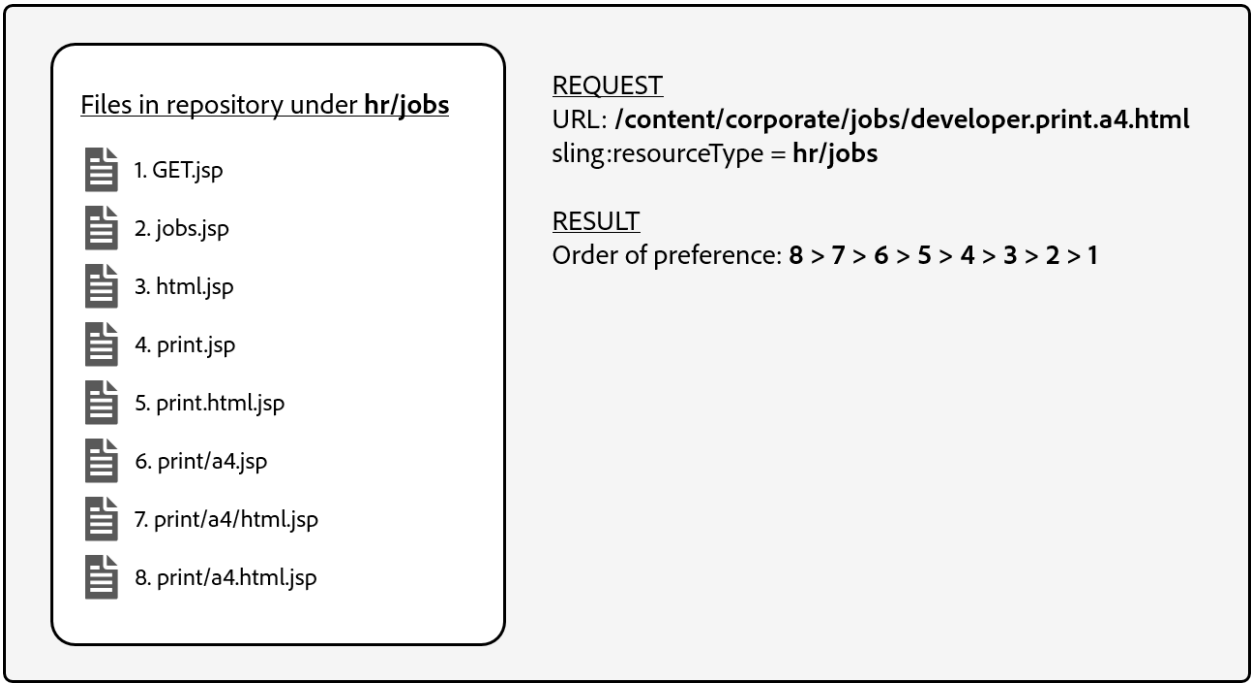
1. Sling checks whether a node exists at the location specified in the request. In the above example, it searches for the node developer.html.
2. If no node is found at that location, the extension is dropped and the search is repeated. For example, it searches for the node developer.
3. If no node is found, then Sling will return the http code 404 (Not Found).
4. If a node is found, then the sling resource type for that node is extracted, and used to locate the script to be used for rendering the content.



5.9 Locating and Rendering Scripts

When the resource is identified from the URL, its resource type property is located and the value extracted. This value is either an absolute or a relative path that points to the location of the script to be used for rendering the content. All scripts are stored in either the **/apps** or **/libs** folder, and are searched in the same order. If no matching script is found in either of the folders, then the default script is rendered.

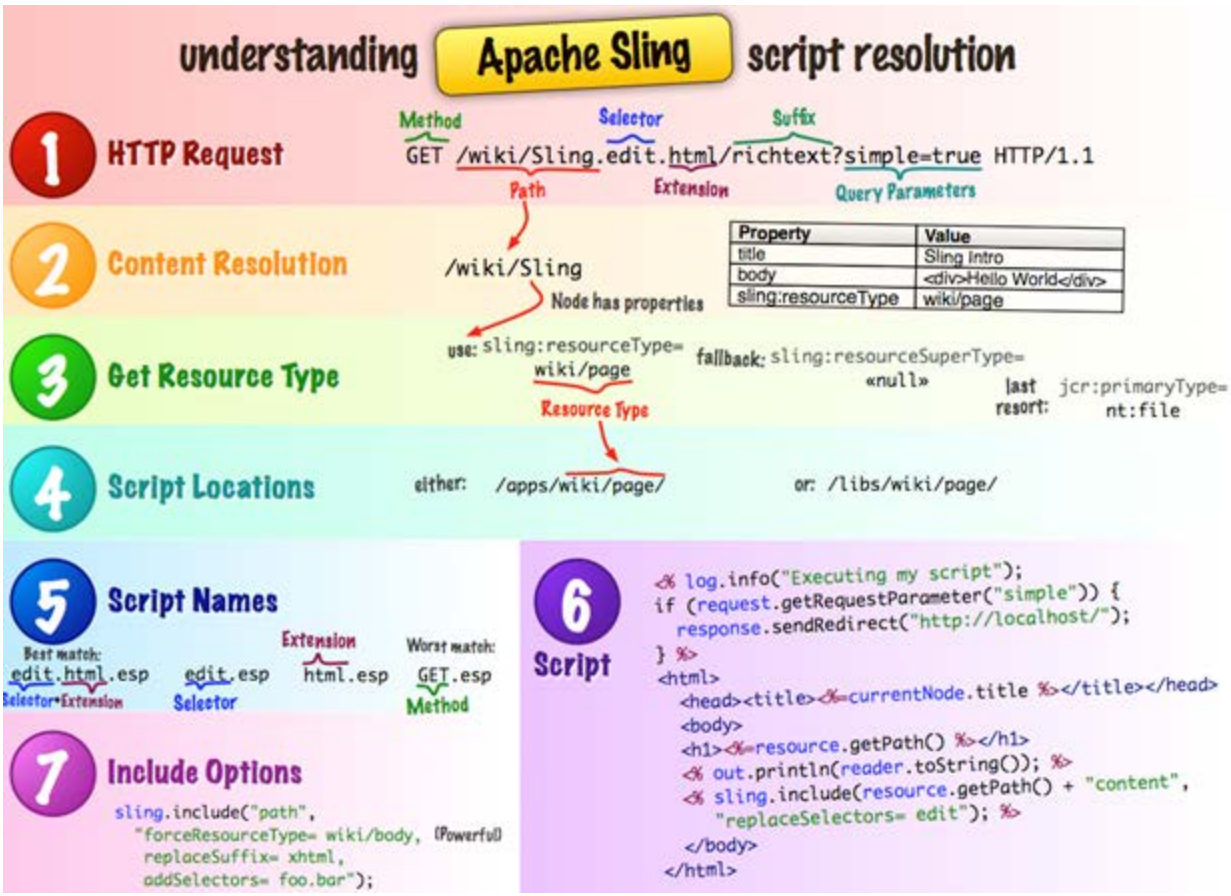
For multiple matches of the script, the script name with the best match is selected. The more selector matches, the better.



5.10 Understanding URL Decomposition

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

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



5.11 Lab Activity – III

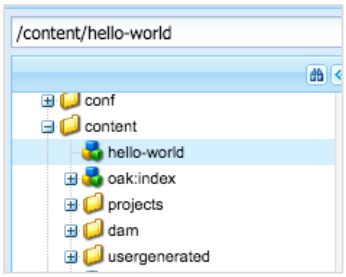
Task – Basic Sling resource resolution and hunt for a rendering script

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

1. Strip off the protocol, server, and port.
2. Strip off any suffix.
3. Examine the remaining portion of the URL and use the information contained therein to assist with resolving the resource and finding its associated rendering script.

Let us follow Sling's actions to see how our hello-world resource was rendered.

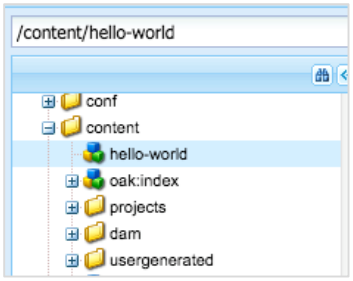
1. Consider the request (URL) http://localhost:4502/content/hello-world.html
2. Disregard http://localhost:4502.
3. In CRXDE Lite, navigate to **/content/hello-world.html**. Notice that path does not exist in the repository.

4. 

5. At this point, Sling will back off the end of the remaining part of the request until the first ".". Everything after the "." is considered the extension. Now Sling will attempt to locate "/content/hello-world".

6. 

7. In CRXDE Lite, navigate to **/content/hello-world**. Yay! We found a node in the repository that matches the resource in the request. What we just did is called "resolving the resource". We successfully resolved a request URL to a resource that is represented by a node in the repository.

8. 

9. **Note:** resources could also map to items in relational databases and/or file system. The Apache Sling specification does not mandate a JCR database.

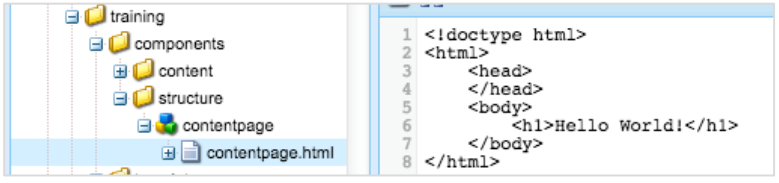
10. Now we need to find the rendering script. On the **/content/hello-world** node, find the **sling:resourceType** property. The value of the sling:resourceType property is what Sling uses to begin its search for a rendering script.

11. Notice the sling:resourceType property points to the page-rendering script that we created previously.

12. 

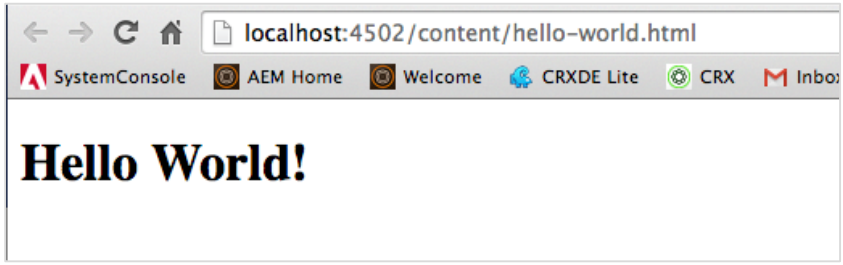
13. Navigate to: /apps/training/components/structure/contentpage

14. Notice there is one script: contentpage.html. This is called the default script because the matches the name of the folder/component in which the script resides. Later we will explore the many options that Sling might use in selecting the "right" script. But right now, we have the default script. 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.

15. 

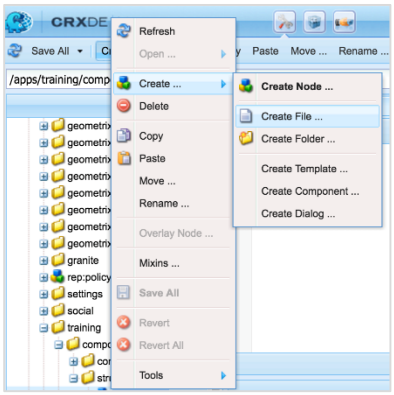
16. The script outputs "Hello World!". We have now successfully resolved the resource, found the rendering script, and invoked the rendering chain.

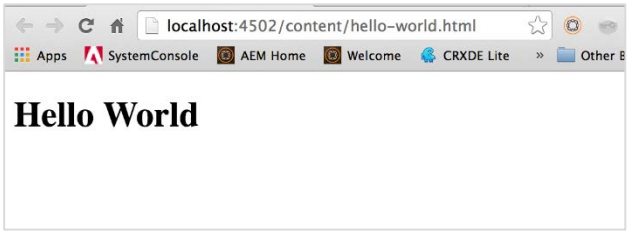
You have just seen how the request "http://localhost:4502/content/hello-world.html" results in the following browser output:

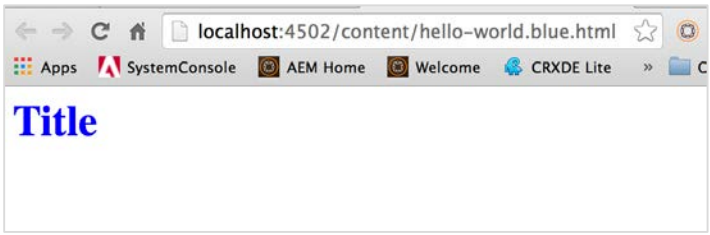


We will continue to learn more about Apache Sling URL decomposition and resource resolution throughout the course.

Task – Selector Manipulation

1. Using CRXDE Lite, right-click on /apps/training/components/structure/contentpage.
2. 

3. Select Create… > Create File.
4. Enter **blue.html** as the file name.
5. Click **OK**. Save your changes.
6. Open the file **blue.html** and enter the following statement. You will find the code in the USB Contents.
7. Save your changes.
8. `<h1 style="color:blue">Title</h1>`
9. To test our selectors, enter the URL that will cause our hello-world node to be rendered: http://localhost:4502/content/hello-world.html.
10. 

11. Now enter http://localhost:4502/content/hello-world.blue.html. Notice the difference. The code from blue.html is chosen, as the script name matches the selector in the URL.
12. 

We will continue to enhance our knowledge of Apache Sling throughout the course.

6 Templates

6.1 What is a template?

A template defines the default components that belong to a page on creation and that cannot be removed by editing the page. Templates also appear in the Create Page dialog so the author can choose a template for the page. They may also be used to front-load content on the page. This template is a hierarchy of nodes that has the same structure as the page to be created, but without any actual content. A template is associated with a page-rendering component. Components use and allow access to widgets, which are used to author or render the content. When you select a template while creating a page, it ensures that the content within the template is copied to the corresponding position in the site tree. This also gives the page its initial content and the property **sling:resourceType**—the path to the 'page' component that is used to render the page.

6.2 Properties of a Template

AEM templates have the following attributes and properties:

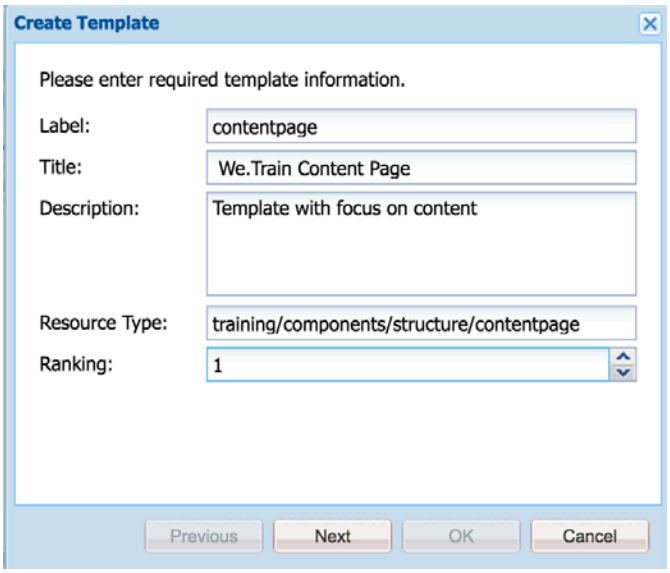
- label – becomes the node name
- sling:resourceType – defines the set of scripts that can be used to render the page
- jcr:title – the string used by the Consoles to display this template
- jcr:description – documentation for this template
- ranking – defines the order in which this template will appear, as compared to other available templates in the "Create" dialog
- allowedPaths – paths where this template can be used
- allowedParents – paths of templates that are allowed to be parent to this template
- allowedChildren – paths of templates that are allowed to be children to this template
- thumbnail.png – a thumbnail image that helps the author identify this template

6.3 Lab Activity - I

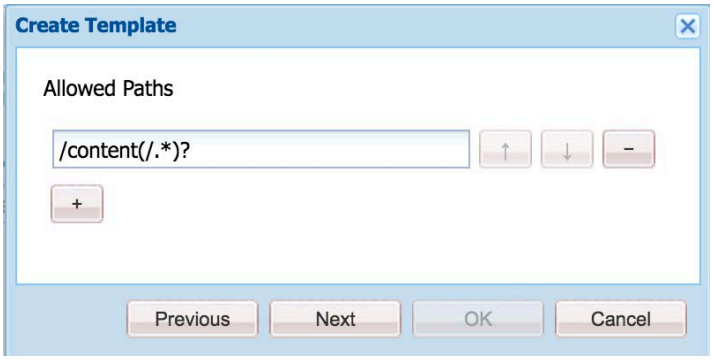
Templates play many parts within the AEM ecosystem. One important part is to be in the list of available templates when the content author creates a page.

Task – Create a Template

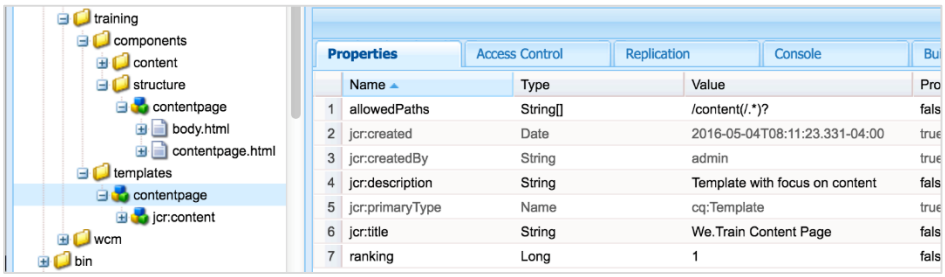
1. Using CRXDE Lite, navigate to **/apps/training/templates**.
2. Right click /apps/training/templates and select Create…> Create Template.
3. Enter the following values:
    - **Label**: contentpage
    - **Title**: We.Train Content Page
    - **Description**: Template with focus on content
    - **Resource Type**: training/components/structure/contentpage
    - Ranking: 1

4. 

5. Click **Next**.
6. Click on the "**+**" and enter "**/content(/.*)?**" for **Allowed Paths**.

7. 

8. Click **Next**.
   Note: Do not click on the "+" for Allowed Parents.
9. Click **OK**.
   Note: Do not click on the "+" for Allowed Children.
10. Save all changes.

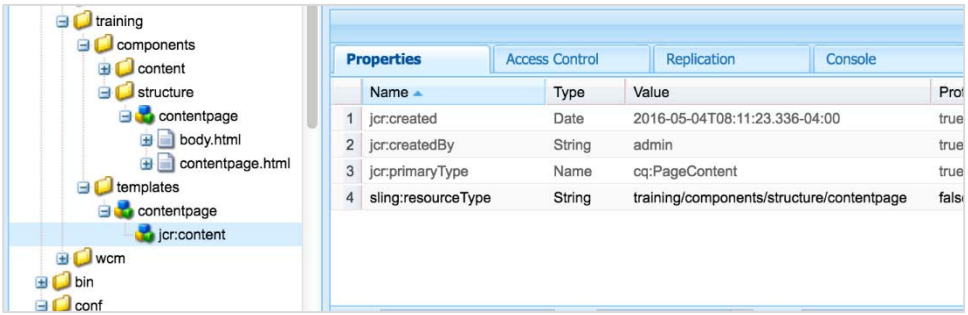


Take a look at the properties that were created on the **/apps/training/templates/contentpage** node:

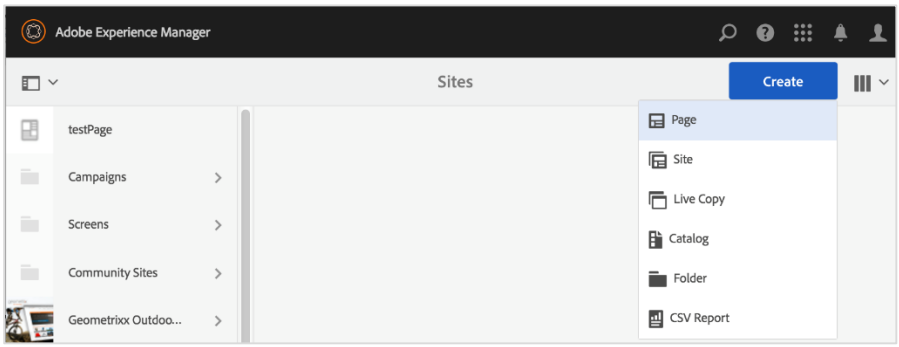
- allowedPaths property specifies where the template can be used to create pages and came from the Allowed Paths pane in the Create Template wizard.

- jcr:description came from the Description entry in the Create Template wizard.
- jcr:title came from the Title entry in the Create Template wizard
- ranking property specifies where in the list of templates this template will appear and came from the Ranking entry in the Create Template wizard. Ours will be first, as we have assigned a ranking of "1".

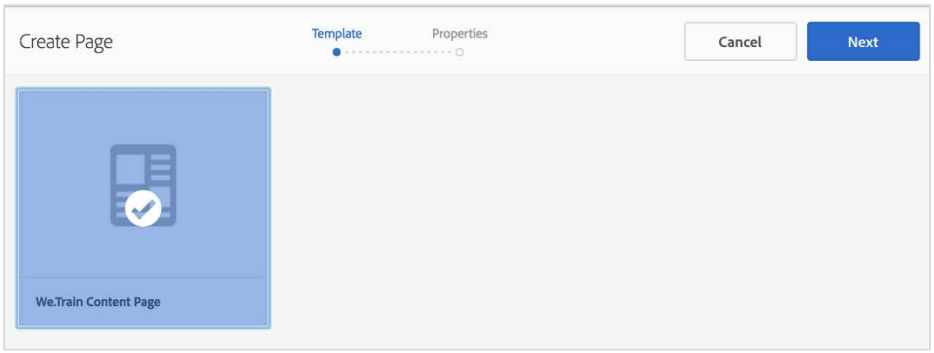
Notice that the value from the Resource Type entry in the Create Template wizard does not appear. This value becomes the sling:resourceType property, which does not appear at this level. We will find the sling:resourceType property on the jcr:content child node of the contentpage template node.



Task – Test the contentpage template

1. Navigate to the Sites Console by entering http://localhost:4502/sites.html or by typing "/", entering sites into the OmniSearch toolbar, and choosing "Go to Sites".
2. Click **Create** and choose **Page**.
3. 

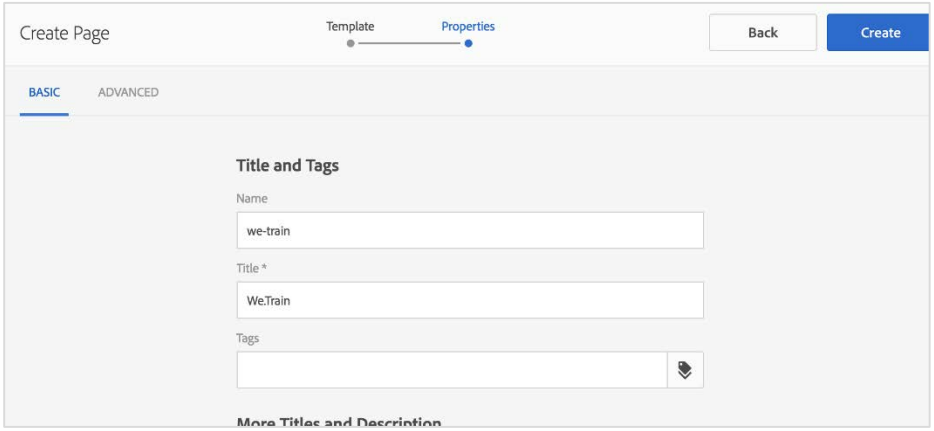
4. Select the **We.Train Content Page** template and click **Next**.

5. 

6. **Troubleshooting tip:** If your template does not appear in the list, check the allowedPaths property on the /apps/training/templates/contentpage template. The value should be: "/content(/.*)?". Any other value will prohibit the template from appearing in the list of available templates.

7. Enter the following properties:

   - **Name**: we-train
   - **Title**: We.Train

8. 

9. Click **Create**.

10. Click **Open**.

11. 

12. **Troubleshooting tip**: If your page is blank, check the sling:resourceType property on the /apps/training/templates/contentpage/jcr:content node. It should exactly match the path to the contentpage page-rendering component. For example: the relative path might be

training/components/structure/contentpage or an absolute path might be /apps/training/components/structure/contentpage.

13. If the sling:resourceType path is incorrect, Apache Sling will not be able to find a rendering script and that is why you are seeing a blank page. Correcting the sling:resourceType property on the template will correct the problem for all new pages. Now you have two choices: 1) you can delete the page you created and recreate it or 2) correct the sling:resourceType property on the jcr:content child of the page you created.
14.
15. **Pro Tip:** When a blank page or blank component occurs, look in the error log, located at <AEM instance root>/crx-quickstart/logs/error.log. You will find an entry for the request that states "… could not find renderer for "<path to the request>. This error message tells you that Sling could not find a rendering script for that request. Carefully examine the path specified in the error message and then look in the repository to fix the problem.

Task – Restrict Template Use

You can restrict where a template can be used in two ways. We have already seen that the allowedPaths property on the /apps/training/templates/contentpage template will define the paths where this template can be used to create pages. This restricts the template from the template's point of view.

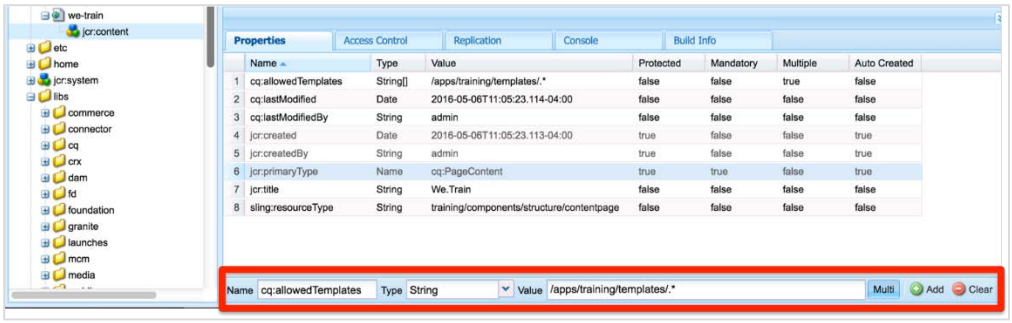
1. Navigate to the AEM Sites Console. Click **Create Page**.
2. Note that the We.Train template appears.
3. Do the same with **Sites** > **Screens**. Note the We.Train template appears.

We probably want the We.Train template to appear as an available template only for the We.Train site.

1. Using CRXDE Lite, navigate to /apps/training/templates/contentpage.
2. Modify the value of the **allowedPaths** property to be: "**/content/we-train(/.*)?**"
3. Repeat steps 1-3. Notice that the We.Train template no longer appears as a choice.
4. Navigate to **We.Train**.
5. Click **Create Page**. Notice that the We.Train Content Page template is available.

The second way to restrict templates is to specify which templates may be used within a specified path. We can accomplish this using the cq:allowedTemplates property on a page.

1. Using CRXDE Lite, navigate to **/content/we-train/jcr:content**.
2. Define the following property:
    a. Name: cq:allowedTemplates
    b. Type: String[]
    c. Value: /apps/training/templates/.*
3. **Tip:** To enter a String Array (String[]), choose String and click the Multi button. Beware! The Multi button stays clicked until you unclick it.

4. 

5. Test out the restriction by going to the AEM Sites Console. Using the same logic as we have used previously in this task, make sure that the We.Train template is still available within the We.Train site.

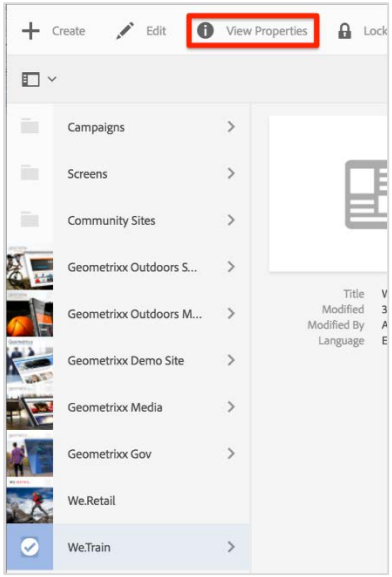
These two mechanisms allow us to control where templates are made available for use. The allowedPaths property allows the developer to decide where templates can be used. The cq:allowedTemplates property allows the website owner to decide which templates can be used within the site.

Task – Add Content Structure to the Template

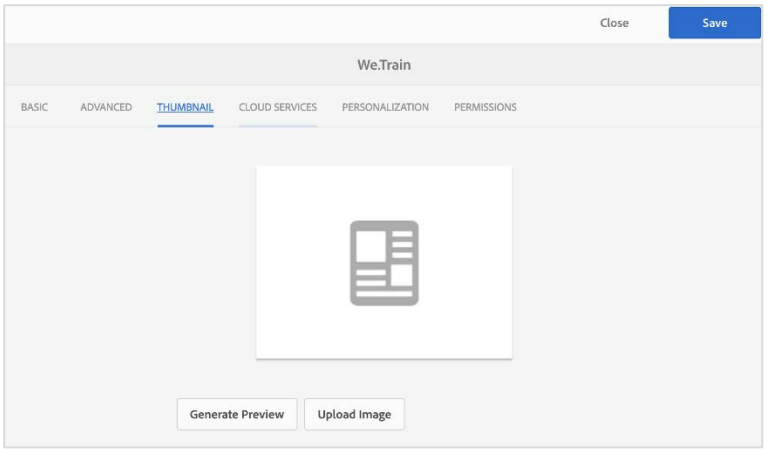
As mentioned earlier, the template plays many roles. One role that we experienced earlier is to appear in the list of available templates when creating a page. Another role that the template plays is to "front load" content onto a page created from that template.

So let's say we wanted every page, created by the contentpage template in the We-train site, to have the same thumbnail image.

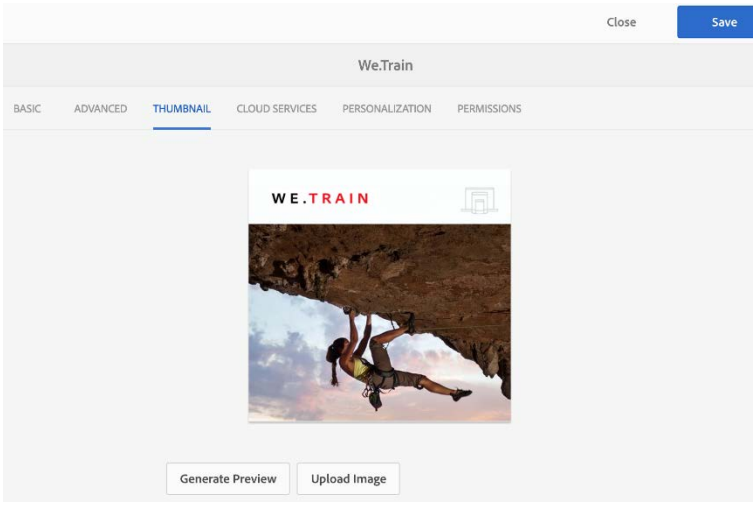
1. Using the Sites Console, navigate to **We.Train**.
2. Select **We.Train**.
3. Click View Properties.



4. Click the **Thumbnail** tab.

5. 

6. Click on **Upload Image** and upload **We-train.png** into the thumbnail.

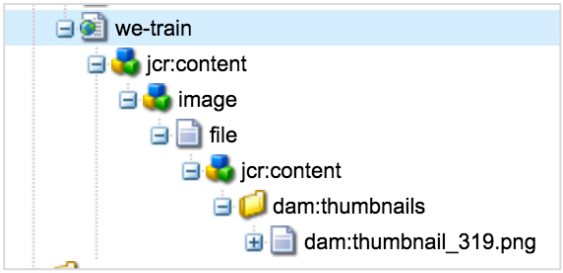
7. 

8. Click **Save.**

9. Now we have created the thumbnail structure, that we want to have on every page, under the /content/we-train page. Lets modify the definition of the /apps/training/templates/contentpage template so that every page created from that template has we-train.png as its thumbnail.

10. Using CRXDE Lite, navigate to **/content/we-train/jcr:content**.

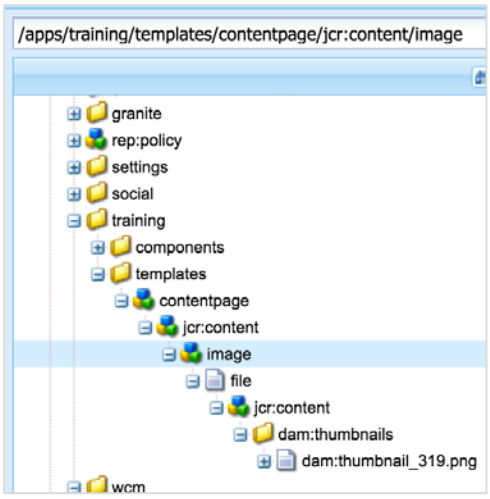
11. Notice the newly defined structure under the jcr:content node.

12. 

13. Right-click on the image node and select **Copy**.

14. Navigate to /apps/training/templates/contentpage/jcr:content.

15. Right-click the **jcr:content** node and select **Paste**.

16. 

When a page is created from a template, AEM writes the cq:Page parent and then copies the template's jcr:content child as the child to the cq:Page node. Now every page we create from the contentpage template will have the thumbnail.

6.4 Creating the Website Structure

A page is where content authors create and edit content that most likely will be published and viewed by site visitors.

A page is many things:

- Website content container
- Instance of a template
- cq:Page JCR Node type (has a mandatory jcr:content child node)

When creating a page, the content that you enter in the dialog box becomes the nodes and associated properties for that page. The Page Creation wizard allows you to enter the following information, which is necessary to create the complete page structure.

- Name: cq:Page node name
- Title: jcr:title property
- Template: cq:template property

The template's *sling:resourceType* property is added as the page's *sling:resourceType* property, so the page knows where its rendering script is. When working with pages, you can use the following WCM APIs:

- com.day.cq.wcm.api.Page
- com.day.cq.wcm.api.PageManager
- com.day.cq.wcm.api.PageFilter

6.5 Lab Activity – II

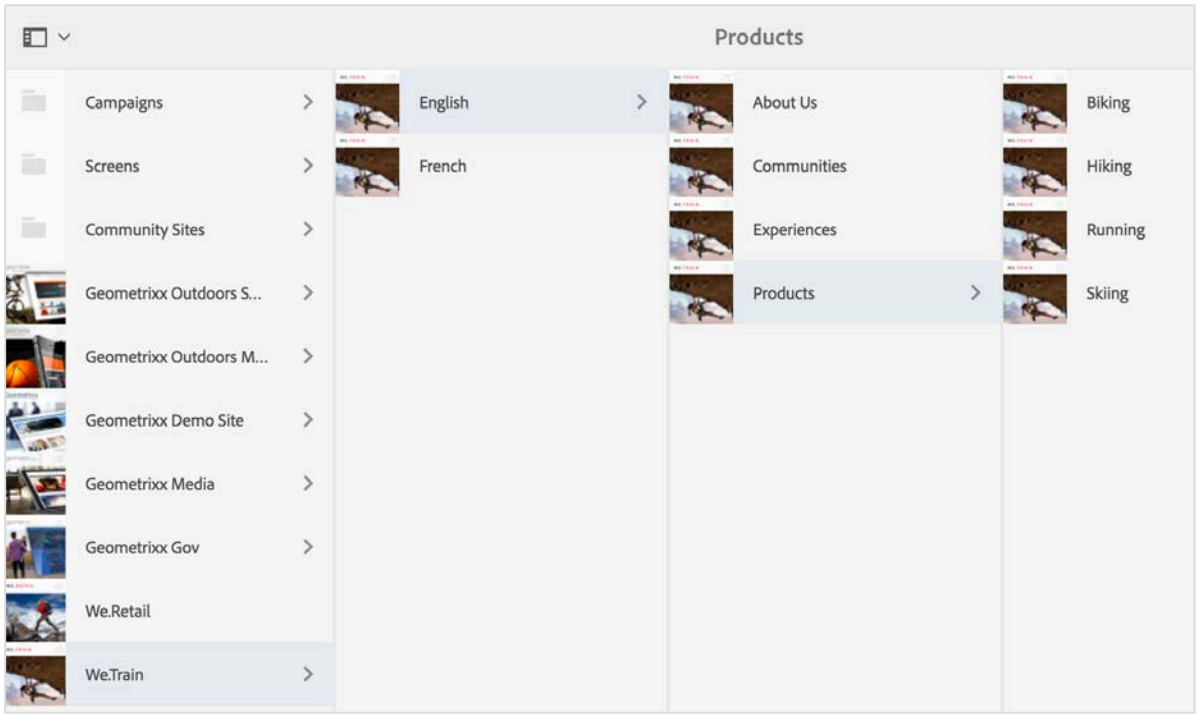
Task – Create the pages for the site

Now it is time to create our website structure. It won't be a very interesting website yet, as all the pages will be identical. But we will continue to build out our page-rendering component and create other components to fill out the page content.

1. Using the Sites Console, navigate to **We.Train**.

2. Click Create Page.
3. Choose the **We.Train** template and click **Next**.
4. Fill in the **Name** and **Title**:
   - Name: en
   - Title: English
   - This page will be the English language root.
5. Click **Create,** and then click **Done**.
6. At the same level, create the **French** language root:
   - Name: fr
   - Title: Francais
7. Notice the thumbnail image appearing under the English and French language roots:

8. This came from the content that we placed under the template in an earlier exercise.
9. Under the **English** root, create the following website structure:
   - English >
     - About Us
     - Communities
     - Experiences
     - Products >
       - Biking
       - Hiking
       - Running
       - Skiing



7 Introduction to HTL

7.1 Working with HTL

HTL is the new templating language developed by Adobe. It is the recommended language to use for developing components using Adobe Experience Manager.

A HTL template defines an HTML output stream by specifying the presentation logic and the values to be dynamically inserted into the stream based on some background business logic. HTL differs from other templating systems in three main ways:

- HTL is HTML5: A template created in HTL is a valid HTML5 file. All HTL-specific syntax is expressed within a data attribute or within HTML text. Any HTL file opened as HTML in an editor will automatically benefit from features such as auto-completion and syntax highlighting, which are provided by the editor for regular HTML.

```
<div class="sightly-example">
    <header data-sly-include="header.html"></header>
    <h1 data-sly-test="${properties.title}">
        ${properties.title}
    </h1>
    <section data-sly-use-navigation="Navigation">
        <h1>
            ${navigation.breadcrumb}
        </h1>
    </section>
    <ul data-sly-list-child="${resource.listChildren}">
        <li>${child.name}</li>
    </ul>
</div>
```

- Separation of concerns: The expressiveness of the HTL markup language is purposely limited so that only relatively 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.
- Compatible with JSP or ESP.

7.1.1 HTL Syntax

There are two different types of syntaxes:

- HTL Block Statements - To define structural elements within the template, HTL employs the HTML data attribute, which is HTML5 attribute syntax purposely intended for custom use by third-party applications. All HTL-specific attributes are prefixed with data-sly-.
- HTL Expressions - 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.

The following is a list of a few of the statements, expressions, and tags.

- **Comments**
  Example: `<!--/* A HTL Comment */-->`
- **Expressions**
  Examples: `${true}`, `${properties.text}`
- **Enumerable objects**
  Examples: `pageProperties`, `properties`, `inheritedPageProperties`
- **URI Manipulation**
  Example: `${'example.com/path/page.html' @ scheme='http'}`
- **HTL Block Statements**
  Examples: `data-sly-use`, `data-sly-unwrap`, `data-sly-text`, `data-sly-list`, `data-sly-include`, `data-sly-repeat`
- **Special HTML tags**
  Example: `<sly>`

For more information on the HTL syntaxes, refer to https://docs.adobe.com/docs/en/htl/overview.html

7.2 Page-rendering Scripts

When you create a component, a rendering script is also created by default. When you view a page, the output is displayed from the rendering script.

Adobe recommends that you use HTL as the rendering script. Apart from using the HTL tags, you just need to change the extension of the script from .jsp to .html.

7.3 Use APIs to Display Basic Page Content

This step is an introduction to rendering the content from the repository, which is a similar concept to querying and displaying data from a database. In the repository, the nodes define the structure and the properties hold the data. To render content on any page, you need to render the data from those properties. Initially, start with the basic properties associated with every page.

Here are some of the ways that you can access the content in Adobe Experience Manager:

- The **currentPage** object
  The currentPage object is an instance of the page (see AEM API) class, which provides some methods to access content. For example: `${currentPage.Title}`
- The **properties** object
  The properties object is an instance of the ValueMap (see Sling API) class and contains all properties of the current resource. For example: `<p> Title : ${properties.jcr:title}</p>`
- The **currentNode** object
  The currentNode object is an instance of the Node (see JCR API) class, which provides access to content using the getProperty() method. For example: `${currentNode.Name}`

7.4 Modularize the Page Component

It is important to modularize a component into multiple scripts and include them at runtime—promoting component or script reuse. You use the HTL **include** tags to support modularization. There are different ways in which you can include a file to the script.

For example, you can include a file at runtime using HTL as: `<div data-sly-include=”myScript.html”/>`

7.5 Lab Activity - I

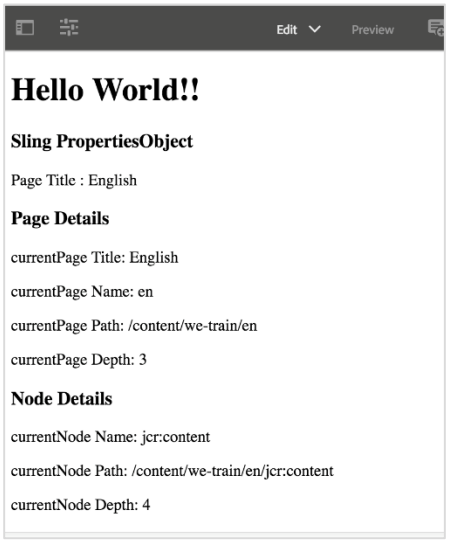
Task – Render Basic Page Content

At this point, all of the pages in the We.Train website are identical. Now we will begin to use HTL to render the content residing in properties.

1. Using CRXDE Lite, navigate to /apps/training/components/structure/contentpage.
2. Open contentpage.html.
3. Replace the existing code with the code from the USB Contents, and save your changes.

contentpage.html

```
<!DOCTYPE html!>
<!--/* A simple HTL script */-->
<html>
       <head>
       <meta charset="utf-8"/>
    </head>
    <body>
       <h1>Hello World!!</h1>
        <h3>Sling PropertiesObject</h3>
                    <p>Page Title : ${properties.jcr:title}</p>

              <h3>Page Details</h3>
                    <p>currentPage Title: ${currentPage.Title}</p>
                    <p>currentPage Name: ${currentPage.Name}</p>
                    <p>currentPage Path: ${currentPage.Path}</p>
                    <p>currentPage Depth: ${currentPage.Depth}</p>

        <h3> Node Details </h3>
                    <p>currentNode Name: ${currentNode.Name}</p>
                    <p>currentNode Path: ${currentNode.Path}</p>
                    <p>currentNode Depth: ${currentNode.Depth}</p>
    </body>
</html>
```

4. Using the Sites Console navigate to **Sites** > **We.Train** > **English** and open the page.
5. 

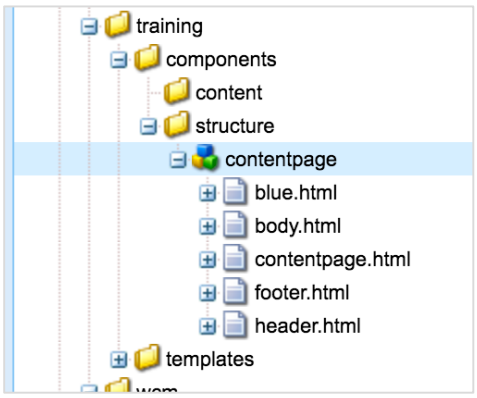
6. Examine the rendered content.
7. Open the **About Us** child page of English.
8. 

9. Examine the rendered content. Notice the differences, as compared to the English page. Each page is now unique, as we are rendering, for each page, the properties associated with that page.

What we have just done are the basics for page rendering. As we have mentioned before, in the repository, the nodes provide the structure and the properties hold the data. By rendering the property values associated with the page, we have rendered the page content.

Task – Modularize the contentpage component

Modularization of components enables reuse of common structures and reduces code redundancy.

1. Using CRXDE Lite, right-click /apps/training/components/structure/contentpage and select Create…> Create File.
2. Enter **body.html** for the file name. Save your changes.
3. Repeat steps 1 and 2 to create **header.html** and **footer.html**.
4. 

5. Examine the code for body.html, header.html, and footer.html.

body.html

```
<div class="container we-Container--main">
    <div class="root responsivegrid">
        <div class="aem-Grid aem-Grid--12 aem-Grid--default--12  ">
            <div class="header aem-GridColumn aem-GridColumn--default--12"
data-sly-include="header.html"></div>
            <div class="hero-image image parbase aem-GridColumn aem-
GridColumn--default--12" style="padding-top: 100px;">
                <div>Hero Component</div>
                <div class="responsivegrid aem-GridColumn aem-GridColumn--
default--12">
                    <div class="aem-GridColumn aem-GridColumn--default--12">
                        <div class="row">
             <div>Breadcrumb</div>
                            <div class="we-Header">Title Component</div>
                            <div>Responsive Content Area</div>
                        </div>
                    </div>
                    <form class="page__print">
                        <input value="Print Friendly" type="submit" />
                    </form>
                    <div class="footer aem-GridColumn aem-GridColumn--
default--12" data-sly-include="footer.html"></div>
                </div>
            </div>
```

```
        </div>
    </div>
</div>
```

6. header.html

```
<div class="navbar navbar-inverse navbar-fixed-top hidden-xs">
    <div class="container-fluid">
        <nav style="color: white;" >Language Navigation</nav>
        <ul class="nav navbar-nav navbar-right" style="color: white;" >
            <sly>Toolbar</sly>
        </ul>
    </div>
</div>
<div >Site Navigation</div>
```

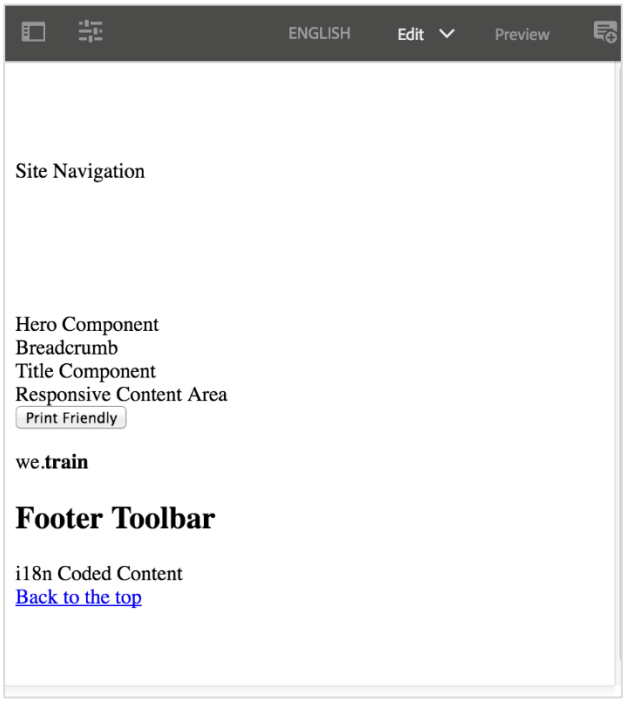
7. footer.html

```
<footer class="we-Footer width-full">
    <div class="container">
        <div class="row">
            <div class="row we-Footer-section we-Footer-section--sub">
                <div class="we-Footer-section-item">
                    <div class="we-Logo we-Logo--big">
                        we.<strong>train</strong>
                    </div>
                    <div class="we-Footer-nav">
                        <h2>Footer Toolbar</h2>
                    </div>
                </div>
            </div>
            <div class="row we-Footer-section we-Footer-section--sub">
                i18n Coded Content
            </div>
            <div class="row">
                <div class="col-md-12">
                    <div class="text-center">
                        <a href="#top" class="btn btn-primary">Back to the
top</a>
                    </div>
                </div>
```

```
            </div>
        </div>
    </div>
</footer>
```

8. Using the code from the USB Contents, paste the code for each of the three scripts into the script files that you created.
9. Modify the contents of **contentpage.html** using the content from the USB Contents.

```
<!DOCTYPE html!>
<!--/* A simple HTL script */-->
<html>
 <head>
   <title>${properties.jcr:title}</title>
  </head>
  <body>
    <div data-sly-include="body.html"></div>
  </body>
</html>
```

10. Using the Sites Console, open **Sites** > **We.Train** > **English**.
11. 

Notice that there are placeholders for components and other elements that we will be implementing throughout the course. There is no design applied yet, so the class definitions that we referenced in the code are not active yet. We will define and apply the design later in the course.

8 Inheritance

8.1 Inheriting Foundation Components

Components can be given a hierarchical structure to implement the inheritance of script files, dialog boxes, and so on. Therefore, it is possible for a specific 'page' component (or any component) to inherit from a 'base' component. For example, allowing inheritance of a script file for a specific part of the page, for example, the <head> section.

During this course, you would inherit a few of the foundation components such as:

- Page component
- Paragraph system component

8.1.1 Types of Hierarchies

Components within AEM are subject to three different hierarchies:

1. **Resource Type Hierarchy**
   This is used to extend components using the property sling:resourceSuperType. This enables the component to inherit from a 'base' component. For example, a text component will inherit various attributes from the foundation text component, including:

   a. scripts (resolved by Sling)

   b. Dialog boxes

   c. Descriptions (including thumbnail images, icons, and so on)

   It is important to note that a local copy or instance of a component element (for example, body.html) will take precedence over an inherited element.

2. **Container Hierarchy**
   This is used to populate configuration settings to the child component and is most commonly used in a paragraph system scenario. For example, 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) can be defined on the parent component and propagated to the children components.

   Configuration settings (related to edit functionality) in cq:editConfig and cq:childEditConfig are propagated.

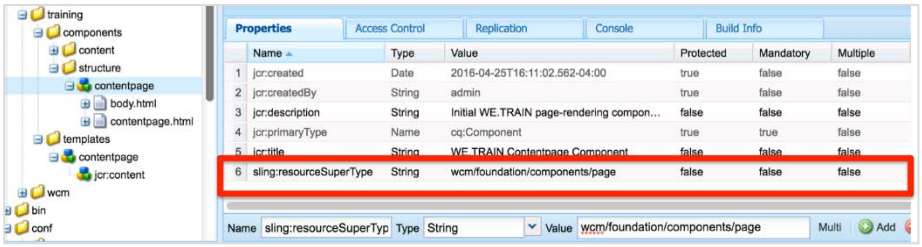
3. **Include Hierarchy**
   This is imposed at runtime by the sequence of includes. It is typically used by the designer, which in turn acts as the base for various design aspects of the rendering—including layout.

   Example: Information, CSS information, the available components in a paragraph system, and so on.

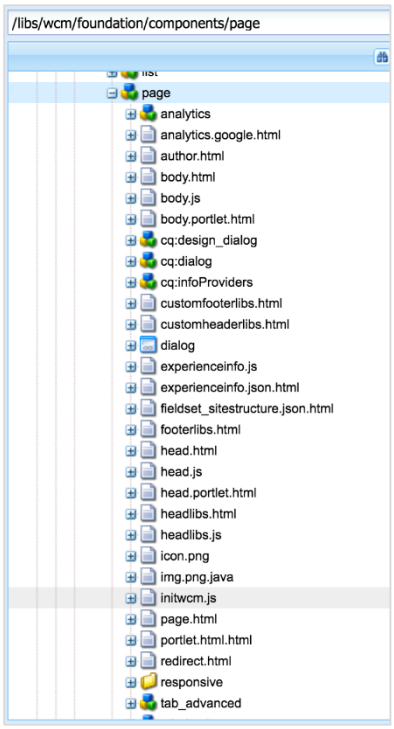
8.2 Lab Activity

When the /apps/training/components/structure/contentpage component was defined, we declared the Foundation Page component as the superType of our page-rendering component. By doing this, we allowed our contentpage component to initialize the WCM inorder to take full advantage of the authoring environment. By declaring the Foundation Page component as its superType, the contentpage component will also inherit and make use properties, scripts, and other features of the Foundation Page component.

Task – Investigate the contentpage sling:resourceSuperType property

1. Navigate to /apps/training/components/structure/contentpage.
2. Notice the sling:resourceSuperType property
3. Defining the sling:resourceSuperType property enables inheritance from a super type or, as it is sometimes called, a base type. Once you define the sling:resourceSuperType property with the value of wcm/foundation/components/page, the /apps/training/components/structure/contentpage component may now inherit scripts, properties, dialog box structures, and other elements from the Foundation Page component.
4. 

Task – Investigate the Foundation Page Component

1. Using CRXDE Lite, navigate to **/libs/wcm/foundation/components/page**. You will note that the Foundation page component contains all the pieces necessary to be a complete page-rendering component.
2. 

3. Take particular notice of the default script: **page.html** and **head.html**.

Task – Delete the contentpage.html script

1. Using CRXDE Lite, delete **/apps/training/components/structure/contentpage/contentpage.html** script by right-clicking it and selecting Delete. Save changes.
2. Using the Sites Console, open **Sites** > **We.Train** > **English**

Notice that the page renders just fine, even without the default script, contentpage.html. The proper rendering of the page is possible because the contentpage component is inheriting a number of scripts from the Foundation Page component.

Extra Credit Task – Build the rendering chain

1. Use CRXDE Lite to follow the Sling URL decomposition and resource resolution process.
2. Use CRXDE Lite to find the initial rendering scripts and build the rendering chain for the **Sites** > **We.Train** > **English** page.

9 Design and Styles

9.1 Adding a design to your site

Designing your site involves creating a package of CSS files, client libraries, images, and anything else that gives the overall look and feel of your site. Creating and assigning a design(er) in Adobe Experience Manager allows you to enforce a consistent look and feel across your website, as well as share global content. The pages that use the same design(er) will have access to common CSS files, defining the formats of specific areas or components, and images that you use for features such as backgrounds and buttons.

Adobe Experience Manager was developed to maximize compliance with the Web Accessibility Guidelines. Web accessibility means that people with disabilities can perceive, understand, navigate, and interact with the web, and they can contribute to the web. This can include measures such as providing textual alternatives to images or any non-text item. These can then be used to help people with sight impairment by outputting the text on a Braille keypad, or through a voice synthesizer. Such measures can also benefit people with slow Internet connections, or any Internet user—when the measures offer the user more information.

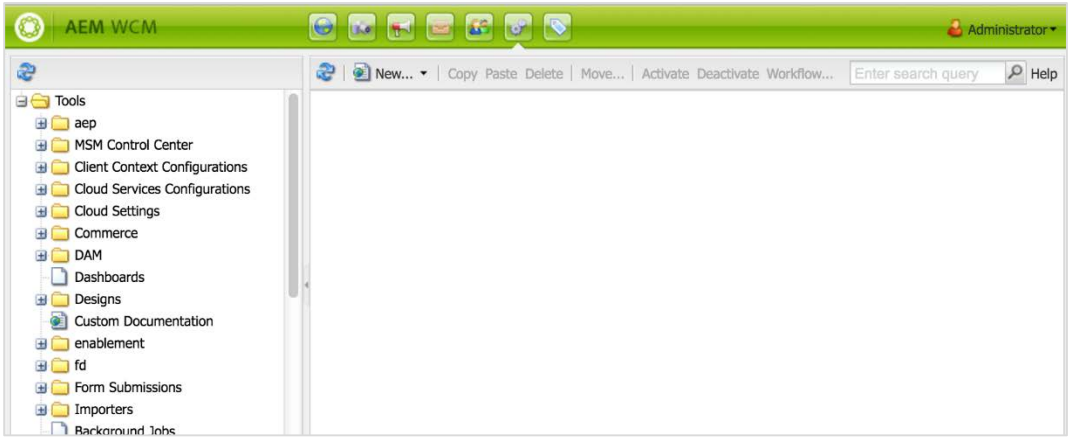
These mechanisms must be carefully planned and designed to ensure that they provide the information required for the user to successfully understand and use the content. Certain aspects are integral to Adobe Experience Manager, whereas other aspects must be realized during your project development.

In Adobe Experience Manager, designs are stored in the **/etc/designs** folder.

9.2 Lab Activity

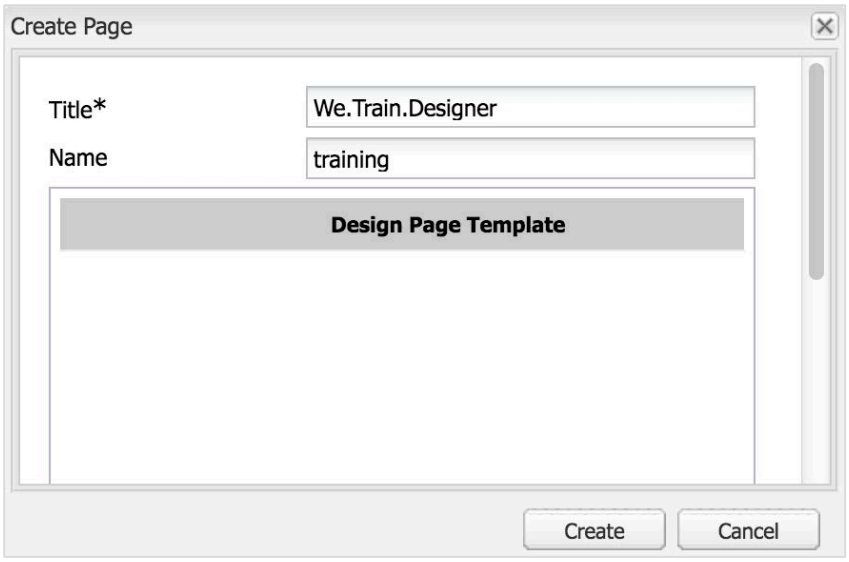
Task - Define a design

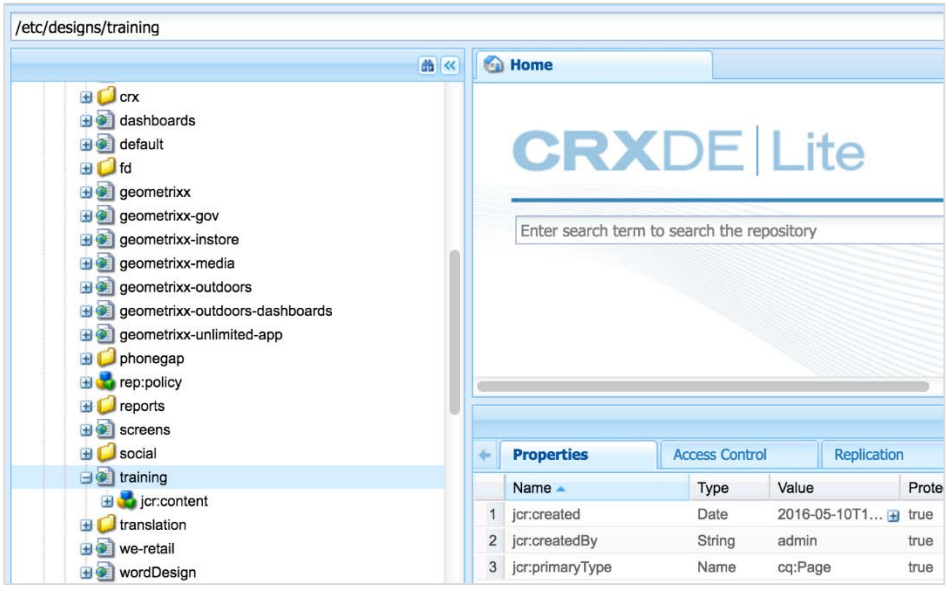
1. From the AEM Home screen, navigate to the **Tools** console using one of the following methods:
   - http://localhost:4502/miscadmin
   - Type "**/**" and enter "**Configuration**" into the OmniSearch
   - Click on the Global Navigation bar. Then click on **Tools** (the hammer icon) in the left-hand column, and then click on **Operations** > **Configuration**.



2. Navigate to and select **Designs**.

3. From the toolbar, Select **New** > **New Page**.
4. Enter the following into the **Create Page** dialog box.
    a. **Title**: We.Train.Designer
    b. **Name**: training



5. The **Design Page** Template is pre-populated, click **Create**.
6. Using CRXDE Lite, navigate to **/etc/designs/training**. This is where the design is stored.
7. 

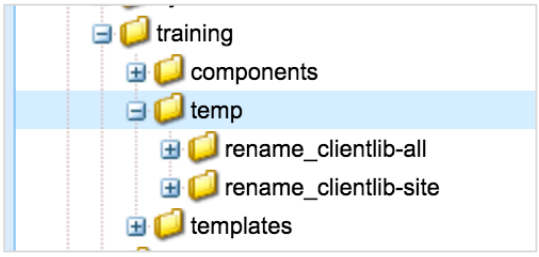
8. Note that the design is a node of type cq:Page with its associated jcr:content child. The client libraries that make up the look and feel of the site will be stored under the design node.

Due to time contraints, we will upload an already created design.

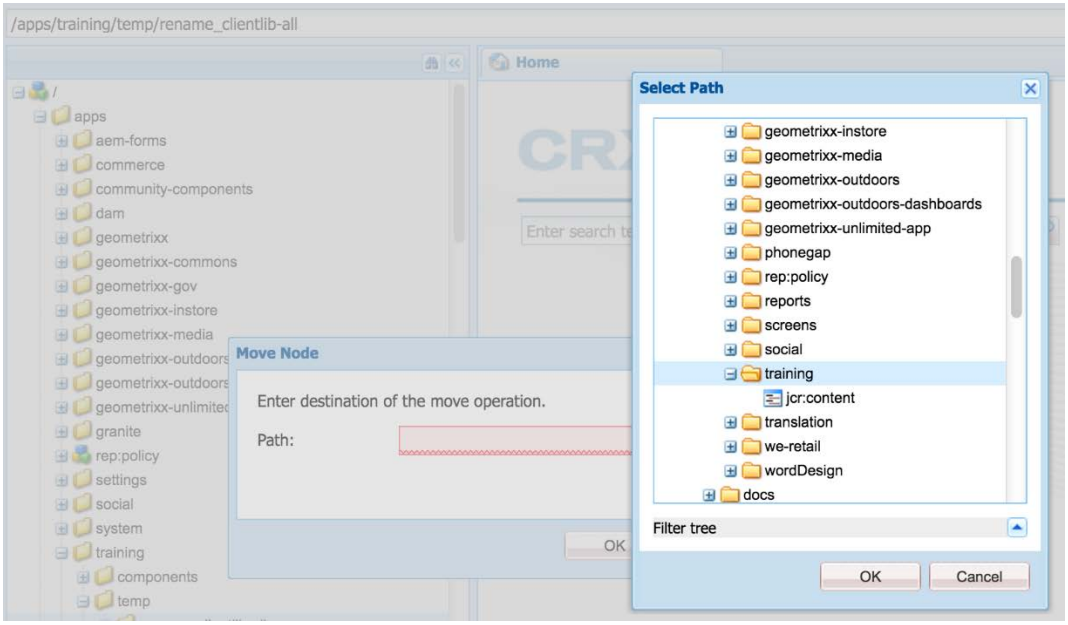
9. Enter http://localhost:4502/crx/packmgr/index.jsp or click on the package icon in the toolbar on the CRXDE Lite page.

10. Click on Upload Package.
11. In the file upload dialog box, browse to the USB Contents, and select **we.train-design-package.zip**.
12. Click **OK** and then **Install** twice.

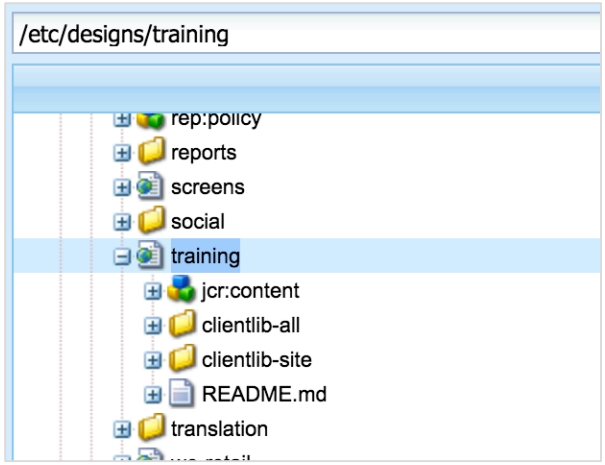
Let's check out what happened in the repository.

13. Using CRXDE Lite, click the **Develop** icon in the top tool bar and navigate to **/apps/training/temp**. You will notice two client libraries, that will make up our design.
14. 

15. Right click on each client library, and select **Move**.
16. Click the search icon, and select the path as **/etc/designs/training**. Save changes.

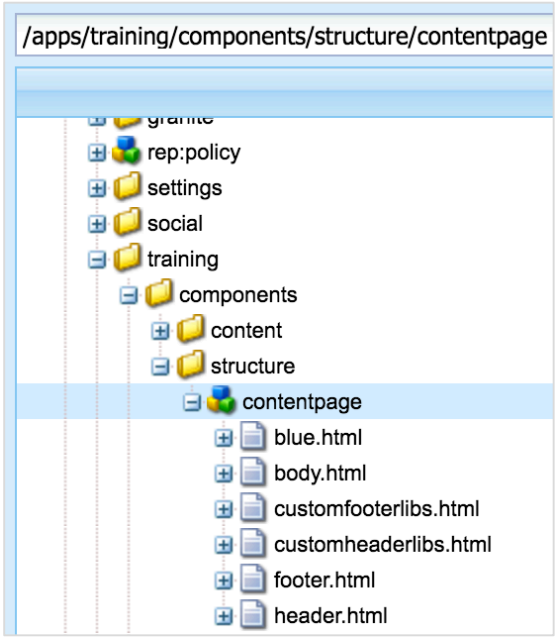


17. Right-click on each client library, rename (as follows), and save changes.
    a. rename_clientlib-site becomes clientlib-site
    b. rename_clientlib-all becomes clientlib-all



Now we have a design defined. Next thing to do is to associate the design with the We.Train website.

Task – Modify the contentpage component to call in the design.

1. Using CRXDE Lite, navigate to /apps/training/components/structure/contentpage.
2. Add the following two files to the contentpage component: **customheaderlibs.html** and **customfooterlibs.html**.
3. 

4. Using the code from the USB Contents, enter the contents of **customheaderlibs.html** and **customfooterlibs.html**

customheaderlibs.html

```
<!--/* Include the site client libraries (loading only the CSS in the header,
JS will be loaded in the footer) */-->

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

customfooterlibs.html

```
<!--/* Include the site client libraries (loading only the JS in the footer,
CSS was loaded in the header) */-->
<sly data-sly-use.clientLib="/libs/granite/sightly/templates/clientlib.html"
data-sly-call="${clientLib.js @ categories='we.train.all'}"/>
```

Notice that, as suggested by best practices, the css files are called in at the top of the page and the javascript files are called in at the end of the page.

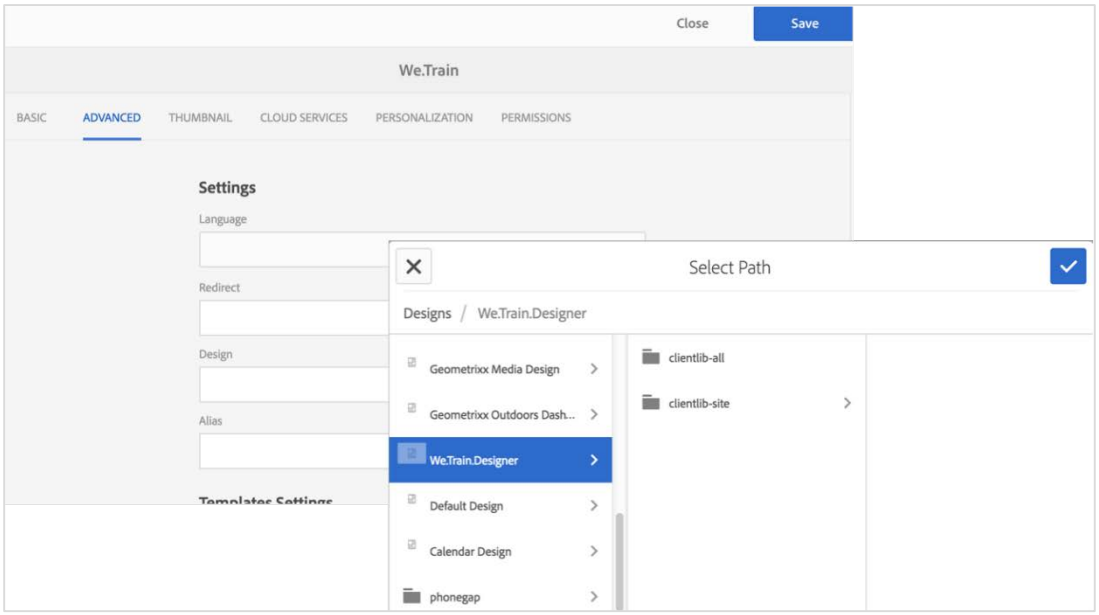
5. Using the code from the USB Contents, modify **footer.html** to call the **customfooterlibs.html**.

footer.html

```
<footer class="we-Footer width-full">
    <div class="container">

        <div class="row">
            <div class="row we-Footer-section we-Footer-section--sub">
                <div class="we-Footer-section-item">
                    <div class="we-Logo we-Logo--big">
                        we.<strong>train</strong>
                    </div>
                    <div class="we-Footer-nav">
                        <h2>Footer Toolbar</h2>
                    </div>
                </div>
            </div>

            <div class="row we-Footer-section we-Footer-section--sub">
                i18n Coded Content
            </div>

            <div class="row">
                <div class="col-md-12">
                    <div class="text-center">
                        <a href="#top" class="btn btn-primary">Back to the
top</a>
                    </div>
                </div>
            </div>
        </div>
          <sly data-sly-include="customfooterlibs.html" />
```

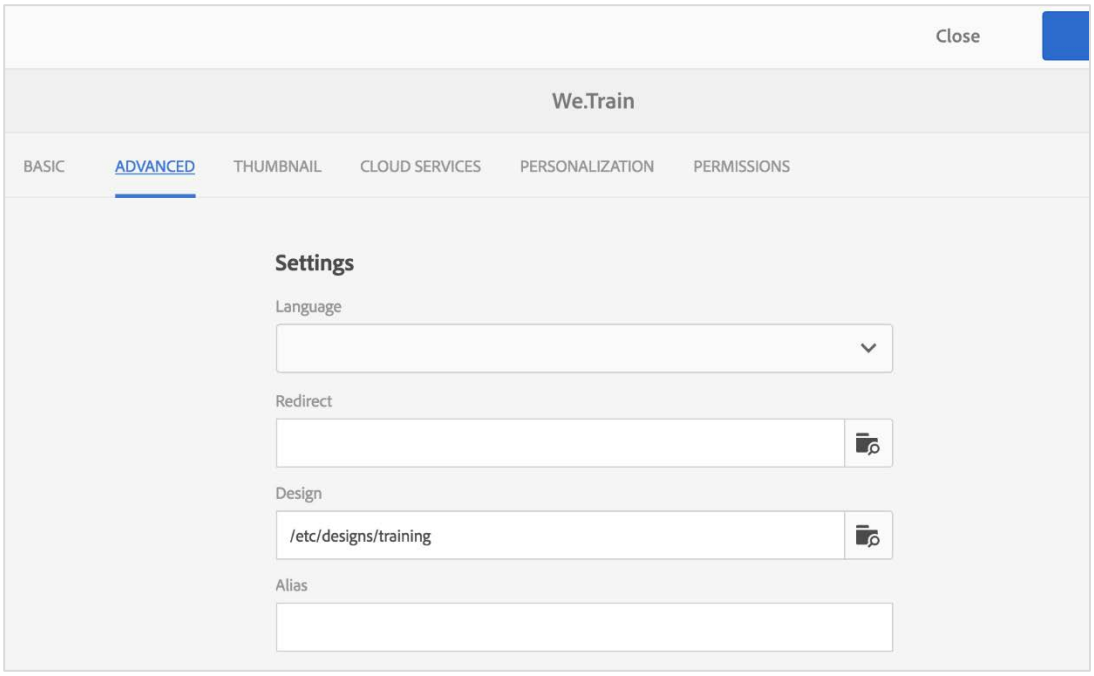
```
    </div>
</footer>
```

Task – Assign a design to the website

1. Using the Sites Console, navigate to **Sites** > **We.Train**.
2. Select the **We.Train** page, and click **View Properties**.
3. On the **Advanced** tab, click on the browser icon to the right of the design input field.
4. 

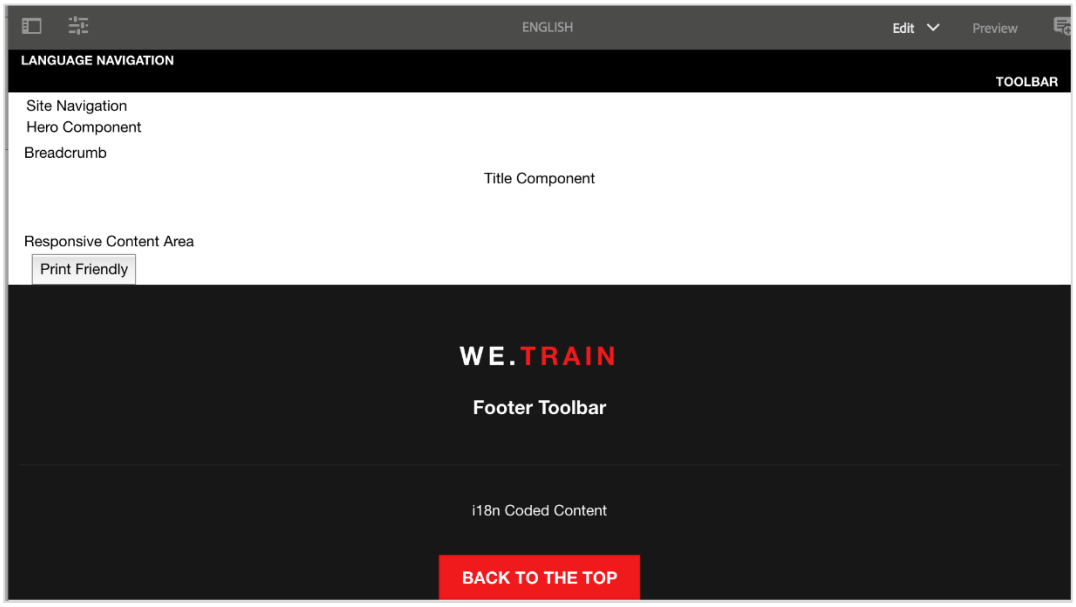
5. Select We.Train.Designer.
6. Click the checkmark.
7. Click **Save,** and then **Close**.

8.



Now let's see the design in action.

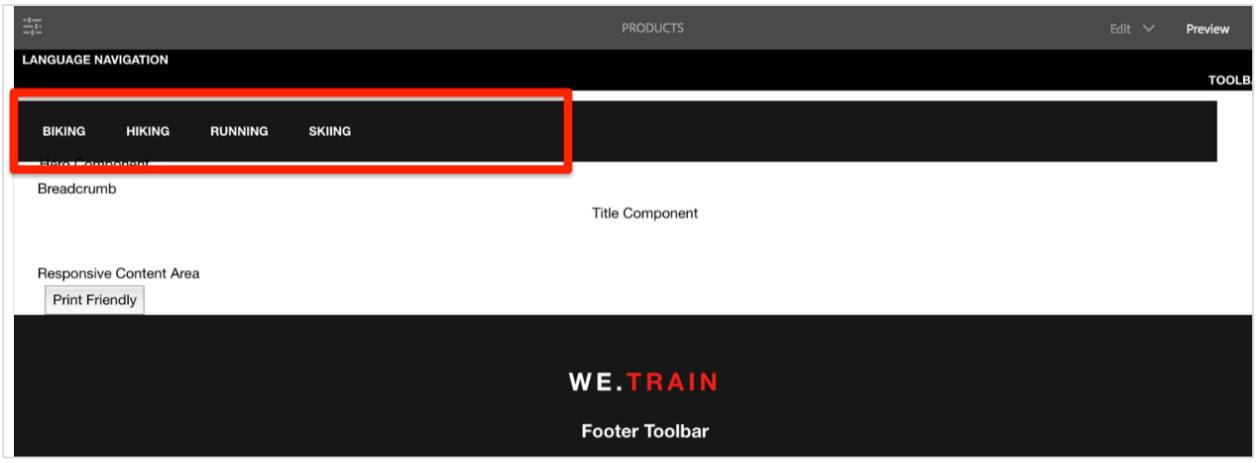
9. Using the Sites Console, navigate to **Sites** > **We.Train** > **English**.
10. Open the **English** page.



Notice that English page has inherited the design from the We.Train website root. Now each page in the website is using the newly defined design.

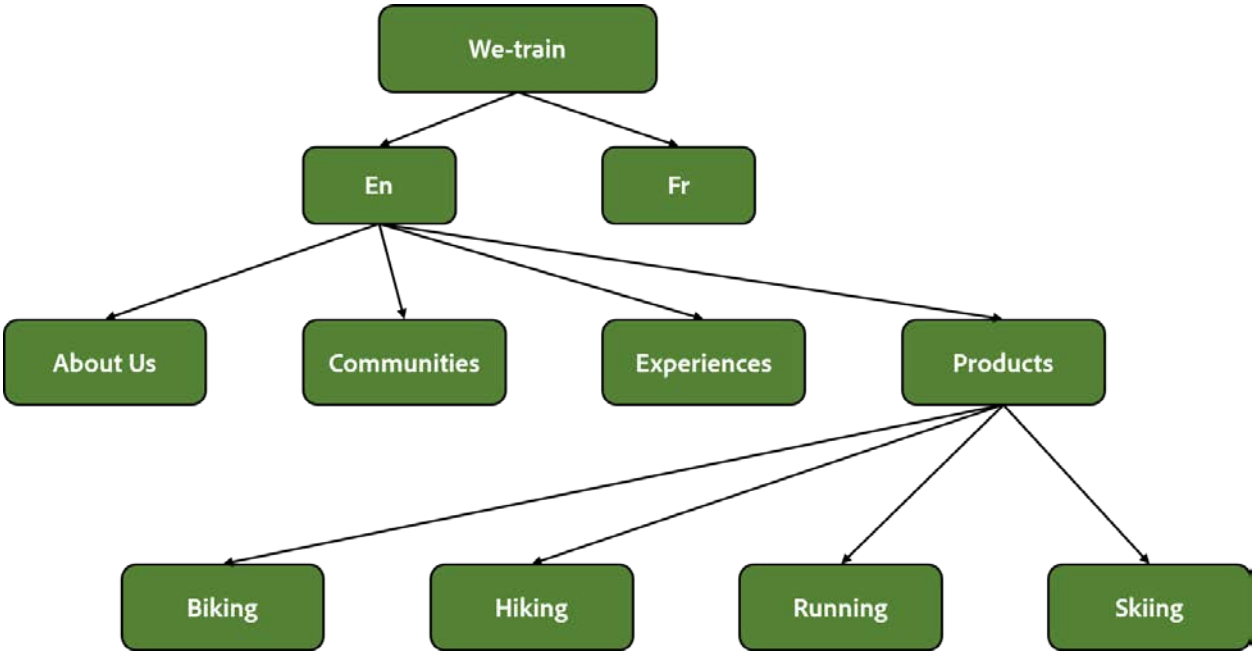
10 Authoring Structure Components

The navigation component is a horizontal menu at the top of the page that provides links to navigate to a specific page in the site.



10.1 Creating a Top Navigation Component

To demonstrate the component creation process, you will create a dynamic text-based navigation component, allowing for website structure to be easily modified, represented, and navigated in real time. The following image indicates a simple website structure that has four levels.



A simple navigation component would have all the child page listed of the current page. For example, in the above structure, if you were in the Products page, you would have the navigation component listing Biking, Hiking, Running, and Skiing. However, when you take a page that has no child pages, the navigation component would be blank. For example, navigating to the About Us page would have no pages listed in the navigation component, as it has no child pages. To overcome this issue, you could have a component that always lists the child pages of the root page.

Let's see how you can create a navigation component, and then see how to optimize that component.

How Do I Create Dynamic Navigation?
Providing dynamic navigation capabilities, allowing for the easy addition and removal of pages, is one of the most important (and sometimes difficult) tasks you can do as a developer in Adobe Experience Manager.

You can include a component from within the script using the **data-sly-resource** tag. The following code includes a component named *topnav* to the script.

```
<div data-sly-resource="${'topnav'@ resourceType='training/components/topnav'}"></div>
```

The resourceType provides the location of the component. It also provides the name of the component that appears in Design mode.

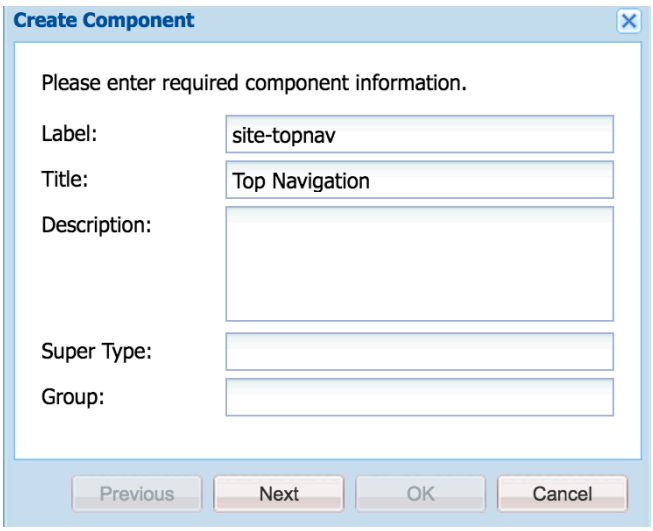
The following code snippet is used to extract the child pages of the current page:

```
<ul class="topnav" data-sly-list="${currentPage.listChildren}">
<li><a href="${item.path}.html">${item.title}</a></li>
</ul>
```

To get child pages of the root page, add a JavaScript file with the method: `currentPage.getAbsoluteParent()`

10.2 Lab Activity - I

Task – Create a simple Navigation component

1. Using CRXDE Lite, navigate to /apps/training/components/structure.
2. Right-click on /apps/training/components/structure and select Create… > Create Component.
3. Enter the following values in the dialog:
    a. Label: site-topnav
    b. Title: Top Navigation
4. 

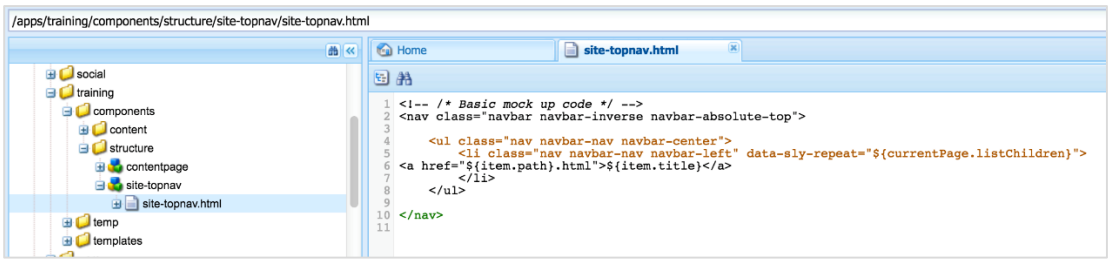
5. Click **Next**. Click **OK**.

6. Save your changes.
7. Rename the default script to **site-topnav.html**.
8. Using the code from the USB Contents, replace the sample code in **site-topnav.html**.

site-topnav.html

```
<!-- /* Basic mock up code */ -->
<nav class="navbar navbar-inverse navbar-absolute-top">
<ul class="nav navbar-nav navbar-center">
<li class="nav navbar-nav navbar-left" data-sly-
repeat="${currentPage.listChildren}">
<a href="${item.path}.html">${item.title}</a>
</li>
</ul>
</nav>
```

9. You will notice that the code is a simple navigation that just provides a list of the children of the current page.
10. Save the changes.

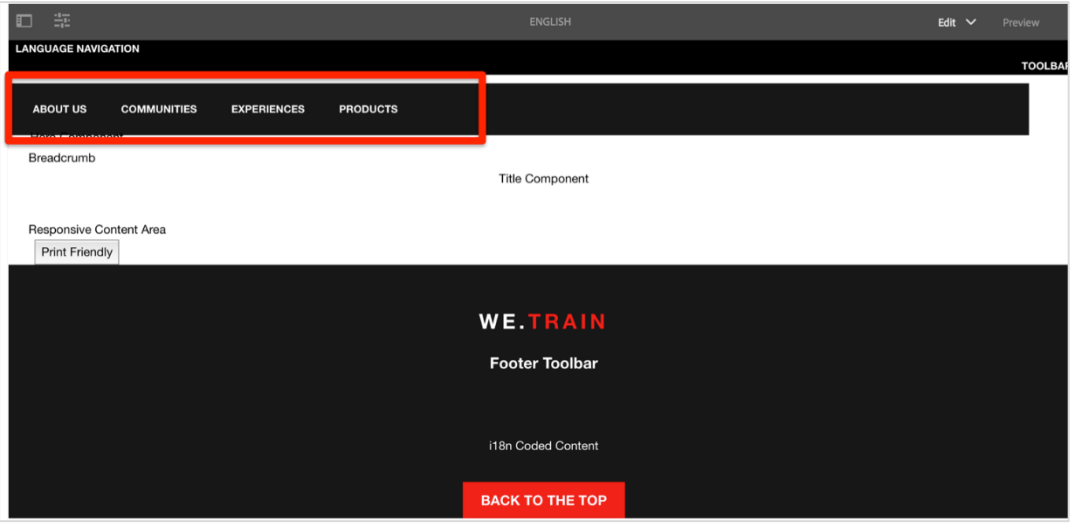


11. Navigate to /apps/training/structure/contentpage.
12. Using the code from the USB Contents, modify the contents of **header.html** to call the Top Navigation component.

header.html

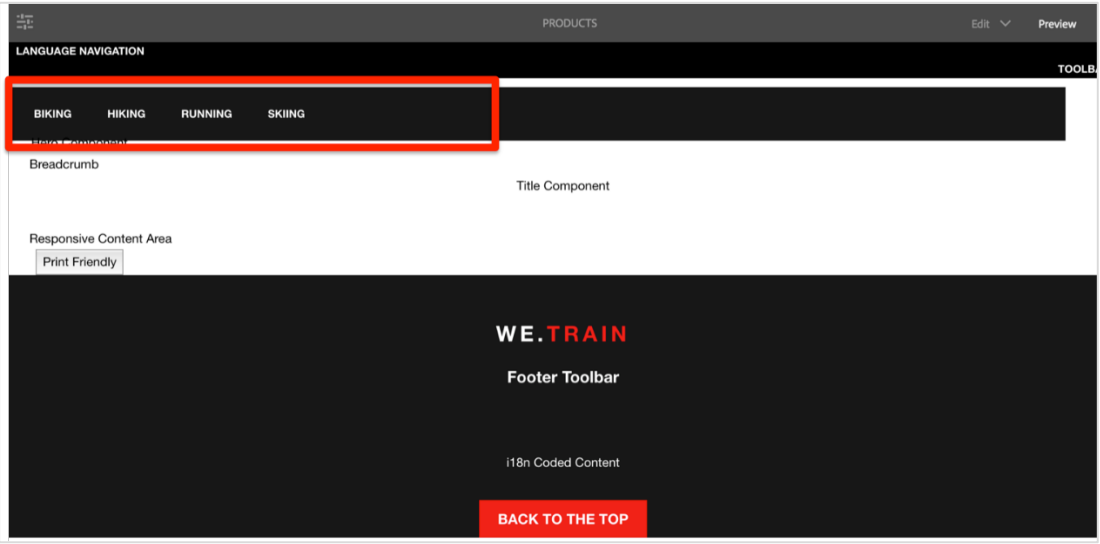
```
<div class="navbar navbar-inverse navbar-fixed-top hidden-xs">
<div class="container-fluid">
<nav style="color: white;" >Language Navigation</nav>
<ul class="nav navbar-nav navbar-right" style="color: white;" >
<sly>Toolbar</sly>
</ul>
</div>
</div>
<div data-sly-resource="${'site-topnav' @
resourceType='training/components/structure/site-topnav'}"></div>
```

13. Using the Sites Console, navigate to **Sites** > **We.Train** > **English**.



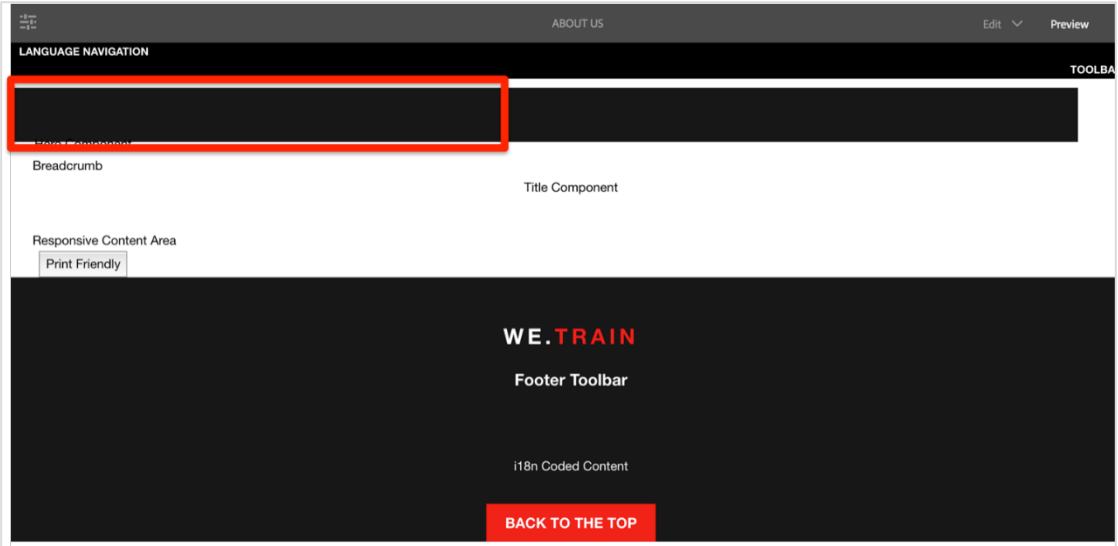
14. Notice that a navigation element has appeared.

15. Navigate to Sites > We.Train > English > Products.



16. Notice that the Navigation menu values have changed. Probably not what we really want for a top navigation which does not typically change as we navigate through the site.

17. Navigate to Sites > We.Train > English > About Us.



18. And for pages that have no children, the navigation menu is empty.

Task – Make the Navigation Component Responsive

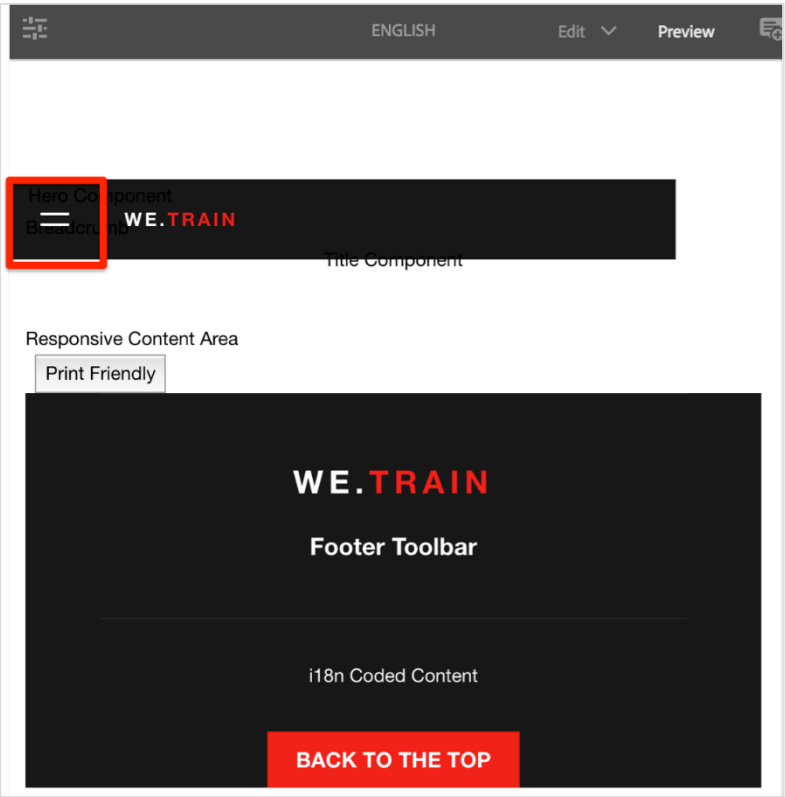
1. Using the code from the USB Contents, replace the code in **site-topnav.html**.

site-topnav.html

```
<!-- /* Add the full responsive design */ -->
<div class="container we-Container--top-navbar">
<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</span>
<span class="icon-bar"></span> <span class="icon-bar"></span>
</button>
<button type="button" class="navbar-toggle navbar-toggle-close collapsed"
data-toggle="collapse" data-target="#we-example-navbar-collapse-inverse"
aria-expanded="false">
<span class="sr-only">Toggle navigation</span>
</button>
<a class="navbar-brand" href="#">we.<strong>train</strong></a>
<div class="pull-right visible-xs"></div>
</div>
<!-- /.navbar-header -->
<div class="collapse navbar-collapse width" id="we-example-navbar-collapse-
inverse">
<ul class="nav navbar-nav navbar-center">
```

```
<li class="visible-xs"><a href="#">we.<strong class="text-
primary">train</strong></a></li>
<!-- /* Basic mock up code */ -->
<li class="nav navbar-nav navbar-left" data-sly-
repeat="${currentPage.listChildren}">
<a href="${item.path}.html">${item.title}</a>
</li>
<li class="visible-xs divider" role="separator"></li>
</ul>
</div>
<span style="height: 0px;" class="navbar-shutter"></span>
</nav>
<!-- /.navbar -->
</div>
```

2. Open **Sites** > **We.Train** > **English** and reduce the size of the window. Notice that the menu changes to become a list.



So now we have a responsive navigation menu, but it is still just a simple list of the children of the current page. It still changes on every page and displays no items when the page has no children.

Task – Create a complex navigation component with a Java Helper

Typically, the top navigation is the same on all pages. So let's create a top navigation component that is more typical. For this task, we will be using a Java helper servlet to perform the business logic of constructing a navigation menu.

Note: the Java class that we will be implementing will be local to the site-topnav component. In a production application, that java class would typically be deployed in an application bundle.

1. Create a file named TopNav.java under /apps/training/components/structure/site-topnav.
2. Using the code from the USB Contents, paste the contents of **TopNav.java**.

TopNav.java

```
package apps.training.components.structure.site_topnav;
import java.util.*;
import java.util.Iterator;
import com.day.cq.wcm.api.Page;
import com.day.cq.wcm.api.PageFilter;
import com.adobe.cq.sightly.WCMUsePojo;
public class TopNav extends WCMUsePojo{
private List<Page> items = new ArrayList<Page>();
private Page rootPage;
// Initializes the navigation
@Override
public void activate() throws Exception {
rootPage = getCurrentPage().getAbsoluteParent(2);
if (rootPage == null) {
rootPage = getCurrentPage();
}
Iterator<Page> childPages = rootPage.listChildren(new
PageFilter(getRequest()));
while (childPages.hasNext()) {
items.add(childPages.next());
}
}
// Returns the navigation items
public List<Page> getItems() {
return items;
}
// Returns the navigation root
public Page getRoot() {
return rootPage;
}
```

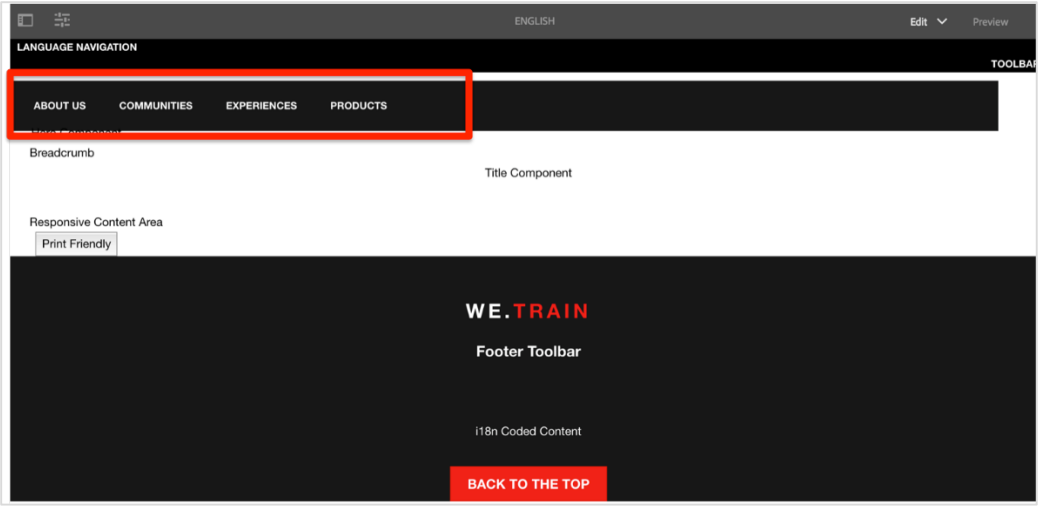
```
}
```

3. Using the code from the USB Contents, modify the contents of **site-topnav.html**. Now site-topnav.html uses TopNav.java to perform the business logic.

site-topnav.html

```
<!-- /* Add the business logic*/ -->
<div data-sly-use.topnav="TopNav" class="container we-Container--top-navbar">
<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</span>
<span class="icon-bar"></span> <span class="icon-bar"></span>
</button>
<button type="button" class="navbar-toggle navbar-toggle-close collapsed"
data-toggle="collapse" data-target="#we-example-navbar-collapse-inverse"
aria-expanded="false">
<span class="sr-only">Toggle navigation</span>
</button>
<a class="navbar-brand"
href="${topnav.root.path}.html">we.<strong>train</strong></a>
<div class="pull-right visible-xs"></div>
</div>
<!-- /.navbar-header -->
<div class="collapse navbar-collapse width" id="we-example-navbar-collapse-
inverse">
<ul class="nav navbar-nav navbar-center">
<li class="visible-xs"><a href="${topnav.root.path}.html">we.<strong
class="text-primary">train</strong></a></li>
<!-- /* Nav with business logic */ -->
<li class="nav navbar-nav navbar-left" data-sly-repeat="${topnav.items}">
<a href="${item.path}.html">${item.title}</a>
</li>
<li class="visible-xs divider" role="separator"></li>
</ul>
</div>
<span style="height: 0px;" class="navbar-shutter"></span>
</nav>
<!-- /.navbar -->
</div>
```

4. Using the Sites Console, navigate to **Sites** > **We.Train** > **English**.



5. Notice the navigation menu.
6. Navigate to Sites > We.Train > English > Products.
7. Notice that the Navigation menu values are unchanged.
8. Navigate to Sites > We.Train > English > About Us.
9. Notice that the Navigation menu values are unchanged.

Now the top navigation menu is the same on all pages.

Extra Credit task

1. Why does the navigation menu show the children of the English page (level 2)?
2. Navigate to Sites > We.Train and open the page. Why does the navigation menu show the children of We.Train (level 1)?

Task – Create a complex Navigation component with a Javascript Helper

As we have discussed, there are two ways to provide more complex business logic than possible with HTML alone. We have already explored using Java helpers. For this task, we will be using a javascript helper to perform the business logic of constructing a navigation menu.

1. Using CRXDE Lite, create a file named topnav.js under /apps/training/components/structure/site-topnav.
2. Using the code from the USB Contents, paste the contents of **topnav.js**.

topnav.js

```
// Server-side JavaScript for the topnav logic
use(function () {
var items = [];
var root = currentPage.getAbsoluteParent(2);
```

```
//make sure that we always have a valid set of returned items
//if navigation root is null, use the currentPage as the the navigation root
if(root == null){
root = currentPage;
}
var it = root.listChildren(new Packages.com.day.cq.wcm.api.PageFilter());
while (it.hasNext()) {
var page = it.next();
items.push(page);
}
return {
items: items,
root: root
};
});
```

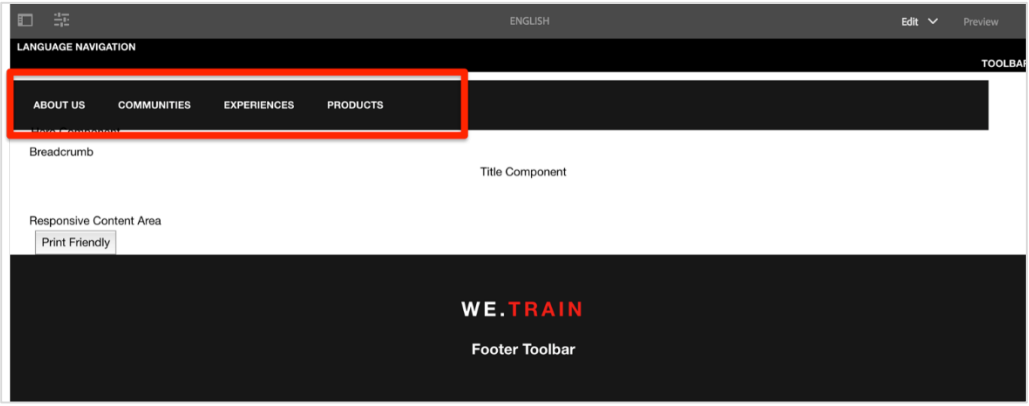
3. Using the code from the USB Contents, modify the contents of **site-topnav.html**. Now **site-nav.html** uses **topnav.js** to perform the business logic.

site-topnav.html

```
<!-- /* Add the business logic*/ -->
<div data-sly-use.topnav="topnav.js" class="container we-Container--top-
navbar">
    <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</span>
                <span class="icon-bar"></span> <span class="icon-bar"></span>
            </button>
            <button type="button" class="navbar-toggle navbar-toggle-close
collapsed" data-toggle="collapse" data-target="#we-example-navbar-collapse-
inverse" aria-expanded="false">
                <span class="sr-only">Toggle navigation</span>
            </button>
            <a class="navbar-brand"
href="${topnav.root.path}.html">we.<strong>train</strong></a>
            <div class="pull-right visible-xs"></div>
        </div>
        <!-- /.navbar-header -->
```

```
        <div class="collapse navbar-collapse width" id="we-example-navbar-
collapse-inverse">
            <ul class="nav navbar-nav navbar-center">
                <li class="visible-xs"><a
href="${topnav.root.path}.html">we.<strong class="text-
primary">train</strong></a></li>
                <!-- /* Nav with business logic */ -->
                <li class="nav navbar-nav navbar-left" data-sly-
repeat="${topnav.items}">
                    <a href="${item.page.path}.html">${item.page.name}</a>
                </li>
                <li class="visible-xs divider" role="separator"></li>
            </ul>
        </div>
        <span style="height: 0px;" class="navbar-shutter"></span>
    </nav>
    <!-- /.navbar -->
</div>
```

4. Using the Sites Console, navigate to **Sites** > **We.Train** > **English**.



5. Notice the navigation menu is the same as when we used the Java helper.

6. Verify that the navigation menu is the same on all children of English.

10.3 Logging

Some Adobe Experience Manager log files provide detailed information about the current system state. In addition to the default system log files, you can also create and customize your own log files. They can help you better track messages produced by your own applications and separate them from the default log entries.

Adobe Experience Manager offers you the possibility to configure:

- global parameters for the central logging service.
- request data logging; a specialized logging configuration for request information.

- specific settings for the individual services; for example, an individual log file and format for the log messages.

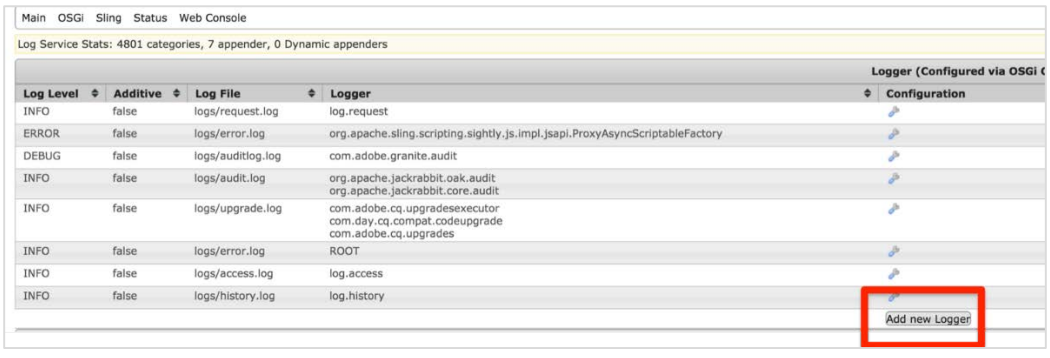
Adding log messages to a component script will allow you to easily debug various scripts you might be working on. In the daily life of a developer, it is often crucial to monitor the values of variables assigned or used. There are several possibilities, in various usability levels. Adobe Experience Manager and CRXDE make your life a little easier by implementing the popular Log4j framework, which is designed to provide an easy-to-use logging solution. The initialization of a Logger object, called log, has already been accomplished during the inclusion of global.jsp in whatever component you may be working on. The log file entries are formatted according to the Sling configuration.

Two pieces of information are required to append an entry to the log file:

- l**og level**: This is provided by the corresponding method call. For example, a log.debug(<message>) produces a message with log level, debug, while a log.info(<message>) produces a message with log level, info.
- Possible methods of the Logger object include:
  - trace()
  - debug()
  - info()
  - warn()
  - error()
- **message**: The message itself is provided as a parameter to the method call. For example, log.debug("This is the log message") appends the message "This is the log message," with a log level of "debug" to the error.log file.

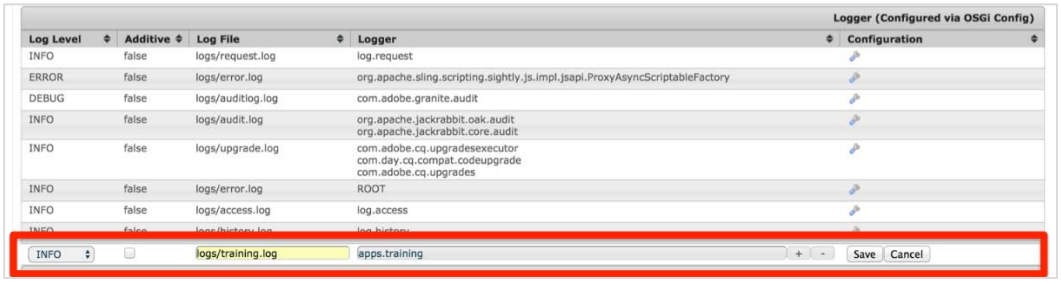
10.4 Lab Activity - II

Task – Create a custom Log File

1. Open the System Console, http://localhost:4502/system/console.
2. Navigate to **Sling** > **Log Support**.
3. 

4. Click on **Add new Logger** button at the bottom of the top panel.
5. Enter the following in the input fields:
   a. Log File: logs/training.log
   b. Logger: apps.training

NOTE: Do not change any other field values.

6. 

7. Click **Save**.

Task – Use log statements in the topnav Component

1. Using CRXDE Lite, navigate to /apps/training/components/structure/site-topnav.
2. Open **topnav.js**. Replace the code with the code from the USB Contents.

topnav.js

```
// Server-side JavaScript for the topnav logic
use(function () {
    var items = [];
    var root = currentPage.getAbsoluteParent(2);

    //make sure that we always have a valid set of returned items
    //if navigation root is null, use the currentPage as the the navigation
root
    if(root == null){
       root = currentPage;
    }

    //Logging Message
    log.info("########[JS] Root page is: " + root.getTitle());

    var it = root.listChildren(new Packages.com.day.cq.wcm.api.PageFilter());
    while (it.hasNext()) {
        var page = it.next();
        items.push(page);
    }

    return {
        items: items,
        root: root
    };
```

```
});
```

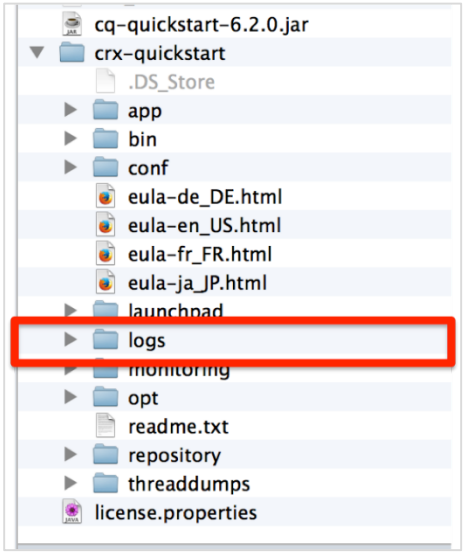
3. Take note of the logging statement:

```
//Logging Message
    log.info("########[JS] Root page is: " + root.getTitle());
```

4. We discussed the global objects before. The Logger object is one of the global objects that are pre-defined and available to components.

5. From the Sites Console, open **Sites** > **We.Train** > **English**.

6. In your file system, navigate to <AEM instance dir>/crx-quickstart/logs.

7. 

8. Open training.log.

9. Find the message logged in **training.log**.

10. 

Extra Credit task – Add logging to the Java helper

1. Using the same concepts as the previous tasks, enable logging using the Java Helper.

2. Remember you will have to modify the contents of site-nav.html to call the Java helper.

10.5 Component Dialog Boxes

Dialog boxes are small temporary windows that are used to enable the user to provide additional inputs. These inputs are either used to perform some action, or are stored in the repository.

A dialog box in Adobe Experience Manager is similar to other dialog boxes that you have used or created in the past—it gathers user input using a 'form', potentially validates it, and then makes that input available for further use (storage, configuration, and so on).

10.5.1 Understanding the types of Dialog Boxes

With Adobe Experience Manager, there are two types of dialog boxes available—the classic dialog box, and the touch-optimized dialog box. For a better user experience, you would use the touch-optimized dialog box.

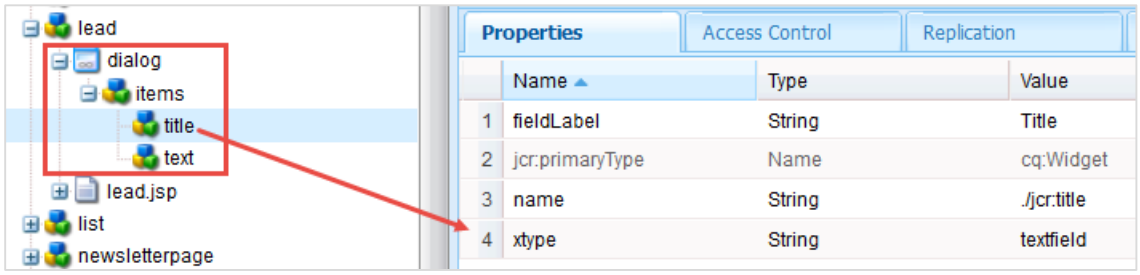
Both dialog boxes use different libraries so you need to create them separately.

10.5.2 Classic UI Dialog Box

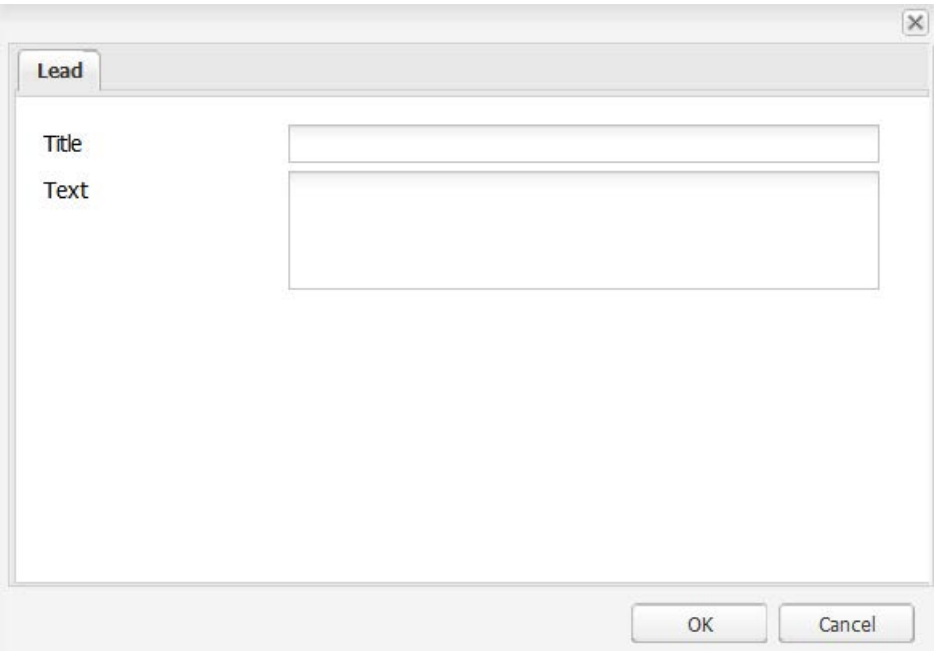
This is the default dialog box that is used in the absense of a touch-optimimzed dialog box. The Classic UI uses a widget library called ExtJS. ExtJs is a cross-browser JavaScript library for building interactive web applications. It provides UI elements that work across all the most important browsers and allow the creation of desktop-grade UI experiences. The popular ExtJS JavaScript framework gives developers the ability to easily create Rich Internet Applications (RIA) using Ajax. It includes:

- High-performance, customizable UI widgets
- A well-designed and extensible component model
- An intuitive, easy-to-use API

For example, a simple classic dialog box would have the following structure and properties:



The dialog box for the above would look like this:

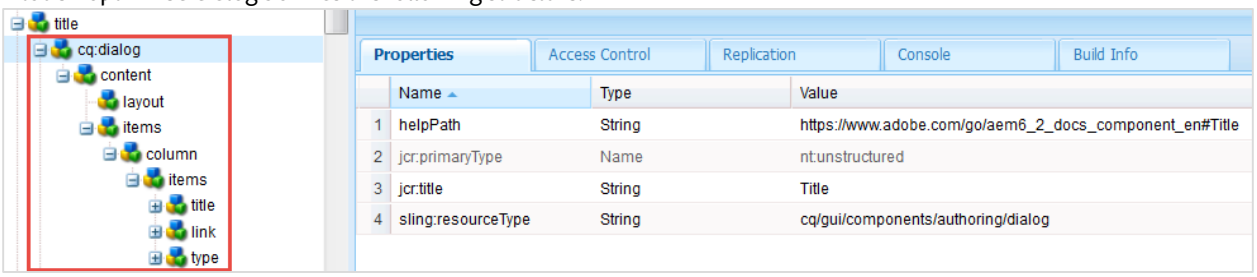


10.5.3 Touch-Optimized UI Dialog Boxes

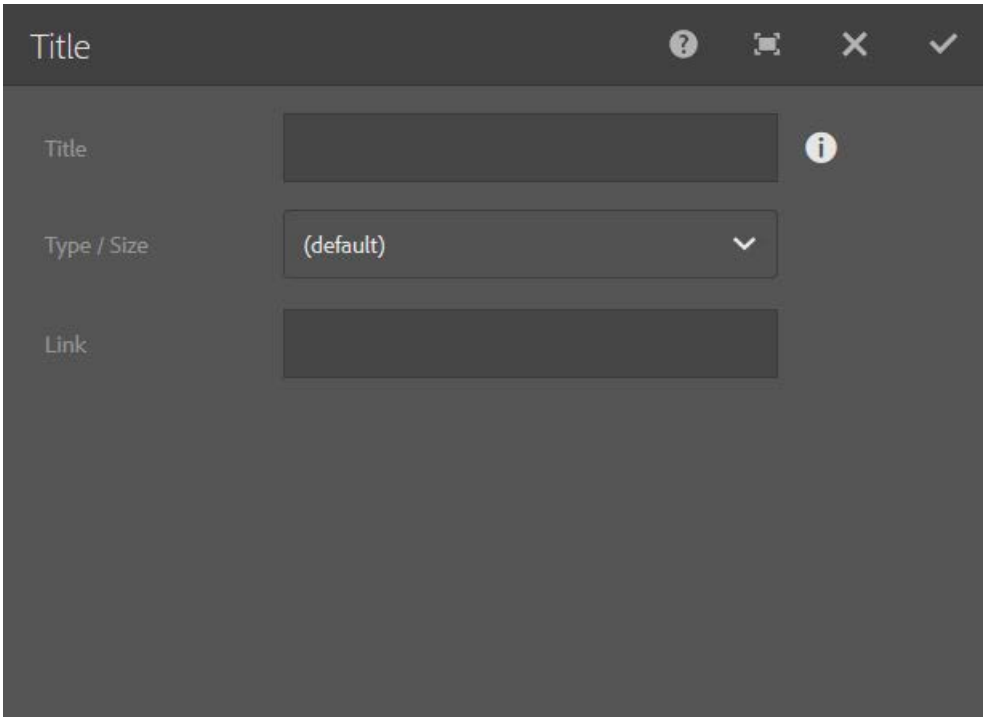
Touch-optimized UI makes use of Granite.js. All the elements you can use for creating the dialog boxes are saved in the **/libs/granite/ui/components/foundation/form** directory. The root node of a dialog box should extend **cq/gui/components/authoring/dialog**.

A touch-optimized UI dialog box is defined by using nodes of type **nt:unstructured**. To define the type of control used, you need to set the node's **sling:resourceType** property. For example, to define a text field on a Touch UI dialog, set the **sling:resourceType** property to **granite/ui/components/foundation/form/textfield**.

A touch-optimized dialog box has the following structure:



The dialog box would appear as follows:



Take a look at what each node means:

- The **cq:dialog** node produces the outer border and the title bar.
- The **content** node provides the container holding the dialog box content.
- The **layout** node defines the layout, in this case, "fixed columns".
- The **column** node defines the container for the widgets.
- The **items** node is the parent of the input forms (widgets).

Here's a description of a few of the properties of the widget:

- **jcr:primaryType**: Defines the Node type, in this case, nt:unstructured
- **fieldLabel**: Tells the author what to do, in this case, "Title".
- **fieldDescription**: Gives the author more information, in this case, "Leave empty to use the page title".
- **name**: Tells the JS code that backs this widget what property name to write, in this case, a property named 'title'.
- **sling:resourceType**: Defines the type of input form, in this case, a 'textfield' - single line, alphanumeric

10.5.4 Difference Between Granite UI and Classic UI Dialog Boxes

Granite UI (Touch-optimized UI)	Classic UI
Defined as nt:unstructured node with resourceType property	Defined as cq:Dialog node with xtype property
Node structure describes actual fields of the dialog box	Contains node structure defining Widgets (cq:widgets)

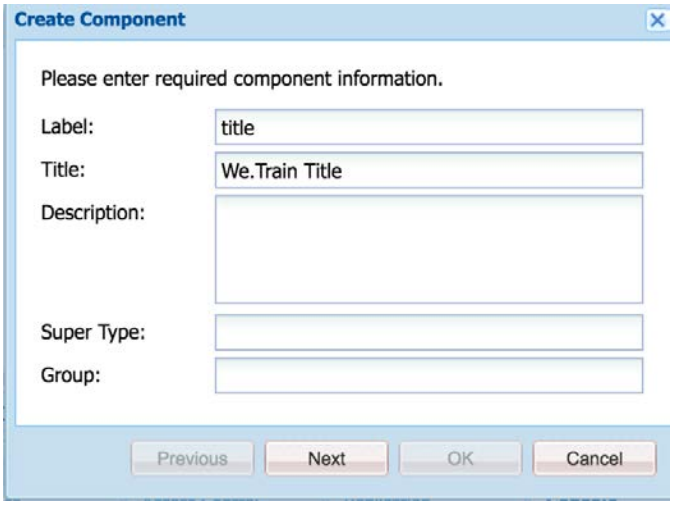
Dialog boxes are rendered on the server-side (as Sling components), based on their content structure	Dialog boxes are rendered on the client-side out of a JSON object coming from the server
To customize a component, you only need to know how to develop components	To customize, you need to know and use the ExtJS code style
NOTE: By default, if a component has no touch-optimized dialog box defined, then the classic UI dialog box is used.	

A quick glance at the following table summarizes the two UIs.

	Touch-optimized UI	Classic UI
Framework	Granite UI	EXTJS
Rendering	resource types (server-side)	xtypes (client-side)
Dialog node	cq:dialog	dialog
Terminology	dialog / fields	dialog / widgets

10.6 Lab Activity - III

Task – Create a Title Component

1. Using CRXDE Lite, navigate to /apps/training/components/structure.
2. Right-click on /apps/training/components/structure and select Create... > Create Component.
3. Enter the following values in the dialog:
    a. Label: title
    b. Title: We.Train Title
4. 

5. Click **Next**, and then click **OK**.
6. Save your changes.
7. Rename the default script **title.jsp** to **title.html**.

8. Using the code from the USB Contents, replace the sample code in **title.html**.

title.html

```
<h1 data-sly-use.title="title.js">${title.text}</h1>
```

9. Create a file named **title.js**.
10. Using the code from the USB Contents, paste the code into **title.js**.

title.js

```
"use strict";

use(function () {

    var CONST = {
        PROP_TITLE: "jcr:title",
        PROP_TAG_TYPE: "type"
    }

    var title = {};

    // The actual title content
    title.text = properties.get(CONST.PROP_TITLE)
            || pageProperties.get(CONST.PROP_TITLE)
            || currentPage.name;

    return title;

});
```

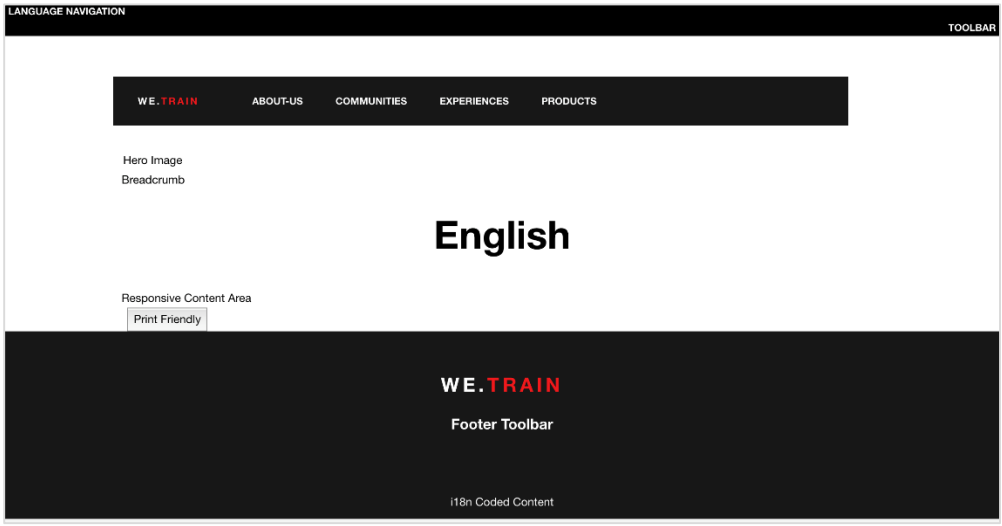
11. Using the code from the USB Contents, modify the code in **body.html**.

body.html

```
<div class="container we-Container--main">
    <div class="root responsivegrid">
        <div class="aem-Grid aem-Grid--12 aem-Grid--default--12  ">
            <div class="header aem-GridColumn aem-GridColumn--default--12"
data-sly-include="header.html"></div>
            <div class="hero-image image parbase aem-GridColumn aem-
GridColumn--default--12" style="padding-top: 150px;">
                <div>Hero Image </div>
```

```
                <div class="responsivegrid aem-GridColumn aem-GridColumn--
default--12">
                    <div class="aem-GridColumn aem-GridColumn--default--12">
                        <div class="row">
                                                <div>Breadcrumb</div>
                            <div class="we-Header">
                                <div data-sly-resource="${'title' @
resourceType='training/components/structure/title'}"></div>
                                                </div>
                            <div>Responsive Content Area</div>
                        </div>
                    </div>
                    <form class="page__print">
                        <input value="Print Friendly" type="submit" />
                    </form>
                    <div class="footer aem-GridColumn aem-GridColumn--
default--12" data-sly-include="footer.html"></div>
                </div>
            </div>
        </div>
    </div>
</div>
```

12. Using the Sites Console, navigate to **Sites** > **We.Train** > **English**.
13. Open the **English** page.

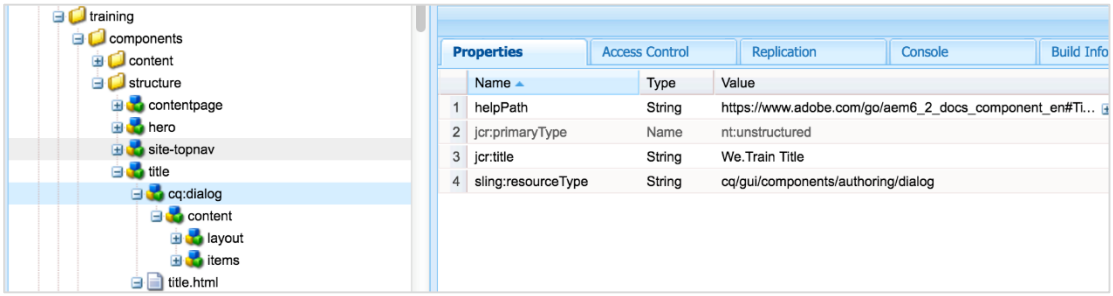


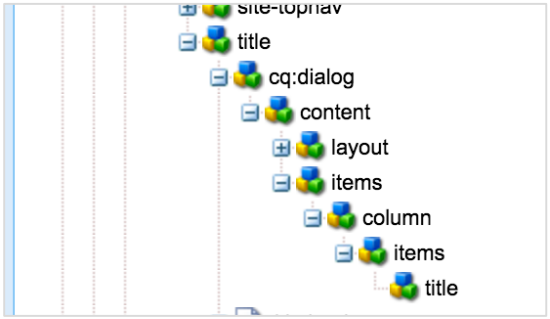
Task – Create a dialog box for the Title Component

Now we have a title component that displays the default page title, but no way to customize the test of the title. We need to add a component dialog box to the component so that the author can enter a custom title.

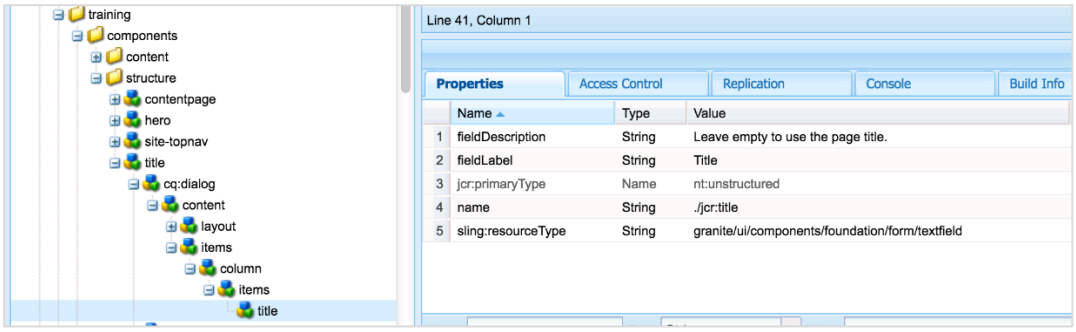
1. Navigate to /libs/wcm/foundation/components/title.
2. Copy the **cq:dialog** node.
3. Navigate to /apps/training/components/structure/title.
4. Paste the copied node.

Tip: the Touch-optimized dialog box must be named **cq:dialog** and it is case sensitive.

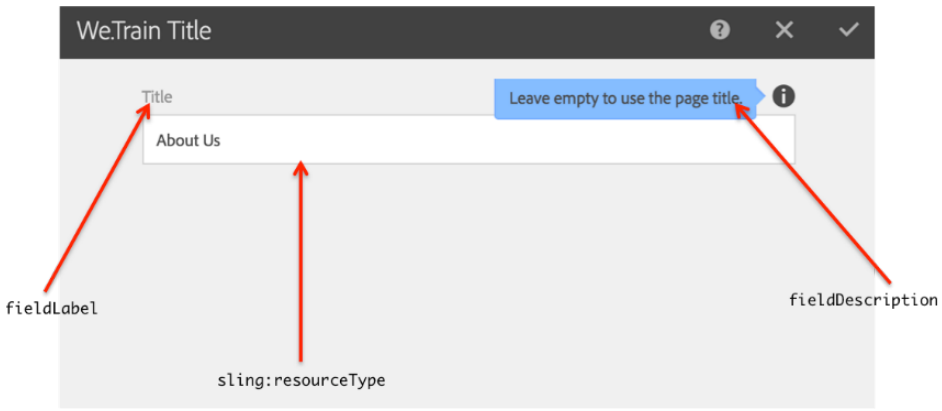
5. Select the **cq:dialog** node.
6. Modify the **jcr:title** property to be **We.Train Title**.
7. 

8. Navigate to the lowest items node under **cq:dialog**.
9. Delete the **type** node.
10. 

11. Save your changes.
12. Select the **title** node.
13. Take notice of the properties on the node:

14. 

- **fieldLabel**: Label of the input field
- **fieldDescription**: Text when author asks for extra information
- **name**: Name of the property the javascript will write when the author clicks OK in the dialog box
- **sling:ResourceType**: Definition of the input field type



10.7 Using EditConfig to Enhance a Component

cq:EditConfig is used to configure content input actions when the dialog box is not in control. For example:

- Drag and drop from the Content Finder
- In-place editing
- Refresh of a dialog box or page after an author action

To add in-place editing functionality, the following are required:

- Node of type cq:editConfig that is a child node to the cq:component node
- cq:InplaceEditing node

Similarly, to add drag-and-drop functionality from the Content Finder, the following are required:

- Node of type **cq:editConfig** that is a child node to the **cq:component** node
- Configuration node that is a child of the **cq:EditConfig** node. This node defines what action you are configuring.

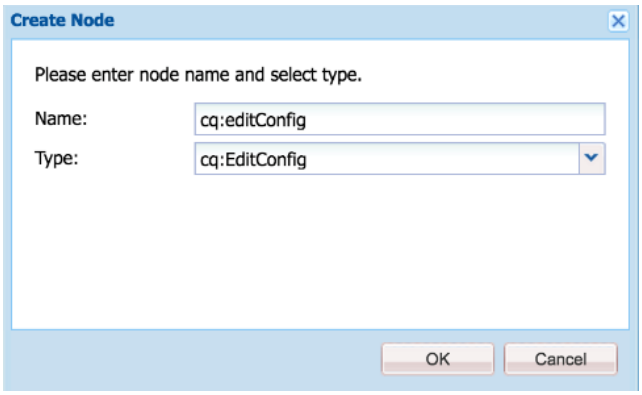
- Asset node that is a child of the **cq:dropTargets** node. This node defines what types of assets the paragraph will accept. In this example, assets of type image.

10.8 Lab Activity - IV

Task – Create editConfig for the Title Component

Now we need to create the editConfig node to define the inline editor.

1. Right-click on the **title** component.
2. Select Create… > Create Node.
3. Enter the following in the dialog box:
   - Name: cq:editConfig
   - Type: cq:EditConfig
4. **Tip:** The node name is a reserved word. It must be named cq:editConfig and it is case sensitive.



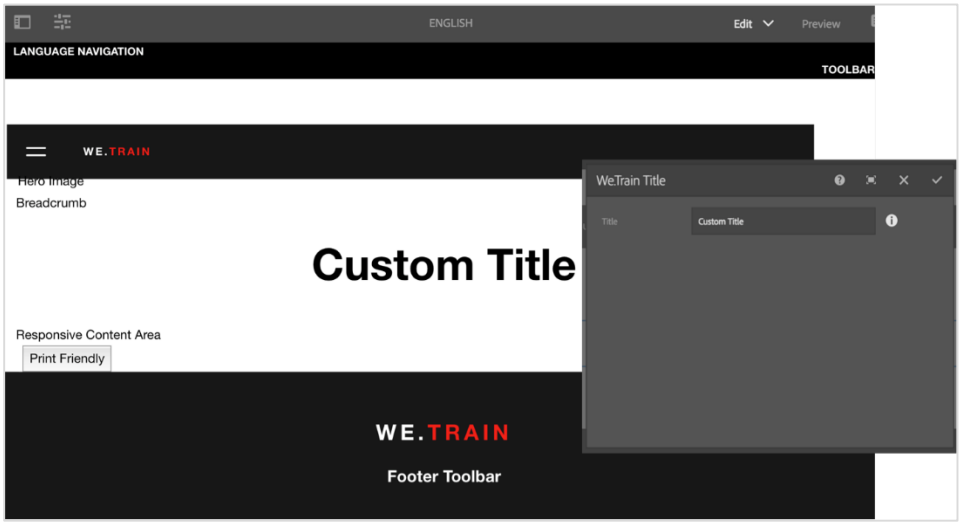
5. Right-click on the **cq:editConfig** node and select **Create…** > **Create Node**.
6. Enter the following in the dialog box:
   - Name: cq:inplaceEditing
   - Type: cq:InplaceEditingConfig
7. **Tip:** The node name is a reserved word. It must be named cq:inplaceEditing and it is case sensitive
8. Add the following properties to the **cq:inplaceEditing** node:

Name	Type	Value
Accept	Boolean	True
editorType	String	title

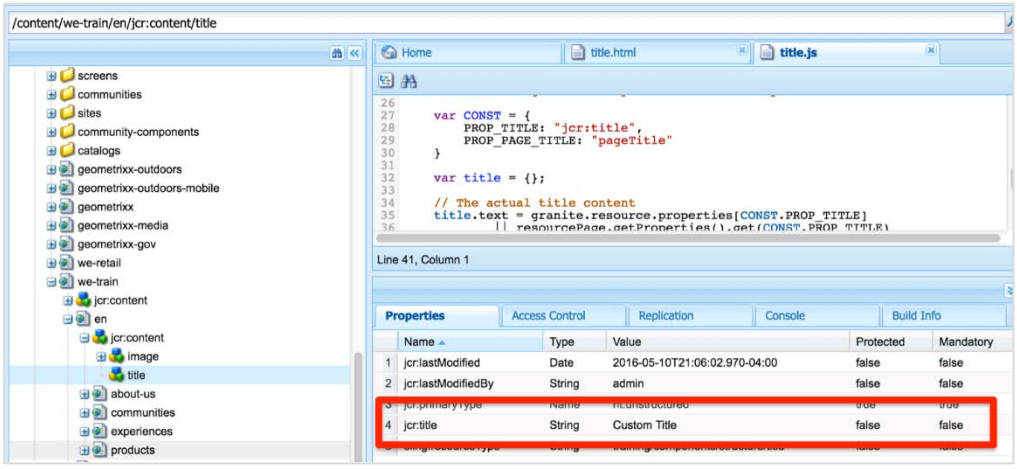
9. Right-click on the **cq:inplaceEditing** node and select **Create…** > **Create Node**.
10. Enter the following in the dialog:
    - Name: config
    - Type: nt:unstructured

Task – Use the Title Dialog

We are now ready to test out our new dialog and enter some custom content.

1. Open the **English** page again, now you will be able to edit the **Title** component.
2. Double-click on the **Title** component and enter a custom title and click on the check mark.
3. 

Now let's see what happened in the repository.

4. Using CRXDE Lite navigate to /content/we-train/en/jcr:content.
5. Notice that a **title** node has appeared.
6. Look at the properties on the **title** node. When you entered a custom title it was stored on the title node in the **jcr:title** property.
7. 

10.9 Use Design Dialog Boxes for Global Content

A Design(er) can be used to enforce a consistent look and feel across your website, as well as share global content. This global content is editable when using a Design dialog box. So once again, the many pages that use the same Design(er)

will have access to this global content. The process to create a Design dialog box is almost identical to creating a 'normal' dialog box; the only difference being the name of the dialog box itself (that is, dialog box vs. design dialog box).

- Global content
- Stored in */etc/designs* instead of the local node of the page
- Root node of the design is of type `cq:Page`
- Child node jcr:content of type cq:PageContent
- sling:resourceType = wcm/designer

The Design(er) values can be accessed by the currentStyle object provided from the global.jsp. This is different from using the properties object as we have done so far. Design dialog boxes are almost identical to component dialog boxes with the following exceptions:

- **name:** design_dialog instead of dialog
- **content storage:** /etc/designs instead of /content/website
- **availability:** design mode instead of edit mode

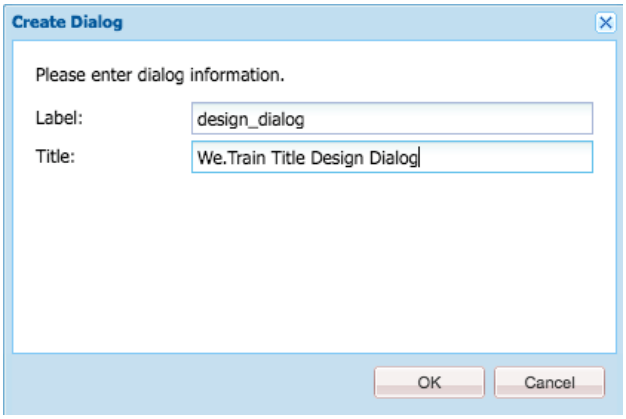
NOTE: Currently, there is no Touch UI version of the design dialog.

10.10 Lab Activity - V

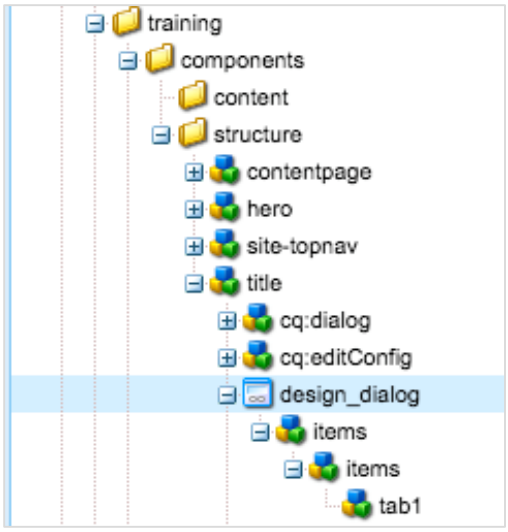
Task – Add a Design Dialog

As we discussed, you can define design elements that affect all instances of a component. We are going to demonstrate using the design to set styling on the title.

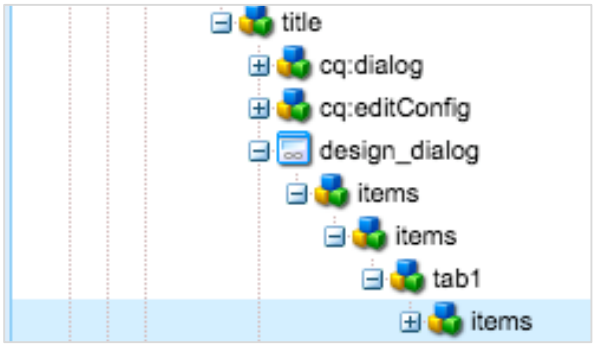
1. Using CRXDE Lite, right-click on the **title** component and select **Create Dialog**.
2. Enter the following values and click **OK**:
   - Label: design_dialog
   - Title: We.Train Title Design Dialog



3. Save your changes.
4. Notice the structure that the dialog wizard created in the repository.

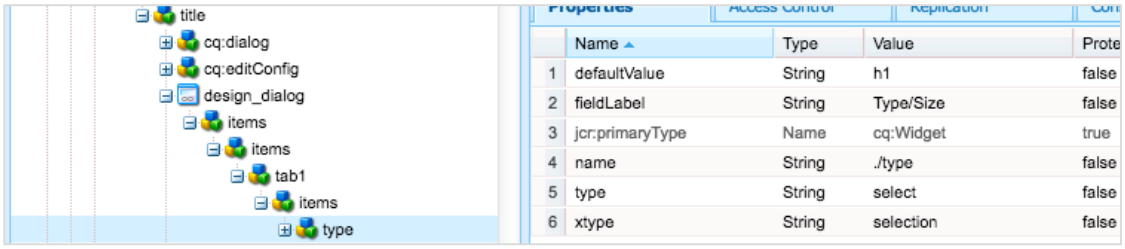


5. Navigate to ~/design_dialog/items/items/tab1.
6. Right-click **tab1** and select **Create…** > **Create Node**. Enter the following values:
   - Name: items
   - Type: cq:WidgetCollection



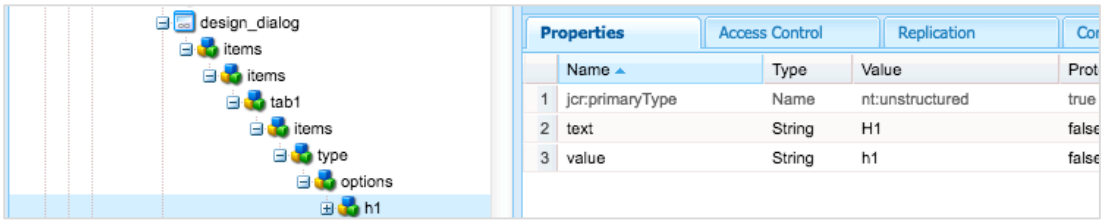
7. Click **OK**.
8. Create a child node to the items node you just created. Enter the following values and click **OK**:
   - Name: type
   - Type: cq:Widget
9. Define the following five properties on the type node that you just created:

Name	Type	Value
defaultValue	String	h1
fieldLabel	String	Type/Size
name	String	./type
type	String	select
xtype	String	selection



10. Create a child node to the type node. Enter the following values:
    - Name: options
    - Type: cq:widgetCollection
11. Create four child nodes to the options node that you just created. These will be the options in the styling dropdown list. Enter the following values:
    - Name: h1
    - Type: nt:unstructured
12. Properties:

Name	Type	Value
text	String	H1
value	String	h1

13.
14. 

- Name: h2
- Type: nt:unstructured

15. Properties:

Name	Type	Value
text	String	H2
value	String	h2

16.

- Name: h3
- Type: nt:unstructured

17. Properties:

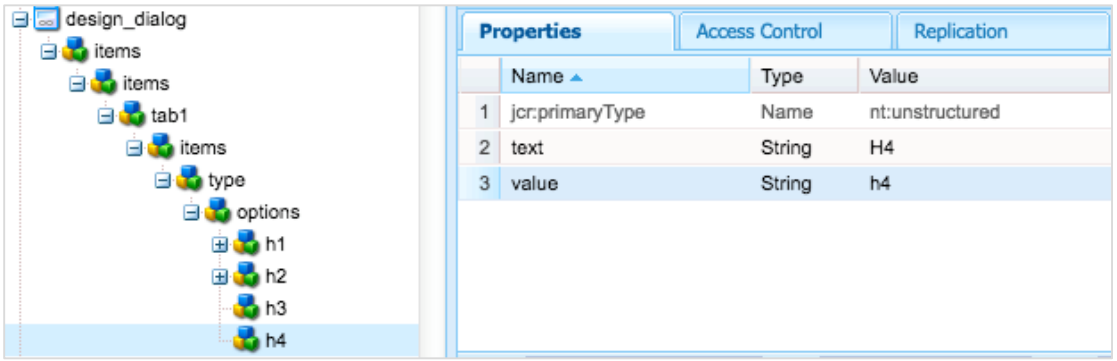
Name	Type	Value
text	String	H3
value	String	h3

- Name: h4
- Type: nt:unstructured

18. Properties:

Name	Type	Value
text	String	H4
value	String	h4

19. Your design dialog should look like this:



Now we need to modify the title.html and title.js to use the information entered in the design dialog.

Task – Modify Title component code to use design dialog

1. Using the code from the USB Contents, replace the code in **title.js**. This will allow the title.js to return the styling specified by the author.

title.js

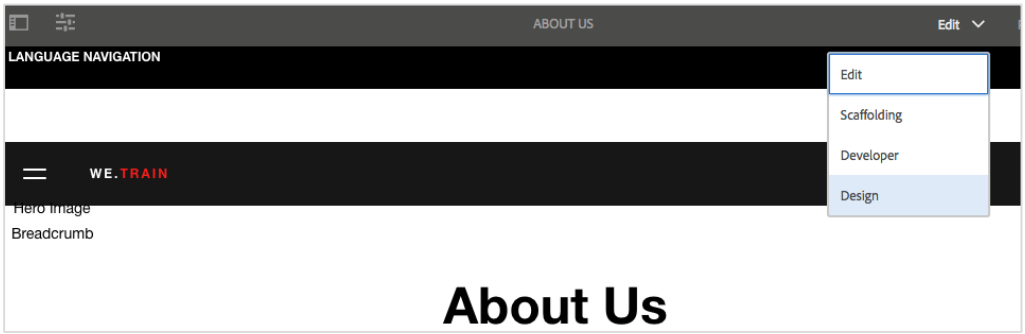
```
"use strict";
use(function () {
    var CONST = {
        PROP_TITLE: "jcr:title",
        PROP_TAG_TYPE: "type"
    }

    var title = {};

    // The actual title content
    title.text = properties.get(CONST.PROP_TITLE)
            || pageProperties.get(CONST.PROP_TITLE)
            || currentPage.name;

    title.type = currentStyle.get(CONST.PROP_TAG_TYPE)
             || "h1";
    return title;
});
```

2. Using the code from the USB Contents, replace the code in **title.html**. This will allow the **title.html** to output the styling specified by the author.

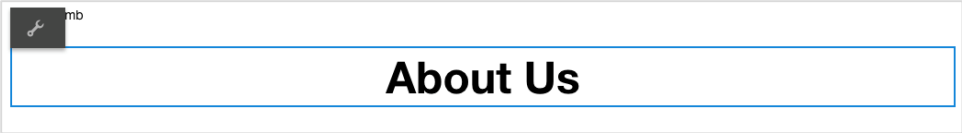
```
<h1 data-sly-use.title="title.js" data-sly-
element="${title.type}">${title.text}</h1>
```

Now let's test our design dialog.

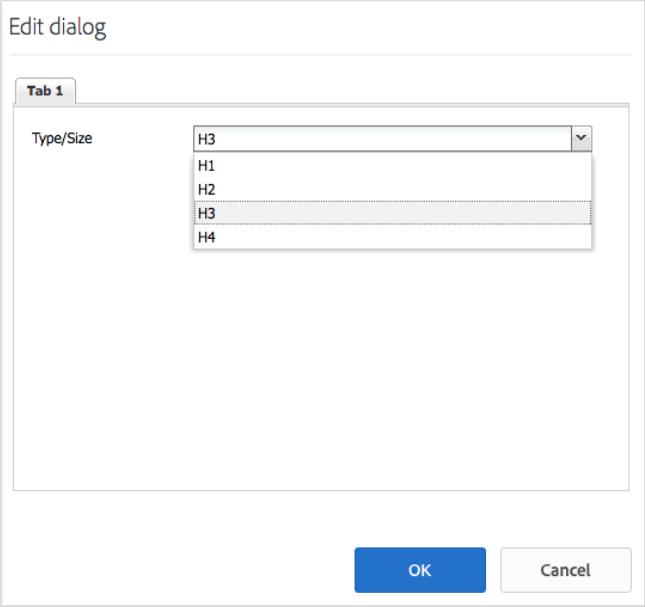
3. Using the Sites Console, navigate to **Sites** > **We.Train** > **English** > **About Us**.
4. Select **Design** from the mode dropdown to enter design mode.



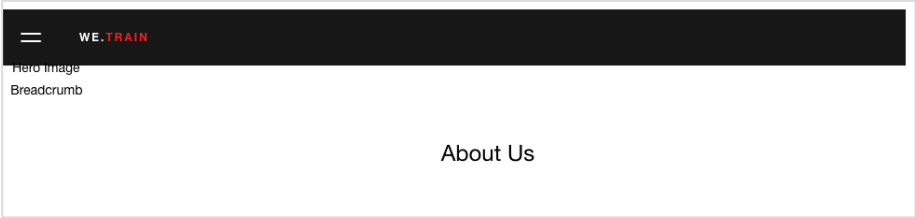
5. Click on the **Title** Component.



6. Click on the wrench.



7. Choose **H3** from the dropdown and click **OK**.



8. Navigate to other pages. Notice that the choice you made for the Title component is reflected on every page.

10.11 The FileUpload Form Field

The FileUpload Form field is a field component that is used to upload files. You will see in the lab activities that follow, how to create a component that accepts binary files as inputs. You can upload an image by both dragging and dropping the image from the Asset browser directly onto the component or through its Configure dialog box.

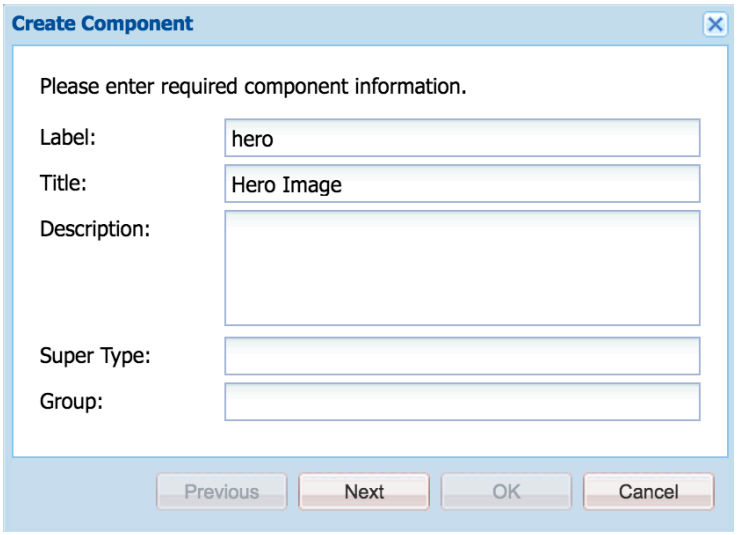
BEST PRACTICE: Use the Assets feature to include the assets to their repositories.

10.12 Lab Activity - VI

Task – Create an initial Hero Image Component

1. Using CRXDE Lite, navigate to /apps/training/components/structure.

2. Right-click on /apps/training/components/structure and select Create... > Create Component.
3. Enter the following values in the dialog:
   - Label: hero
   - Title: Hero Image



4.
5. Click **Next,** and then click **OK**.
6. Save your changes.
7. Rename the default script **hero.jsp** to **hero.html**.
8. Using the code from the USB Contents, replace the sample code in **hero.html**.

hero.html

```
<div data-sly-use.hero="hero.js">
    <strong>${hero.text}</strong>
</div>
```

9. Create a file named **hero.js**, under the hero component.
10. Using the code from the USB Contents, paste the code into **hero.js**.

hero.js

```
"use strict";
use(function() {
    var CONST = {
        PROP_TITLE: "jcr:title",
    }

    var hero = {}
```

```
    //Get the title text
    hero.text = properties.get(CONST.PROP_TITLE)
            || pageProperties.get(CONST.PROP_TITLE)
            || currentPage.name;

    return hero;
});
```

If you take a look at the code, you will see that the initial hero.html renders **${hero.text}**. Since we haven't created a dialog yet, it will be displaying the page Title.

11. Navigate to /apps/training/components/structure/contentpage.

12. Using the code from the USB Contents, replace the code in **body.html**. We need to add a line in the **body.html** that will include the Hero component on the page.

body.html

```
<div class="container we-Container--main">
    <div class="root responsivegrid">
        <div class="aem-Grid aem-Grid--12 aem-Grid--default--12  ">
            <div class="header aem-GridColumn aem-GridColumn--default--12"
data-sly-include="header.html"></div>
            <div class="hero-image image parbase aem-GridColumn aem-
GridColumn--default--12" style="padding-top: 150px;">
                <div data-sly-resource="${'hero' @
resourceType='training/components/structure/hero'}"></div>
                <div class="responsivegrid aem-GridColumn aem-GridColumn--
default--12">
                    <div class="aem-GridColumn aem-GridColumn--default--12">
                        <div class="row">


                                                <div class="aem-
breadcrumb">Breadcrumb</div>
                            <div class="we-Header" data-sly-
resource="${'title' @
resourceType='training/components/structure/title'}"></div>
                            <div>Responsive Content Area</div>
                        </div>
                    </div>
                    <form class="page__print">
                        <input value="Print Friendly" type="submit" />
                    </form>
```

```
                    <div class="footer aem-GridColumn aem-GridColumn--
default--12" data-sly-include="footer.html"></div>
                </div>
            </div>
        </div>
    </div>
</div>
```

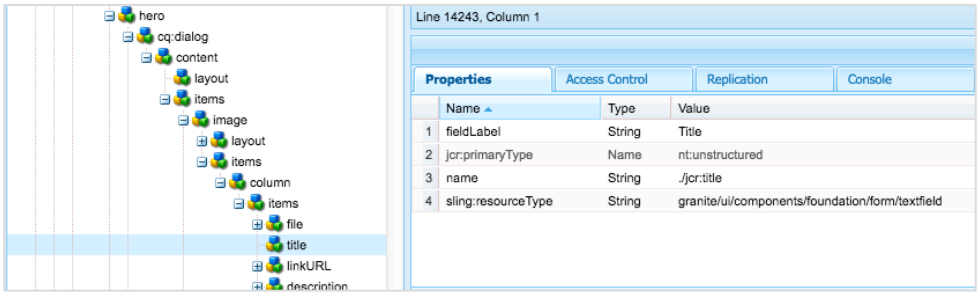
13. Using the Sites Console, navigate to **Sites** > **We.Train** > **English**. You will notice that the page Title shows up.

We need to define a component dialog so the Author can enter a custom string for the Hero Component.

Task – Create a Dialog for Author Input

Because the Hero component deals with an image, we need a dialog structure that knows how to deal with binary data (files). So we are going to copy the dialog from the Foundation Image component.

1. Using CRXDE Lite, navigate to /libs/wcm/foundation/components/image.
2. Copy the **cq:dialog** node.
3. Paste to /apps/training/components/structure/hero.
4. Expand the dialog and examine the structure.



Notice that the title node. The name property specifies that the jcr:title property be written. Compare this to the property that title.js is capturing. This will be the text element for our Hero component.

Task – Modify the text feature of the Hero Component

In order to use our dialog, to modify the jcr:title property, we need to define editConfig. We are also going to add the ability to drag and drop images from the AEM DAM into the Hero component.

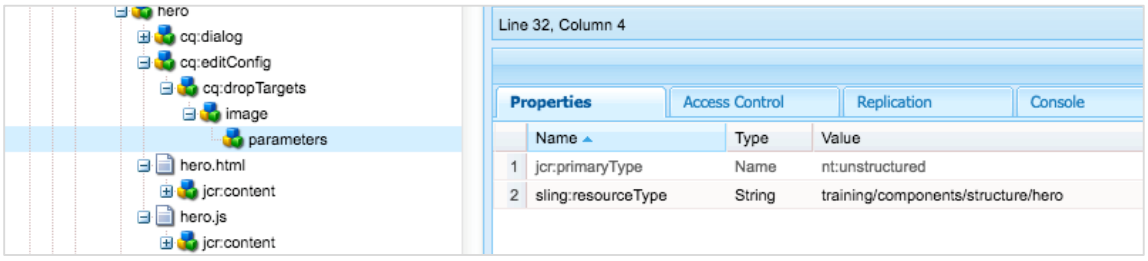
1. Using CRXDE Lite, right-click on /apps/training/components/structure/hero.
2. Select **Create…** > **Create Node**. Enter the following values:
   - Name: cq:editConfig
   - Type: cq:EditConfig
3. Click **OK** and **Save**.

4. Right-click on the **cq:editConfig** node and select **Create…** > **Create Node**. Enter the following values:
   - Name: cq:dropTargets
   - Type: nt:unstructured
5. Right-click on the **cq:dropTargets** node and select **Create…** > **Create Node**. Enter the following values:
   - Name: image
   - Type: cq:DropTargetConfig
6. Define the following three properties on the image node.

Name	Type	Value
accept	String[]	image/.*
groups	String[]	media
propertyName	String	./fileReference

7. Right-click on the image node and select **Create…** > **Create Node**. Enter the following values:
   - Name: parameters
   - Type: nt:unstructured
8. Define the following properties on the parameters node:

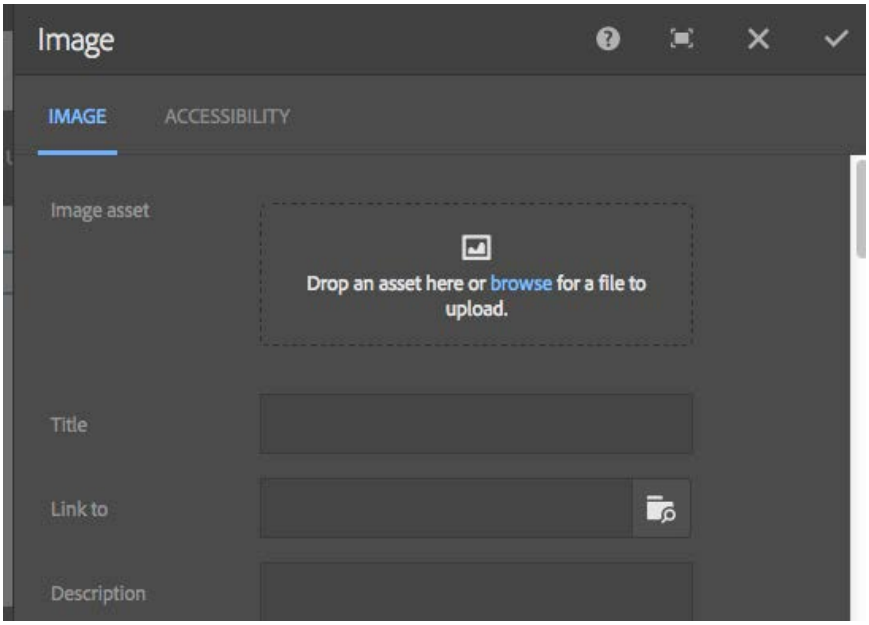
Name	Type	Value
sling:resourceType	String	training/components/structure/hero



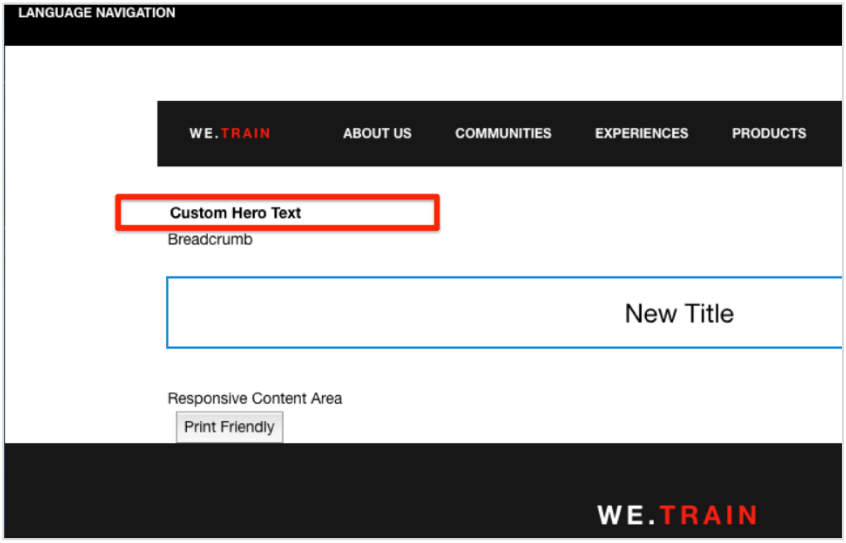
Task – Test the Hero Component

Now to test the Hero Component.

1. Using the Sites Console, navigate to **Sites** > **We.Train** > **English**.
2. Double-click on the **Hero** component and click on the wrench.



3. Enter a value in **Title** and click the check mark.



Task – Modify the Hero Component to display an Image

1. Using CRXDE Lite, modify the contents of **/apps/training/components/structure/hero/hero.js** using the code from the USB Contents. This will add the functionality to capture an uploaded image.

hero.js

```
"use strict";
use(function() {
    var CONST = {
        PROP_TITLE: "jcr:title",
```

```
        PROP_REF_HERO_IMAGE: "fileReference",
        PROP_UPLOAD_HERO_IMAGE: "file"
    }

    var hero = {}

    //Get the title text
    hero.text = properties.get(CONST.PROP_TITLE)
            || pageProperties.get(CONST.PROP_TITLE)
            || currentPage.name;



    //Check for file reference from the DAM
    var image = properties.get(CONST.PROP_REF_HERO_IMAGE, String.class);
    if(image == "undefined"){
       //Check for file upload
       var res = resource.getChild("file");
       if(res != null){
              image = res.getPath();
       }
    }
    if(image != "undefined"){
       hero.style = "background-image:url(" + image + ");";
    }
    return hero;
});
```

2. Modify the contents of **hero.html** using the code from the USB Contents. This will add the functionality to display an uploaded image.

hero.html

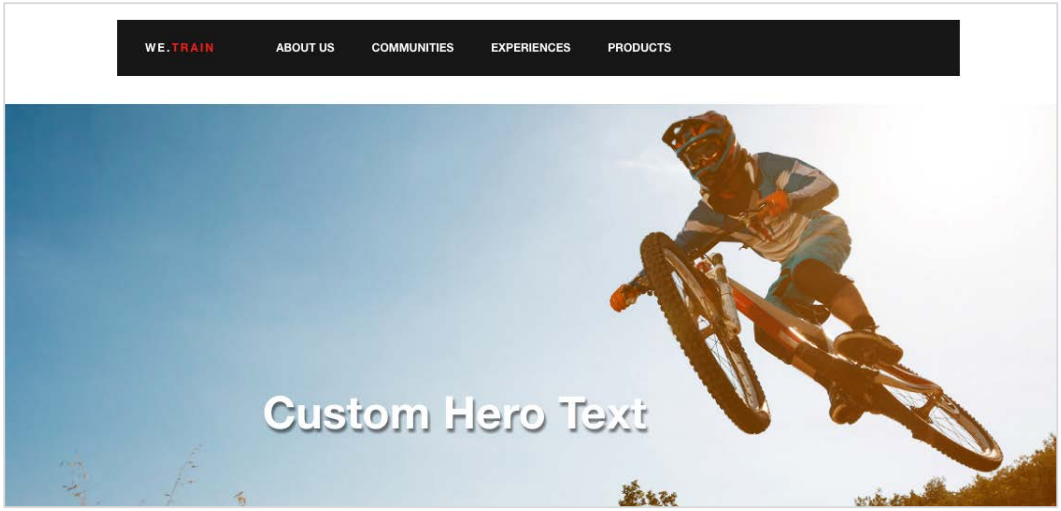
```
<div data-sly-use.hero="hero.js" class="we-HeroImage width-full ratio-16by9"
style="${hero.style @ context='styleString'}">
        <div class="container cq-dd-image">
            <div class="we-HeroImage-wrapper">
                <strong class="we-HeroImage-title">${hero.text}</strong>
            </div>
        </div>
    </div>
```

3. Modify the contents of **body.html** using the code from the USB Contents.

body.html

```
<div class="container we-Container--main">
    <div class="root responsivegrid">
        <div class="aem-Grid aem-Grid--12 aem-Grid--default--12  ">
            <div class="header aem-GridColumn aem-GridColumn--default--12"
data-sly-include="header.html"></div>
            <div class="hero-image image parbase aem-GridColumn aem-
GridColumn--default--12">
                <div data-sly-resource="${'hero' @
resourceType='training/components/structure/hero'}"></div>
                <div class="responsivegrid aem-GridColumn aem-GridColumn--
default--12">
                    <div class="aem-GridColumn aem-GridColumn--default--12">
                        <div class="row">
                                                <div class="aem-
breadcrumb">Breadcrumb</div>
                            <div class="we-Header" data-sly-
resource="${'title' @
resourceType='training/components/structure/title'}"></div>
                            <div>Responsive Content Area</div>
                        </div>
                    </div>
                    <form class="page__print">
                        <input value="Print Friendly" type="submit" />
                    </form>
                    <div class="footer aem-GridColumn aem-GridColumn--
default--12" data-sly-include="footer.html"></div>
                </div>
            </div>
        </div>
    </div>
</div>
```

4. Using the Sites Console, open **Sites** > **We.Train** > **English**
5. Click on the **Hero** Component to open the dialog.
6. Upload one of the images from the USB Contents into the dialog.



7. Try the same for a few other pages.

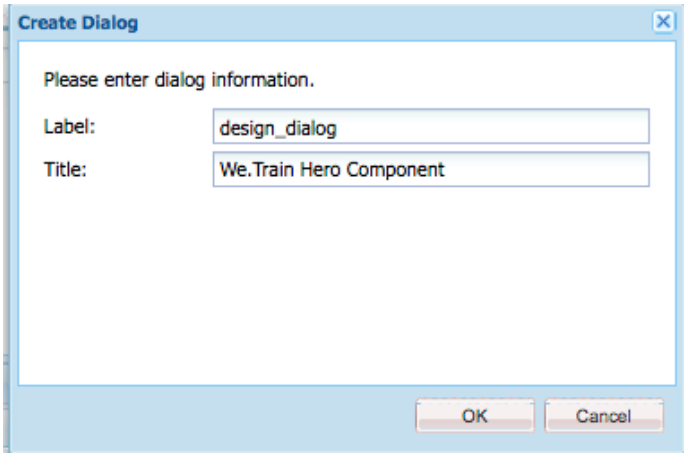
Extra Credit Task – Drag and Drop images into Hero component

1. Upload a few of the supplied images into the DAM
2. Drag and drop an image onto the Hero component

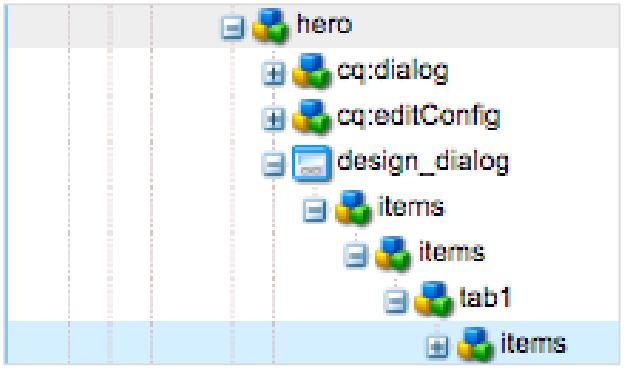
Task – Create a Design Dialog for the Hero Component

Now to add a design dialog that allows the author to specify the styling of the text overlaying the Hero image.

1. Using CRXDE Lite, navigate to /apps/training/components/structure/hero.
2. Right-click on **hero** and select **Create ..** > **Create Dialog**.
3. Enter the following values:
   - Label: design_dialog
   - Title: We.Train Hero Component



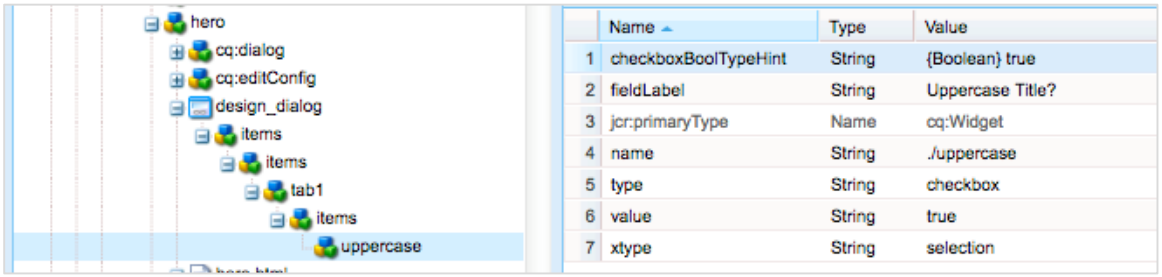
4. **Tip:** The node name is a reserved word. It must be named design_dialog and it is case sensitive
5. Click **OK** and **Save**.
6. Expand the dialog.
7. Right-click **tab1** and select **Create...** > **Create Node**. Enter the following values:
   - Name: items
   - Type: cq:WidgetCollection



8. Click **OK**.
9. Save your changes.
10. Create a child node to the items node you just created. Enter the following values and click **OK**:
    - Name: uppercase
    - Type: cq:Widget
11. Save your changes.
12. Define the following six properties on the type node that you just created:

Name	Type	Value
checkboxBoolTypeHint	String	{Boolean} true
fieldLabel	String	Uppercase Title?
name	String	./uppercase
type	String	checkbox
xtype	String	selection
value	String	true

13. Save your changes.



Now let's modify the code to take advantage of the new information from the design dialog.

14. Using code from the USB Contents, replace the code for **hero.js**. Notice that the new code will extract the styling information from the **currentStyle** object. The javascript behind the design dialog will store the information entered in the design dialog in the design. This design information populates the **currentStyle** object.

hero.js

```
"use strict";
use(["/libs/wcm/foundation/components/utils/AuthoringUtils.js"], function
(AuthoringUtils) {
    var CONST = {
        PROP_TITLE: "jcr:title",
        PROP_REF_HERO_IMAGE: "fileReference",
        PROP_UPLOAD_HERO_IMAGE: "file",
        PROP_UPPERCASE: "uppercase"
    }

    var hero = {}

    //Get the title text
    hero.text = properties.get(CONST.PROP_TITLE)
            || pageProperties.get(CONST.PROP_TITLE)
            || currentPage.name;

    //Check for file reference from the DAM
    var image = properties.get(CONST.PROP_REF_HERO_IMAGE, String.class);
    if(image == "undefined"){
       //Check for file upload
       var res = resource.getChild("file");
       if(res != null){
              image = res.getPath();
       }
    }
    if(image != "undefined"){
       hero.style = "background-image:url(" + image + ");";
    }
    //Add uppercase design to hero text
    if(currentStyle.get(CONST.PROP_UPPERCASE)){
       hero.uppercase = "text-transform:uppercase;";
    }
    return hero;
});
```

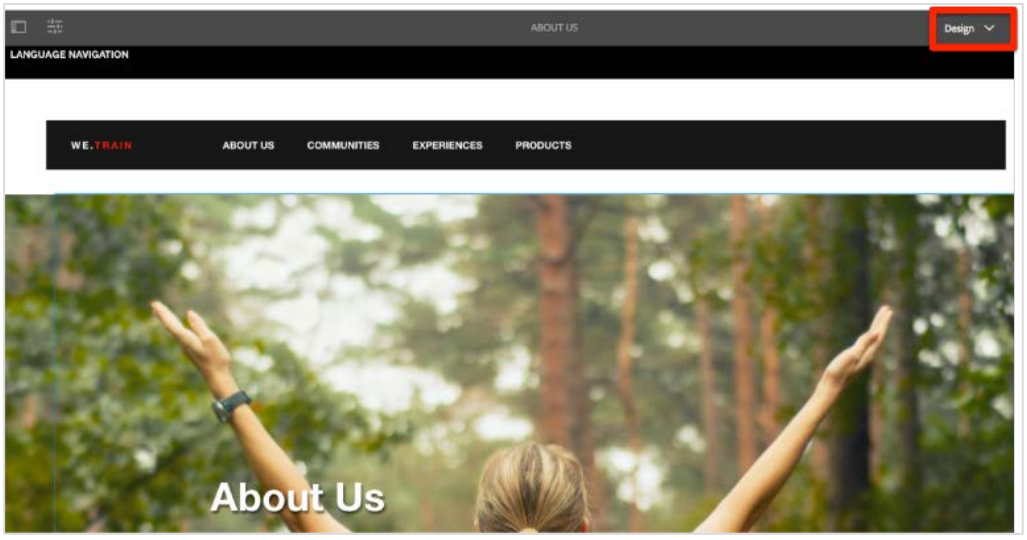
15. Using code from the USB Contents, replace the code for **hero.html**. Notice that the new code will render the text using the styling information returned by **hero.js**.

hero.html

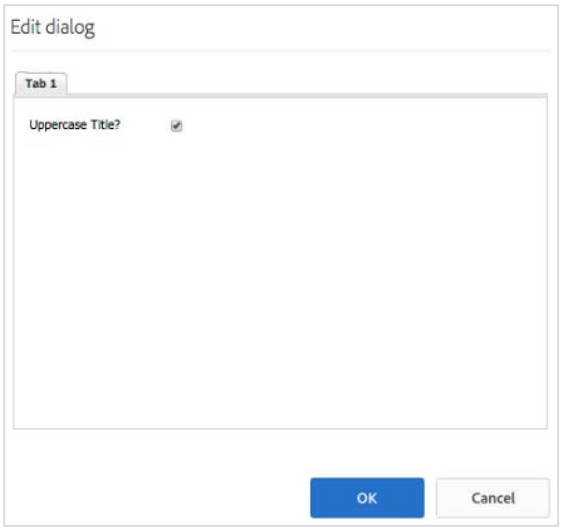
```
<div data-sly-use.hero="hero.js" class="we-HeroImage width-full ratio-16by9"
style="${hero.style @ context='styleString'}">
        <div class="container cq-dd-image">
            <div class="we-HeroImage-wrapper">


                <strong class="we-HeroImage-title" style="${hero.uppercase @
context='styleString'}">${hero.text}</strong>


            </div>
        </div>
    </div>
```

We can now test our design dialog and code.

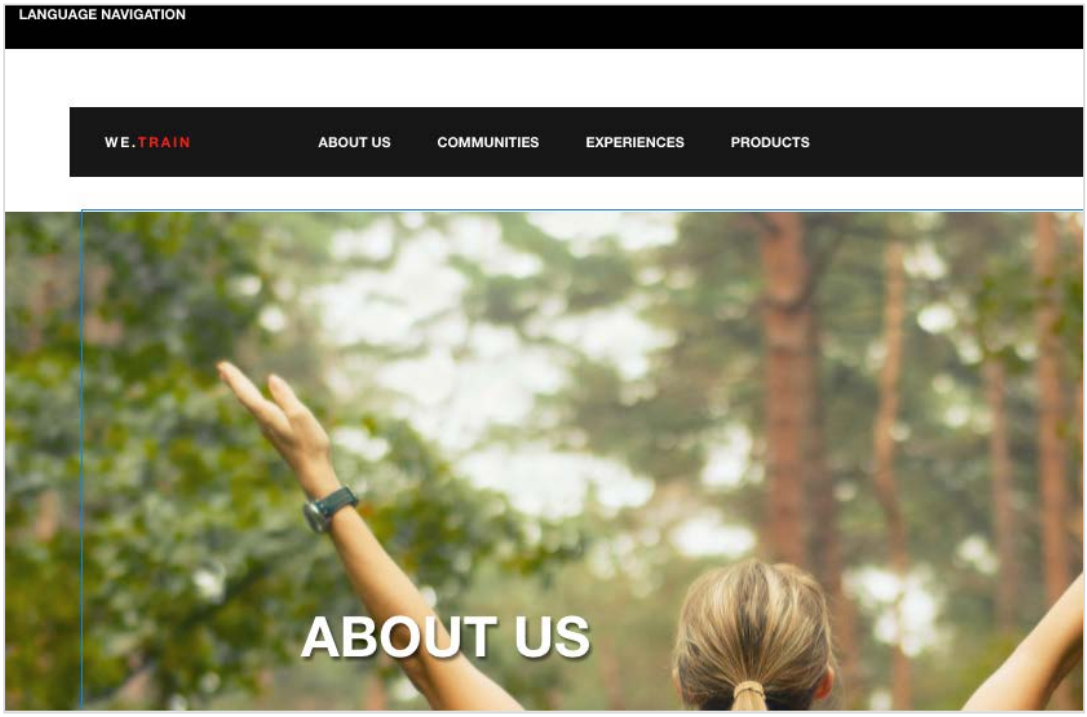
16. Using the Sites Console, navigate to **Sites** > **We.Train** > **English** > **About Us.**
17. Using the mode dropdown, select **Design** mode.



18. Click on the **Hero** component
19. Click on the wrench.



20. Click the **Uppercase** checkbox and click **OK**.

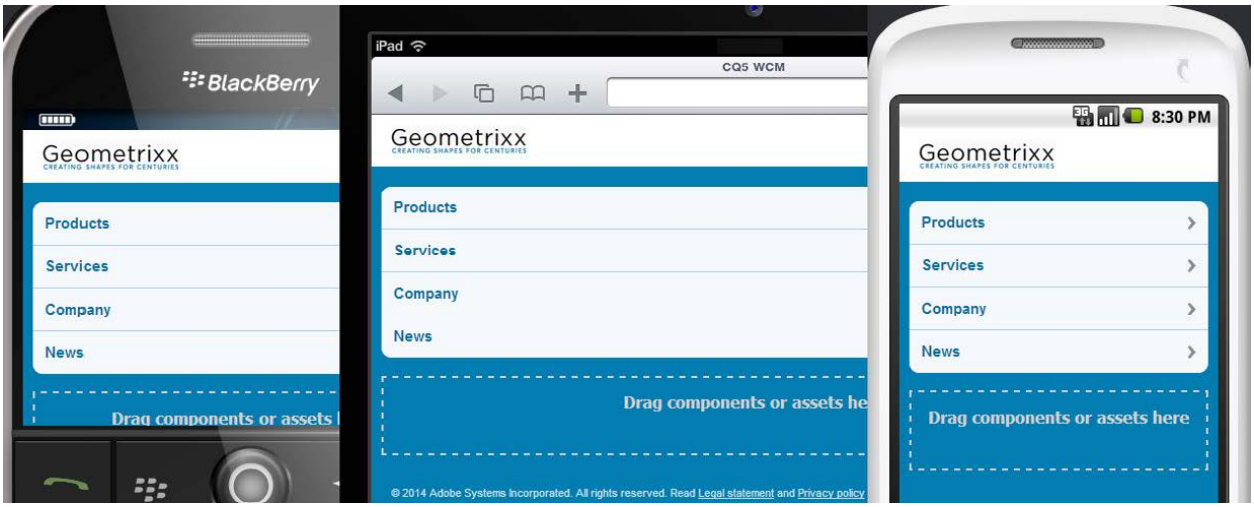


11 The Responsive Grid

The website you created is optimized to view on desktops and laptops. However, your users may want to view your website on handheld devices, such as tablets and mobile phones. Therefore, you need to ensure that the website is optimized to view on all devices to provide your users with a better experience.

Responsive Design for Mobile pages allows you to create sites that provide an optimal viewing experience across various mobile devices. You are provided with emulators to view the site on various devices. There are a number of ways to structure your content to provide an interesting experience to desktop and mobile customers. Two design methodologies are currently popular:

- Separate content for desktop and mobile (where mobile content could be mobile website content and/or mobile apps)
- Responsive Design



Responsive Design is an approach to web design aimed at implementing sites that provide an optimal viewing experience for all website visitors:

- Easy reading and access to content
- Clear and easy navigation
- Minimum of resizing, panning, and scrolling across a wide range of devices (desktop and mobile) with varying:
  - Screen sizes
  - Memory capacity
  - Network speeds
  - CPU speeds

Responsive Design is not a single piece of technology, but rather, a collection of techniques and ideas that allow the site to identify and then respond to the browsing environment or device through which they are being viewed. The three tenets of Responsive Design are:

- Fluid grids
- Flexible images
- CSS3 media inquiries to detect screen resolution

Responsive Design is one of many powerful strategies, but may not be the right solution for your web property. Therefore, a business should not instantly drop plans for a mobile website in favor of the Responsive Design route. Every website has particular, defined objectives. It is important to approach the issue of multi-device experience from a strategic standpoint to ensure that your web goals are met.

Responsive Design works well for sites where users consume content. However, it does not work well when you want your customers to interact with the content. Purchasing applications are complex. Many banks and other businesses that offer mobile apps to sell their products and services do not use Responsive Design because of limitations on the types of things you can do on the website. A popular example is a mobile banking app that allows you to deposit a check by taking a picture of it. The application that creates the 'deposit from a picture' functionality is complicated and often cannot fit into a grid layout.

11.1 Pros and Cons of Responsive Design

Pros

- A single design for all devices to ensure a consistent experience across devices
- A single URL for all devices
- A single code base for your website
- Less expensive to maintain

Cons

- It is hard to get a natural look and feel for desktop devices without messy customization.
- Type is often too small for smartphone users, resulting in pinching and zooming.
- Download times when browsing over a mobile network can be quite long.
- Older devices, without CSS3 support, would still be served with the 'normal' desktop website.
- The user journey for mobile users is typically different from desktop users, especially in retail scenarios. Mobile users would probably want to get right to the end of the journey, whereas desktop users are happier to browse more.

Challenges

- Getting navigation right for smartphones
- Need for new content management workflows
- Need for new image optimization processes

Ideally, you want a blend of Responsive Design with touch-specific elements mixed in, resulting in a true mobile experience.

11.2 Lab Activity

Task – Add the Responsive Grid

The responsive grid is a responsive paragraph system. This will become the free content area where authors can add new content to the page.

1. Using CRXDE Lite navigate to /apps/training/components/structure/contentpage.
2. Using the code from the USB Contents, modify the contents of **body.html**.

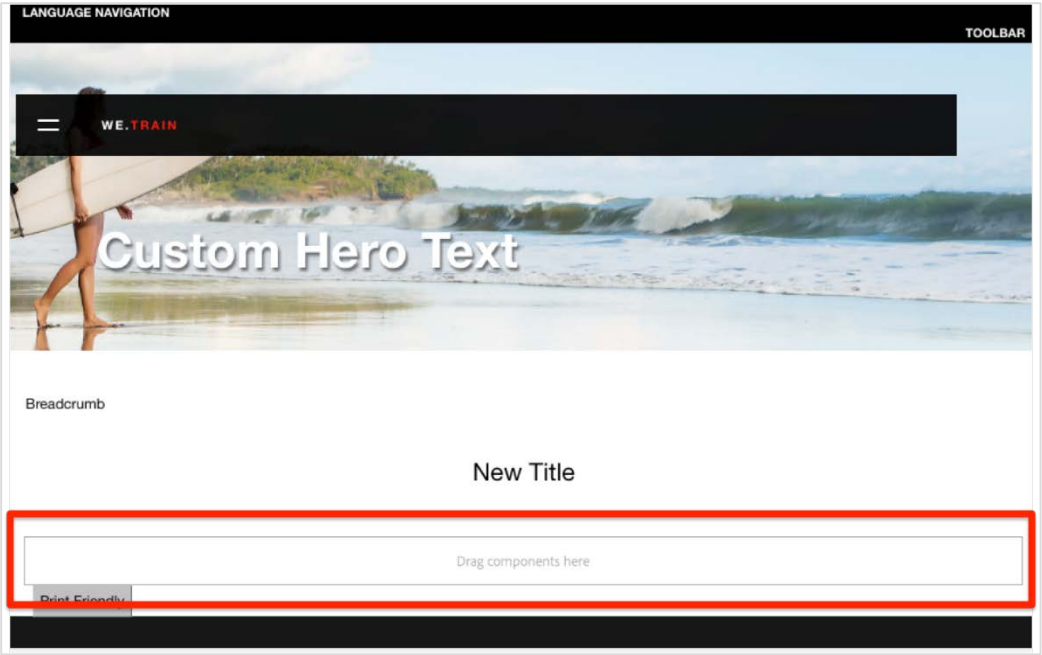
body.html

```
<div class="container we-Container--main">
    <div class="root responsivegrid">
        <div class="aem-Grid aem-Grid--12 aem-Grid--default--12  ">
            <div class="header aem-GridColumn aem-GridColumn--default--12"
data-sly-include="header.html"></div>
            <div class="hero-image image parbase aem-GridColumn aem-
GridColumn--default--12">
                <div data-sly-resource="${'hero' @
resourceType='training/components/structure/hero'}"></div>
                <div class="responsivegrid aem-GridColumn aem-GridColumn--
default--12">
                    <div class="aem-GridColumn aem-GridColumn--default--12">
                        <div class="row">
                                                <div class="aem-
breadcrumb">Breadcrumb</div>
                            <div class="we-Header" data-sly-
resource="${'title' @
resourceType='training/components/structure/title'}"></div>
                            <div data-sly-resource="${'responsivegrid' @
resourceType='wcm/foundation/components/responsivegrid'}"></div>
                        </div>
                    </div>
                    <form class="page__print">
                        <input value="Print Friendly" type="submit" />
                    </form>
                    <div class="footer aem-GridColumn aem-GridColumn--
default--12" data-sly-include="footer.html"></div>
                </div>
            </div>
        </div>
    </div>
</div>
```

3. Notice the HTL statement that includes the responsive grid:

   ```
   div data-sly-resource="${'responsivegrid' @
   resourceType='wcm/foundation/components/responsivegrid'}"></div>
   ```

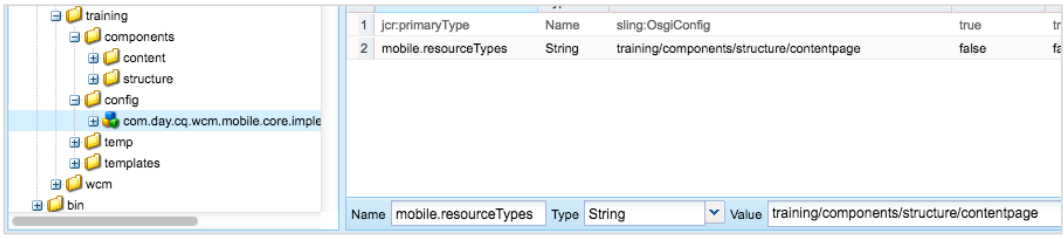
4. Using the Sites Console open **Sites** > **We.Train** > **English**.



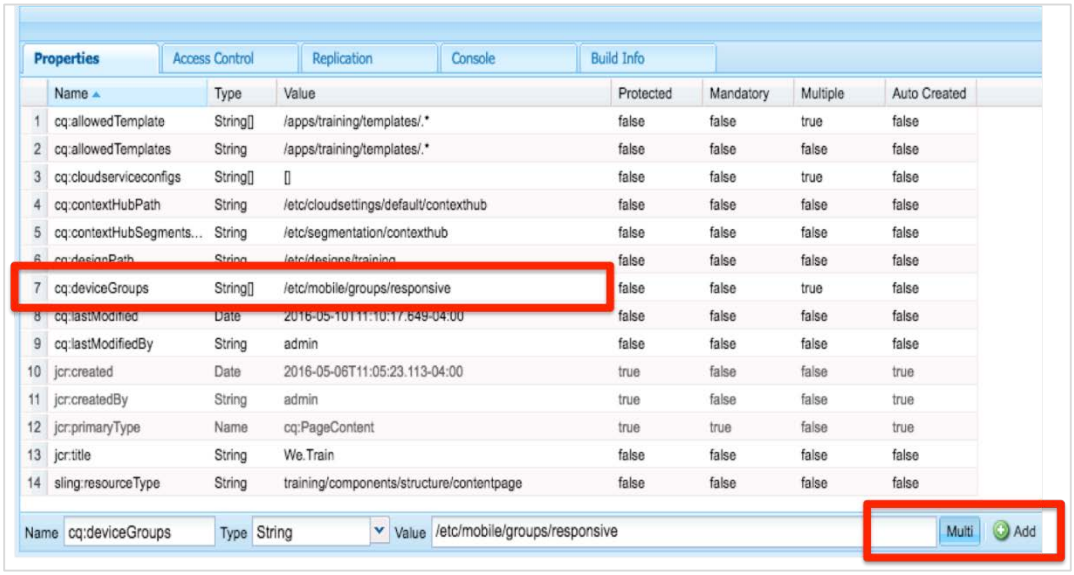
Task – Enable the Responsive Emulator

1. Using CRXDE Lite, navigate to **/app/training**.
2. Right-click on **training** and select **Create…** > **Create Folder**.
3. Enter "**config**"as the folder name.
4. Click **OK** and **Save**.
5. Create a child node of the **config** folder with the following values and save:
   - Name: com.day.cq.wcm.mobile.core.imple.MobileEmulatorProvider-training
   - Type: sling:OsgiConfig

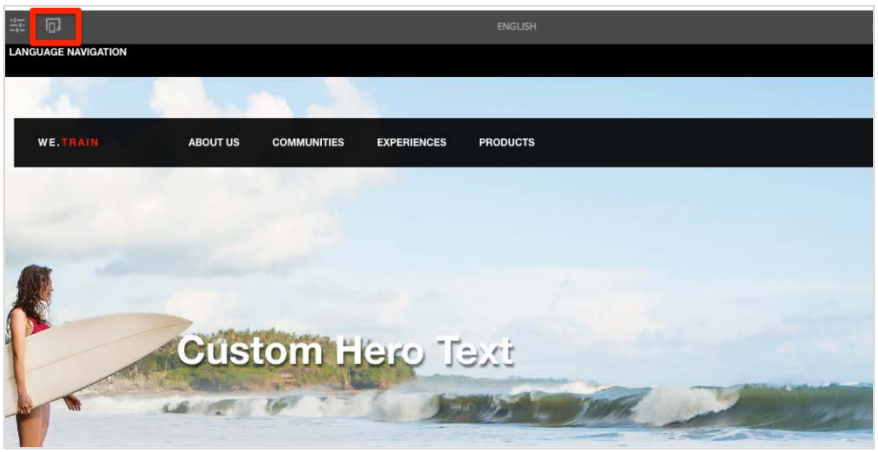
6. Add the following property to the node you just created.
   - Name: mobile.resourceTypes
   - Type: String
   - Value: training/components/structure/contentpage



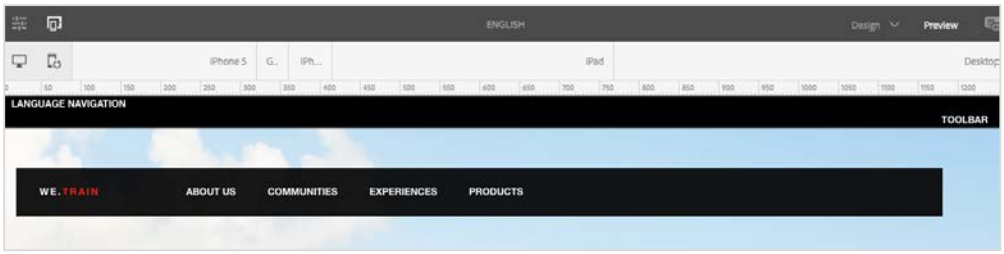
7. Navigate to /content/we-train/jcr:content.
8. Specify the device groups that will appear in the devices list by adding the following property to the **jcr:content** node.
   - Name: cq:deviceGroups
   - Type: String[]
   - Value: /etc/mobile/groups/responsive



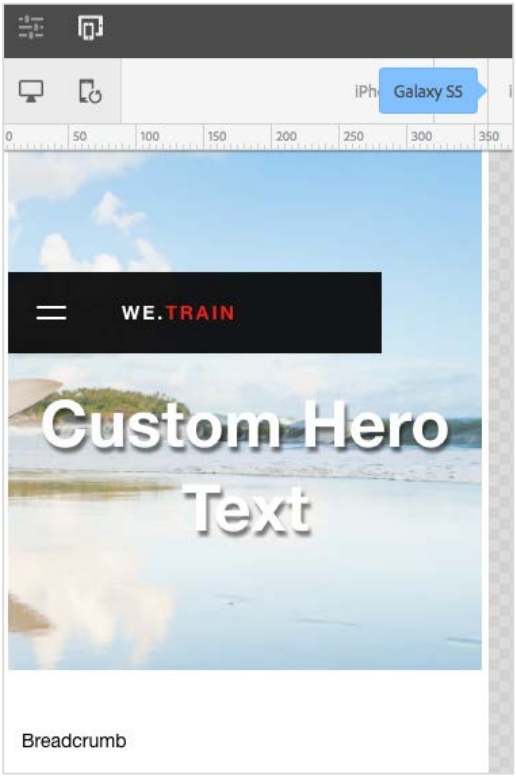
9. Using the Sites Console, navigate to **Sites** > **We.Train** > **English**.
10. Click on **Preview** mode.



11. Click on the **Emulator** icon.

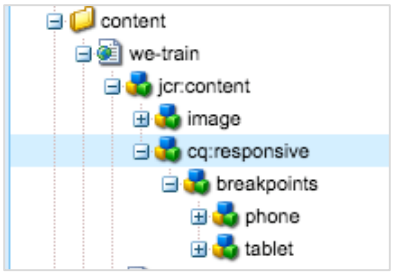


In the emulator, you can select the various devices for which you want to view the page. The following example shows the Products page as seen on an iPhone 6 Plus device. This multi-screen experience helps you to optimize your CSS further.



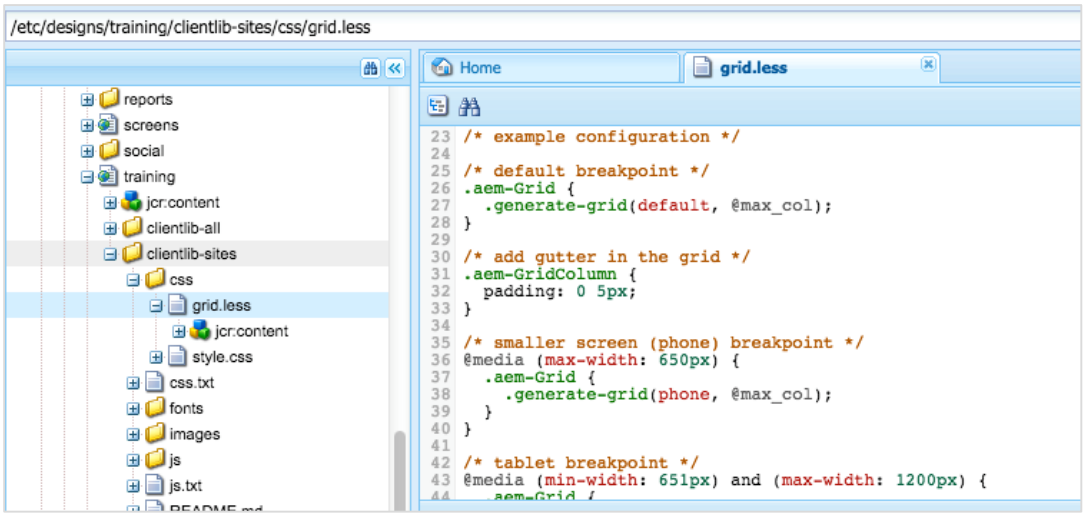
Task – Integrate Layouting Mode

1. Using CRXDE Lite, navigate to /content/geometrixx-media/en/jcr:content.
2. Copy the **cq:responsive** node.
3. Navigate to /content/we-train/jcr:content and paste the cq:responsive node.

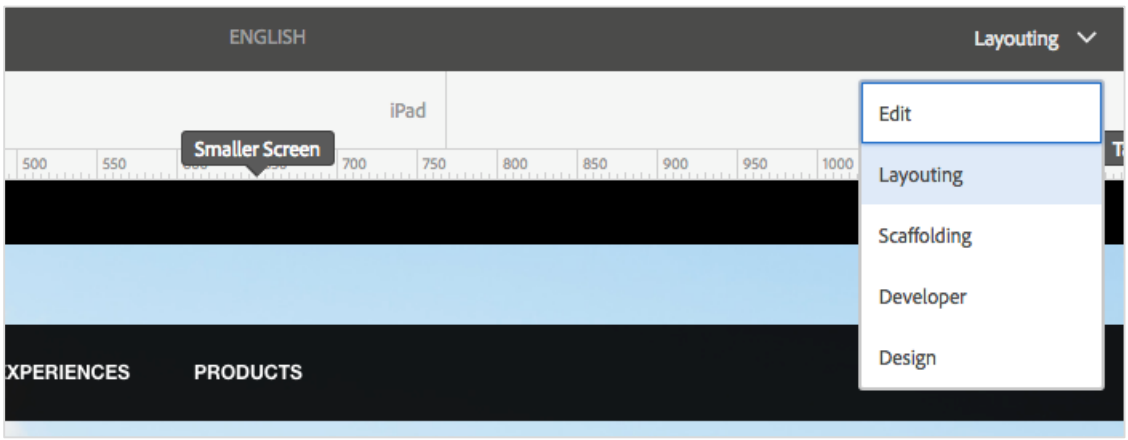


We have already added the responsive grid, but we need to styling to allow authoring of responsive content.

4. Navigate to /etc/designs/training/clientlibs-site/css.
5. Open **grid.less** and investigate the contents.



6. Using the Sites Console, navigate to **Sites** > **We.Train** > **English**.
7. Click on the **Modes** menu. Notice that **Layouting** mode is now available.



12 Advanced Sling Functionality

12.1 Lab Activity – Sling Selectors

Task – Investigate the Print Friendly button

1. Using CRXDE Lite, navigate to /apps/training/components/structure/contentpage.
2. Using the code from the USB Contents, replace the code in **body.html**.

body.html

```
<div class="container we-Container--main">
    <div class="root responsivegrid">
        <div class="aem-Grid aem-Grid--12 aem-Grid--default--12  ">
            <div class="header aem-GridColumn aem-GridColumn--default--12"
data-sly-include="header.html"></div>
            <div class="hero-image image parbase aem-GridColumn aem-
GridColumn--default--12">
                <div data-sly-resource="${'hero' @
resourceType='training/components/structure/hero'}"></div>
                <div class="responsivegrid aem-GridColumn aem-GridColumn--
default--12">
                    <div class="aem-GridColumn aem-GridColumn--default--12">
                        <div class="row">
                                                <div class="aem-breadcrumb"
data-sly-resource="${'breadcrumb' @
resourceType='foundation/components/breadcrumb'}"></div>
                            <div class="we-Header" data-sly-
resource="${'title' @
resourceType='training/components/structure/title'}"></div>
                            <div data-sly-resource="${'responsivegrid' @
resourceType='wcm/foundation/components/responsivegrid'}"></div>
                        </div>
                    </div>
                    <form class="page__print" action="${currentPage.Path @
selectors='print'}.html">
                        <input value="Print Friendly" type="submit" />
                    </form>
                    <div class="footer aem-GridColumn aem-GridColumn--
default--12" data-sly-include="footer.html"></div>
                </div>
            </div>
        </div>
    </div>
</div>
```

3. Look at the following statement:

```
<form class="page__print" action="${currentPage.Path @
selectors='print'}.html">
        <input value="Print Friendly" type="submit" />
</form>
```

When the user clicks the **Print Friendly** button, the following statement is sent as a request:

```
${currentPage.Path @ selectors='print'}.html"
```

If the currentPage is **/content/we-train/en**, then the request is: http://localhost:4502/content/we-train/en.print.html

Task – Create the print.html script

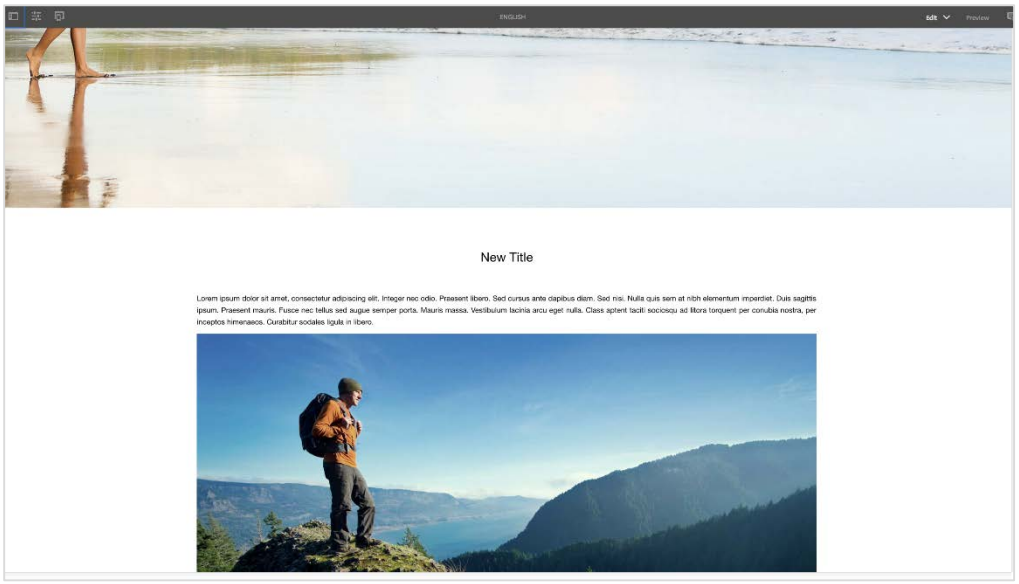
Think back to the previous discussion of Sling URL decomposition, resource resolution and how Sling finds the rendering script. When a URL contains a selector, then you know Sling will first attempt to find a script to match that selector.

1. Using CRXDE Lite, navigate to /apps/training/components/structure/contentpage.
2. Right-click on the **contentpage** node and select **Create…** > **Create File**.
3. Name the file **print.html**, click **OK** and **Save**.
4. Using the code from the USB Contents, paste the code into **print.html**.

print.html

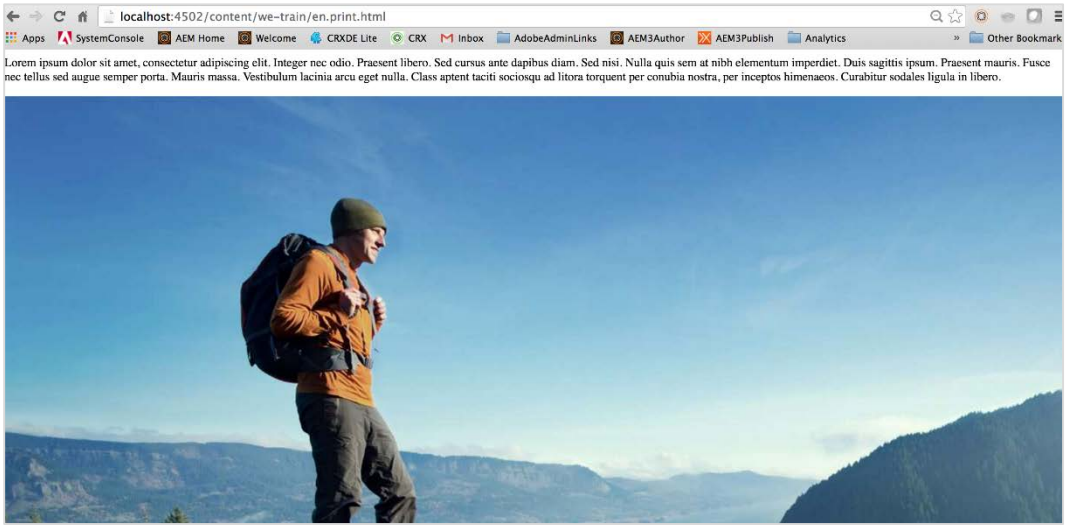
```
<div  data-sly-resource="${'title' @
resourceType='wcm/foundation/components/title'}"></div>

<div  data-sly-resource="${'responsivegrid' @
resourceType='wcm/foundation/components/responsivegrid'}"></div>
```

5. Using the Sites Console navigate to **Sites** > **We.Train** > **English** or any page that has content on it.



6. Remove **editor.html** from the URL. For example, http://localhost:4502/content/we-train/en.html

7. Click the **Print Friendly** button.



Notice that only the items in the paragraph system are displayed and there is no design. If you examine the code in print.html, you will see why the display looks the way it does.

12.2 Overlays and Sling Resource Merger

In Adobe Experience Manager, the UI is based on a set of nodes in the JCR repository, and its underlying structure. These nodes reside in the /libs folder. When you work on your project, any customization required is done by overlaying the corresponding content node in the repository, under the /apps folder. This is done by overlaying the content node in the repository.

There are two methods of performing these customizations.

- Overlays
- Sling Resource Merger

12.2.1 Overlays

Sling's Resource Resolver will search for resources in a list of specific paths, as defined by the Apache Sling JCR Resource Resolver. The default search path is first /apps and then /libs. As a result, you can change the out-of-the-box functionality, as provided in /libs by adding a resource or file at the same path in /apps. This ability to override default functionality is called an overlay.

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

Prior to Experience Manager 6.0, you would copy the whole content node that you want to change in /apps, and then work on this copy. You can change properties, remove or add child nodes, and so on. When a resource is requested, it gets resolved according to the search path logic. That is, it retrieves the resource from /apps, failing which, it looks at /libs.

Beginning with Experience Manager 6.0, resources are merged based on their search path. When requesting for a resource with a relative path prefixed by /mnt/overlay, the server will return the resource resulting from the merge of the resource from /apps with the resource from /libs.

With Experience Manager 6.1, resources are merged based on their resource type hierarchy. When requesting for a resource with an absolute path prefixed by /mnt/override, the server will return the resource resulting from the merge of the current resource with its super resource.

12.2.2 Sling Resource Merger

Sling Resource Merger is applicable only to the touch-optimized UI, and enables you to customize consoles and page authoring. With Sling Resource Merger, you can overlay nodes without replicating all of their ancestors. It uses diff mechanisms along with resource resolver search paths to merge overlays of resources.

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

Sling Resource Merger is used to ensure that all changes are made in /apps, thus avoiding the need to make any changes in /libs. Once 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.

12.2.3 Comparing Overlays with Sling Resource Merger

The following table describes the basic difference between using Overlays and Sling Resource Merger. However, it must be noted that both concepts are supported in AEM 6.2.

Overlays	Sling Resource Merger
Based on search paths (/libs + /apps)	Based on Resource Type Hierarchy
Need to copy the whole subtree	Extends within an almost empty subtree
All the properties are duplicated	Only required properties are overlaid

When upgrades are done to the /libs folder, these changes have to be manually recreated under /apps. As properties are not copied, only the structure, upgrades are automatically reflected in /apps.

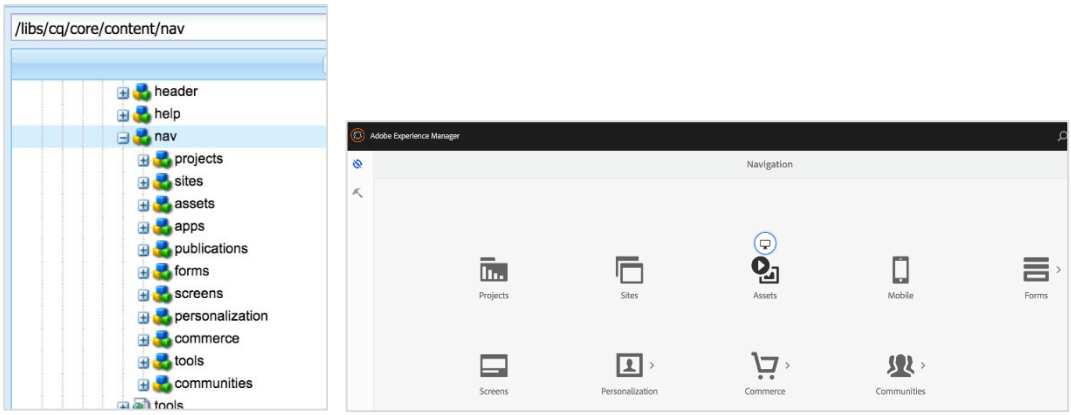
NOTE: After you copy a system file from /libs to /apps to overlay it with custom code, your custom code will not pick up any modifications to the system component/file/script/dialog box, which result from the application of a hotfix, feature pack, or upgrade. A careful examination of the release notes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

12.3 Lab Activity – Sling Resource Merger

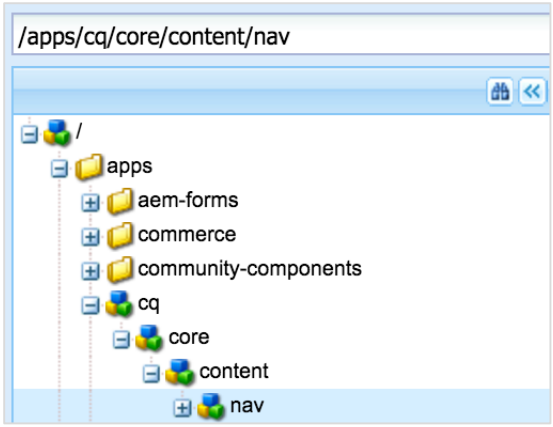
Task – Modify a Navigation Button

1. Using CRXDE Lite navigate to **/libs/cq/core/content/nav**.

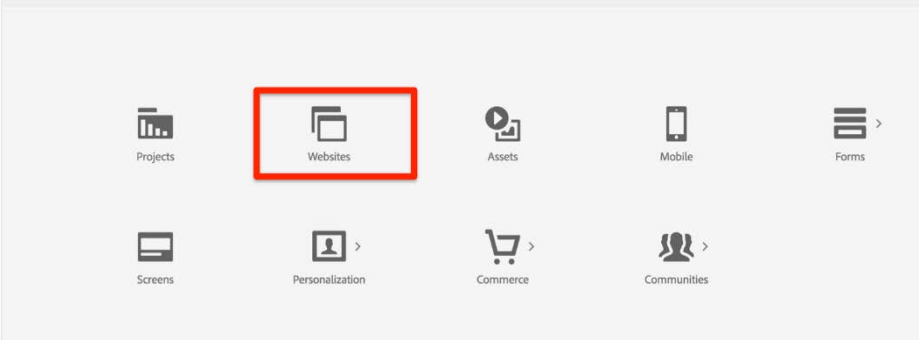
2. You will notice that the child nodes of nav are the definition of the global navigation buttons for AEM.



3. If you want to modify the buttons or add a new button, you need to change the list under nav. Since we know never to modify /libs we need a way to make the change in /apps. This is where the Sling Resource Merger comes in.
4. Navigate to **/apps**.
5. Right-click on **/apps**. Select **Create…** > **Create Node**. Recreate the same structure in **/apps** that exists in **/libs**. Each node is of type **nt:unstructured**.
6. cq > core > content > nav
7. **Tip:** Remember AEM is case sensitive.



8. Save after each node creation.
9. Under the **nav** node, create another node named **sites**, of type **nt:unstructured**.
10. Select the **sites** node.
11. Add the **jcr:title** property, with the value **Websites**.
12. Save your changes.
13. Using the AEM Home Screen, click on the **Global Nav Bar**.

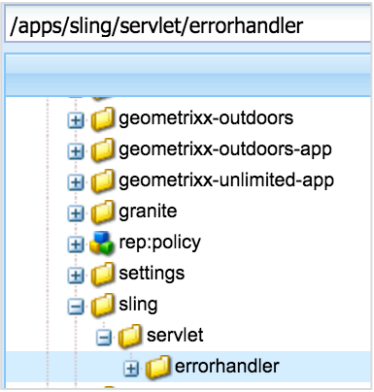


12.4 Lab Activity – Custom Error Handlers

Another common use case for the Sling Resource Merger is the definition of custom error handlers.

Task – Create a custom 404 Error Handler

1. Using CRXDE Lite navigate to **/apps**.
2. Right-click on **/apps** and select **Create…** > **Create Folder**. Create the following folder structure:
3. sling > servlet > errorhandler



4. **Tip:** Remember AEM is case sensitive.
5. Save after creating each folder.
6. In the errorhandler folder create two files: **404.html** and **ResponseStatus.java**.
7. Using the code from the USB Contents, paste the code into **404.html**.

404.html

```
<html data-sly-
use.responseStatus="apps.sling.servlet.errorhandler.ResponseStatus">
<head>
<title>File not found</title>
</head>
<body>
```

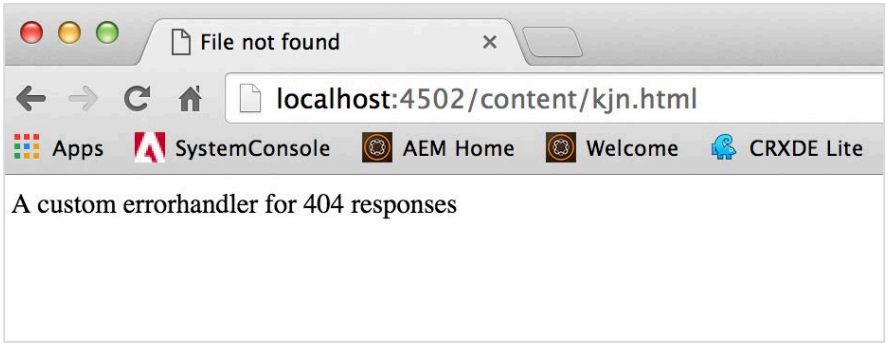
```
    <p>A custom errorhandler for 404 responses</p>
</body>
</html>
```

8. Using the code from the USB Contents, paste the code into **ResponseStatus.java**.

ResponseStatus.java

```
package apps.sling.servlet.errorhandler;

import com.adobe.cq.sightly.WCMUse;

public class ResponseStatus extends WCMUse {

    @Override
    public void activate() throws Exception {
        getResponse().setStatus(404);
    }
}
```

9. Save your changes.
10. Type a non-valid AEM URL into your browser. For example, http://localhost:4502/content/<yourinitials>.html



12.5 Lab Activity – Sling Redirect

Task – Make we-train.html redirect to another page

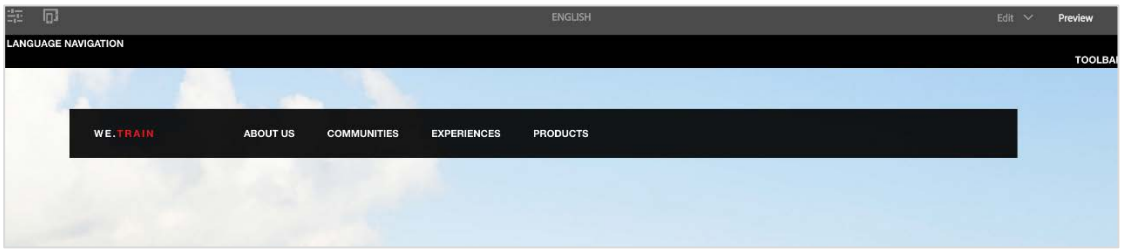
1. Using CRXDE Lite, navigate to **/content/we-train/jcr:content**.
2. Define 3 properties on the **jcr:content** node and save.

Name	Type	Value
redirectTarget	String	/content/we-train/en
sling:redirect	Boolean	true
sling:redirectStatus	Long	302

3. Modify the **sling;resourceType** property to point to the **Foundation** redirect component.

Name	Type	Value
sling:resourceType	String	foundation/components/redirect

4. Using the Sites Console, navigate to **Sites** > **We.Train**.
5. 

6. Switch to **Preview** mode and reload the page.
7. 

Notice that the /content/we-train.html URL redirected to /content/we-train/en.

13 Adobe Experience Manager Environment

13.1 Performance Consideration

A key issue is the time your website takes to respond to visitor requests. Although this value will vary for each request, an average target value can be defined. When this value is proven achievable and maintainable, it can be used by authors, who add and update the content, use this instance, monitor the performance of the website, and indicate the development of potential problems.

The response times you will be aiming for will be different on the author and publish instances, reflecting the different characteristics of the target audience.

Author Instance
Authors who add and update the content use this instance. It must cater for a small number of users— who generate a high number of performance-intensive requests—when updating content pages and individual elements on those pages.

Publish Instance
This instance contains content that you make available to your users, where the number of requests is even greater and the speed is just as vital. Because the nature of the requests is less dynamic, additional performance-enhancing mechanisms can be leveraged, such as the content is cached or load balancing is applied.

Performance Optimization Methodology
A performance optimization methodology for AEM projects can be summarized into five simple rules to avoid performance issues from the start. These rules, to a large degree, apply to web projects in general, and are relevant to project managers and system administrators to ensure that their projects will not face performance challenges when launch time comes.

Plan for Optimization
Around 10% of the project effort should be planned for the performance optimization phase. Of course, the actual performance optimization requirements will depend on a project's level of complexity and the experience of the development team. While your project may ultimately not require all the allocated time, it is a good practice to always plan for performance optimization in that suggested range.

Whenever possible, a project should first be soft-launched to a limited audience to gather real-life experience and perform further optimizations, without the additional pressure that follows a full announcement. When you are live, performance optimization is not over. This is the point in time when you experience the real load on your system. It is important to plan for additional adjustments after the launch.

Because your system load changes and the performance profiles of your system shifts over time, a performance tune-up or 'health-check' should be scheduled at 6–12 months intervals.

Simulate Reality
If you go live with a website and find out after the launch that you run into performance issues, there is only one reason for that—your load and performance tests did not simulate reality close enough. Simulating reality is difficult and how much effort you will reasonably want to invest into getting real depends on the nature of your project. Real means not just real code and real traffic, but also real content, especially regarding content size and structure. Keep in mind that your templates may behave completely different depending on the size and structure of the repository.

Establish Solid Goals
The importance of properly establishing performance goals is not to be underestimated. Often, when people are focused on specific performance goals, it is hard to change these goals later, even if they are based on wild assumptions.

Establishing good, solid performance goals is one of the trickiest areas. It is often best to collect real-life logs and benchmarks from a comparable website (for example, the new website's predecessor).

Stay Relevant
It is important to optimize one bottleneck at a time. If you do things in parallel without validating the impact of one optimization, you will lose track of which optimization measure actually helped.

Agile Iteration Cycles
Performance tuning is an iterative process that involves measuring, analysis, optimization, and validation until the goal is reached. To consider this aspect properly, implement an agile validation process in the optimization phase rather than a more heavyweight testing process after each iteration. This largely means that the developer implementing the optimization should have a quick way to tell if the optimization has already reached the goal, which is valuable information, because when the goal is reached, optimization is over.

Basic Performance Guidelines
Generally, keep your uncached html requests to less than 100ms. More specifically, the following may serve as a guideline:

- 70% of the requests for pages should be responded to in less than 100ms.
- 25% of the requests for pages should get a response within 100ms–300ms.
- 4% of the requests for pages should get a response within 300ms–500ms.
- 1% of the requests for pages should get a response within 500ms–1,000ms.
- No pages should respond slower than 1 second.

The above numbers assume the following conditions:

- Measured on publish (no authoring environment and/or CFC overhead)
- Measured on the server (no network overhead)
- Not cached (no AEM-output cache, no Dispatcher cache)
- Only for complex items with many dependencies (HTML, JS, PDF, and so on)
- No other load on the system

Certain issues frequently contribute to performance issues, which mainly revolve around (a) dispatcher caching inefficiency and (b) the use of queries in normal display templates. JVM- and OS-level tuning usually do not lead to big leaps in performance and should therefore be performed at the tail end of the optimization cycle. Your best friends during a usual performance optimization exercise are the request.log, component-based timing, and lastly, a Java profiler.

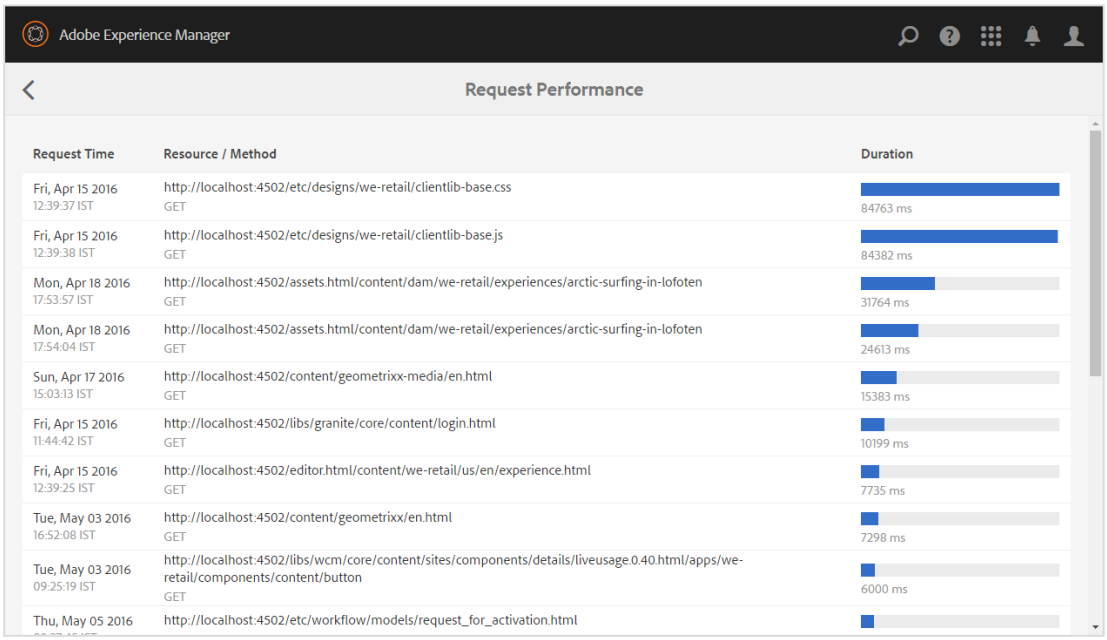
13.2 Lab Activity

Task – Monitor Page Response

1. Navigate to and open the file request.log located at <cq-install-dir>/crxquickstart/logs.
2. Request a page in author that utilizes your Training Template and components. For example, **/content/we-train/en**
3. Review the response times directly related to the previous step's request. (A page request of the page: /content/we-train/en)
4. You successfully reviewed the response time of a page using the request.log. Again, this will aid your development in being able to monitor response times of pages that implement custom templates and components, and comparing said time to your project goals.

Task – Find the Response Performance

1. Log in to AEM Sites. http://localhost:4502
2. Go to **Tools** > **Operations** > **Diagnosis** > **Request Performance**. The Request Performance page appears. The page displays requests in the order of duration taken.



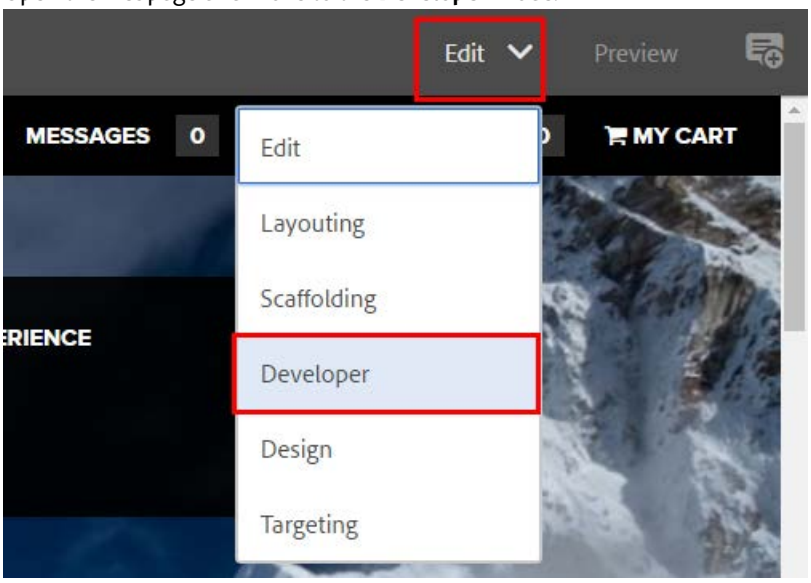
You can use this information to further investigate the cause of long responses.

You successfully found and displayed long-lasting requests/responses in AEM. Again, this is just one of many tools to help you meet your project's performance goals.

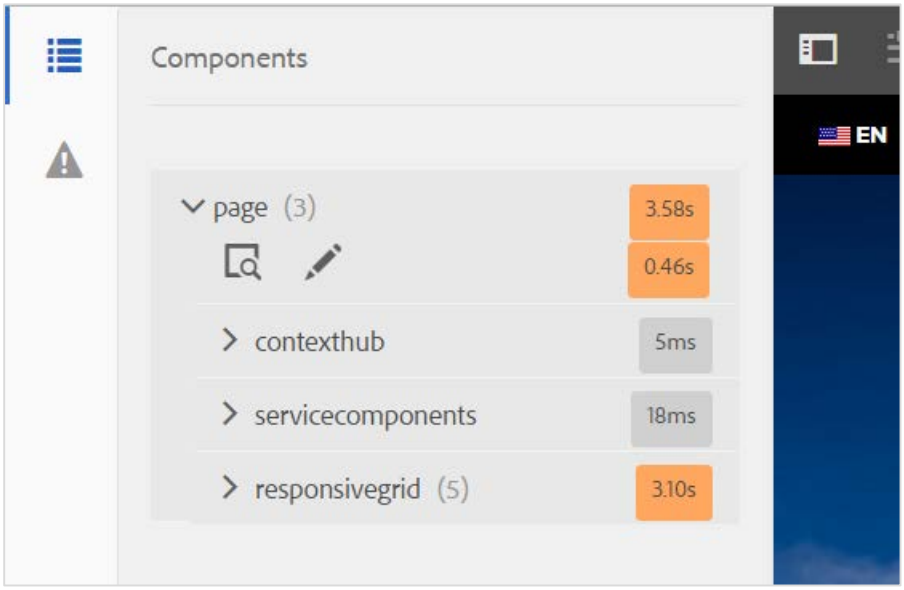
Task – Monitor component-based timing

In the Developer mode of the page, you can view the time taken by the components to load.

1. Open the webpage and move to the **Developer** mode.

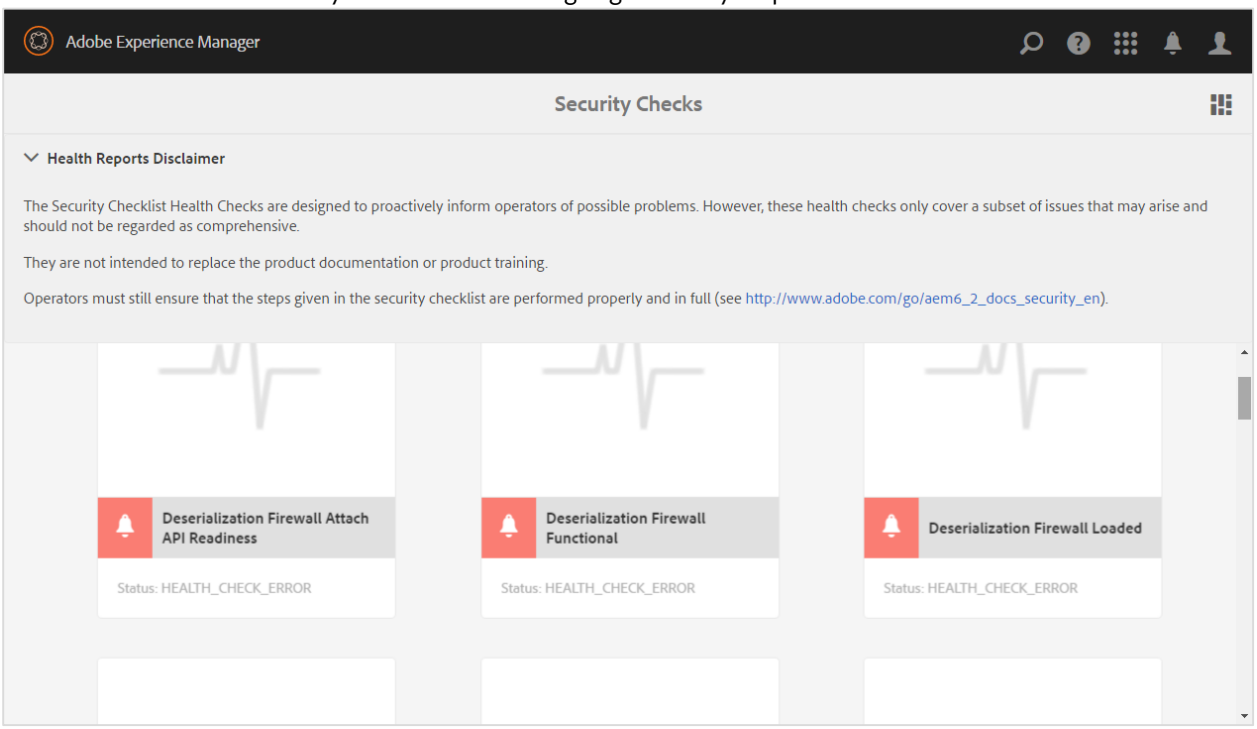


2. Go to the **Components** tab. This tab displays a component tree that provides you with the server-side computation time for each component.



13.3 Performing Security Checks

One of the most important health reports available in the dashboard is the Security Checks. It is recommended that you check the status of all the security health checks before going live with your production instance.



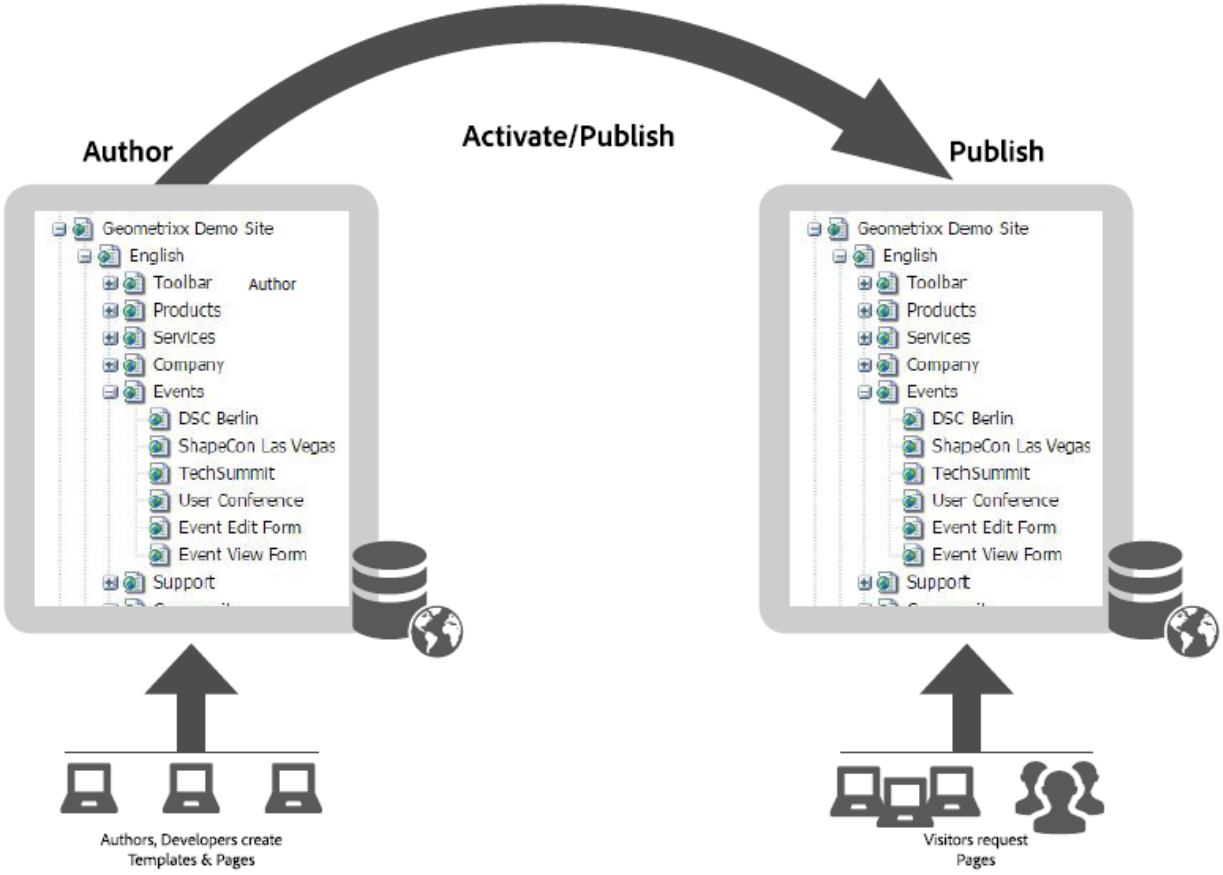
The following are a few of the checks that are essential for a secure instance:

- **Authorizable Node Name Generation** – Checks whether the AuthorizableNodeName implementation exposes the authorizable ID.

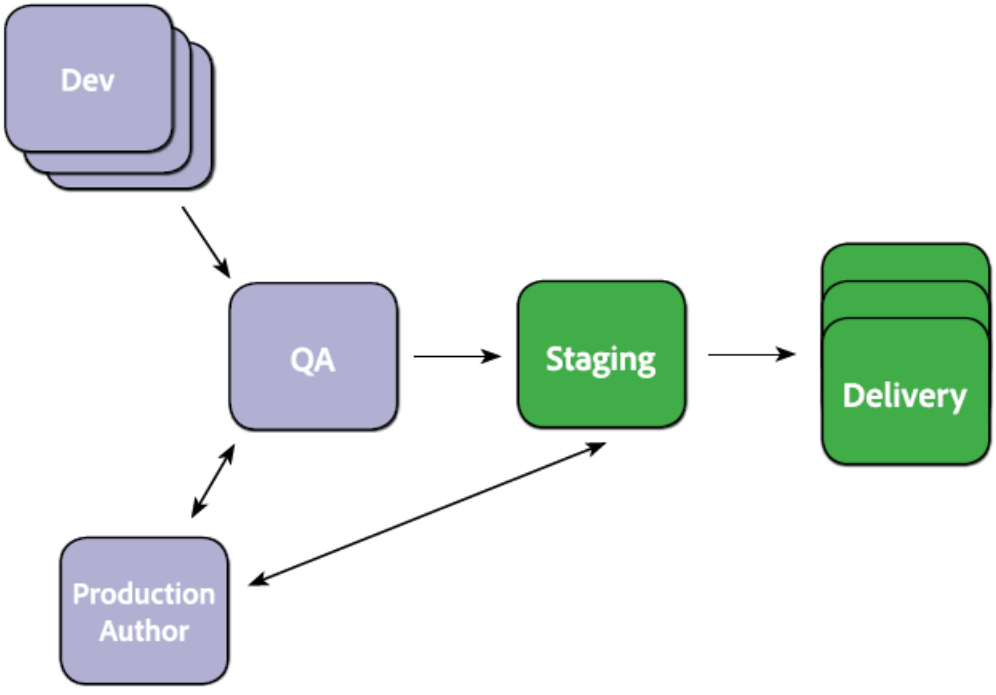
- **CRXDE Support** – Fails if the CRXDE Support bundle is active.
- **Default Login Accounts** - Fails if the default login accounts have not been disabled.
- **Sling Get Servlet** - Fails if the default Sling Get Servlet configuration is not following the security guidelines.
- **CQ Dispatcher Configuration** - Checks the basic configuration of the Dispatcher component.
- **Example Content Packages**—Fails if the sample content packages are found.
- **CQ HTML Library Manager Config**—Checks if the default CQ HTML Library Manager configuration follows the security guidelines.
- **Replication and Transport Users**—Checks the replication and transport users.
- **Sling Java Script Handler**—Checks if the Sling Java Script Handler configuration follows the security guidelines.
- **Sling JSP Script Handler**—Checks if the Sling JSP Script Handler configuration follows the security guidelines.
- **Sling Referrer Filter**—Checks if the Sling Referrer Filter is configured in order to prevent CSRF attacks.
- **User Profile Default Access**—Checks if everyone has read access to user profiles.
- **WCM Filter Config**—Checks if the default WCM Filter configuration follows the security guidelines.
- **WebDav Access**—Checks if the WebDav Access bundle is active.
- **Web Server Configuration**—Checks if the web server sends the X-FRAME-OPTIONS HTTP header set to SAMEORIGIN. In order for it to function, this Health Check needs to be configured with the public server address that the dispatcher is configured with.

13.4 AEM Deployment

An AEM deployment usually consists of multiple environments, used for different purposes on different levels. A production environment often consists of at least one author instance and one publish instance. Depending on the scale of a project, a production environment may consist of several author and/or publish instances, and at a lower level, the CRX repository may be clustered among several instances as well. Additionally, separate development and test environment levels may also consist of author and publish instances, mirroring the production environment to a varying extent.

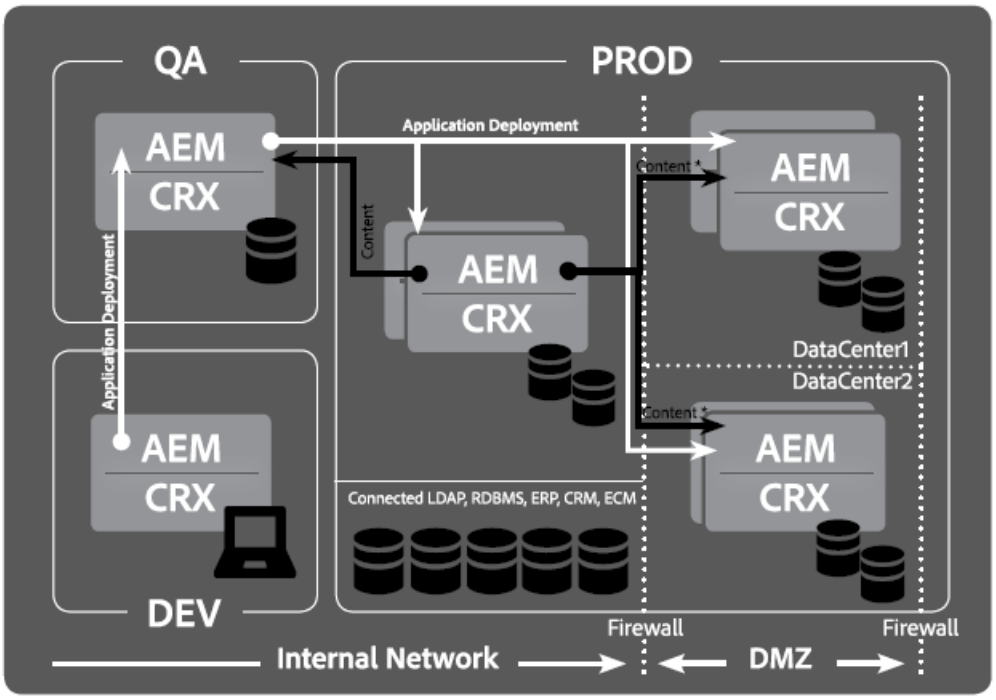


There are two types of installations—Author and Publish. Each Adobe Experience Manager instance will be one of these two. Author and publish instances share the same code base. However, they are started with a different run mode.

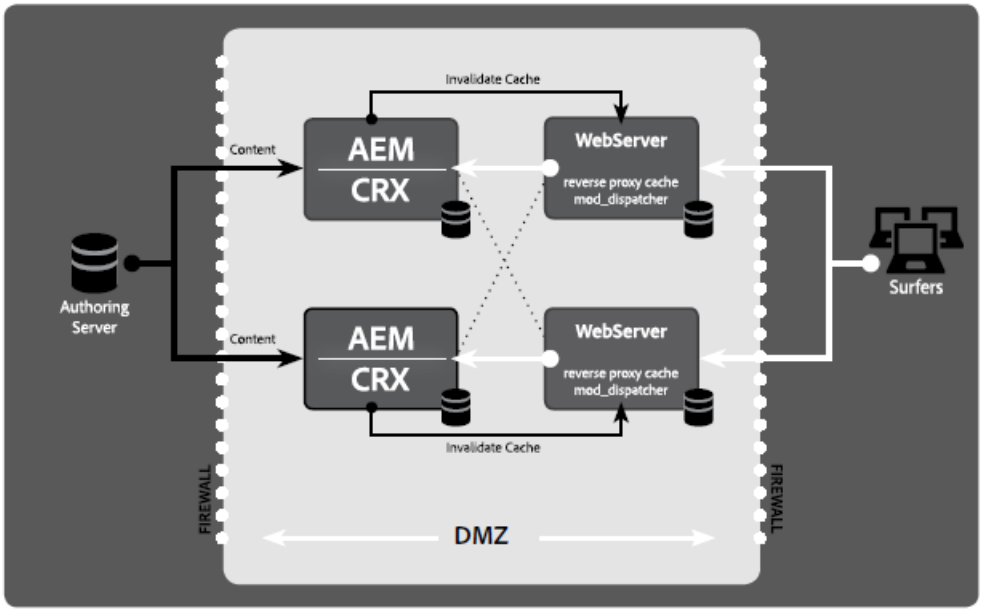


Content and code often move in different directions and get to their destination through different mechanisms. Content typically is moved by use of the Replication Agents. Content will move forward from Author to Publish, but also may move back to QA. This allows testing to be done on real data.

Deployment of code and configuration information is typically done through automated processes that use content packages, Replication Agents, and/or FileVault.

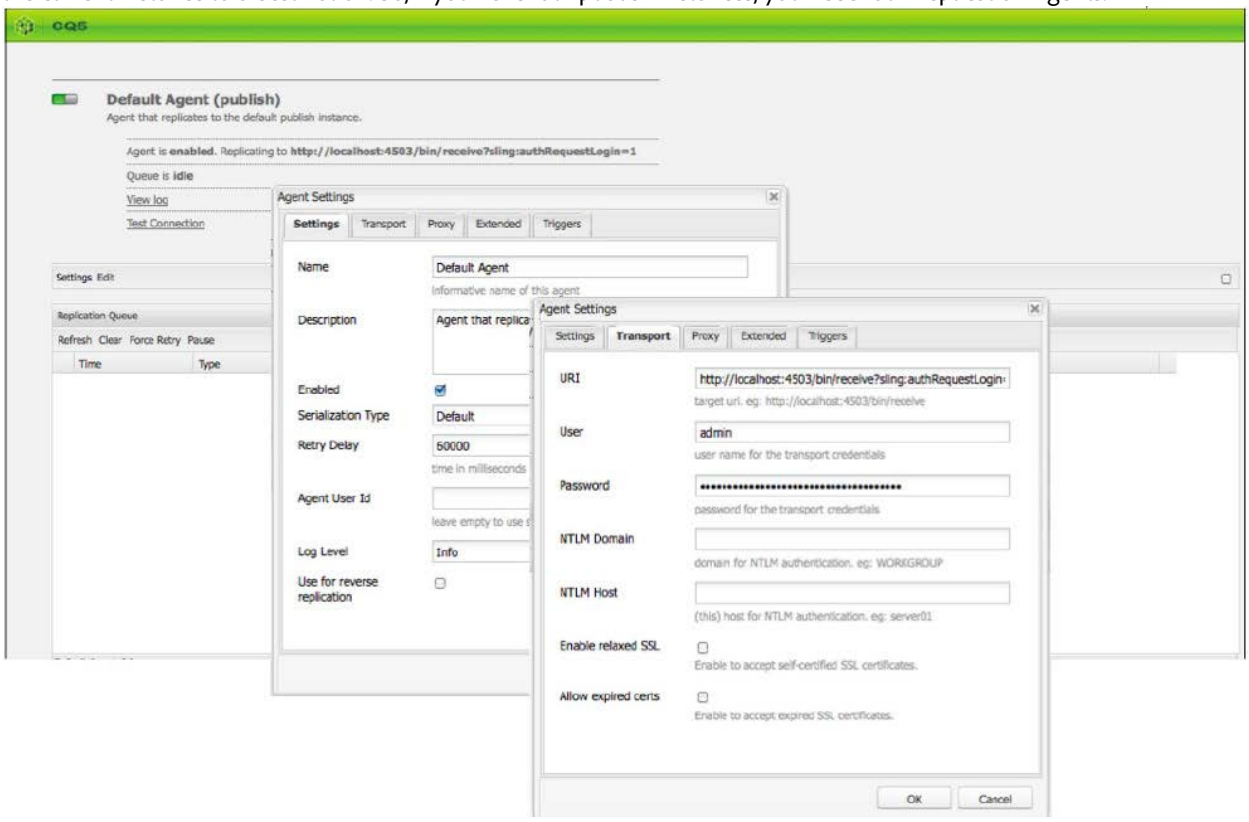


There are many designs for deployment of the AEM application itself. One of them is pictured below:



13.4.1 Replication

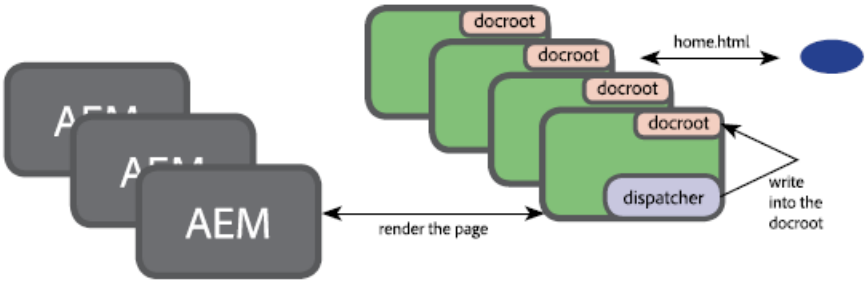
Activation or publication of content is handled by the Replication Agents. Each Replication Agent replicates content from the current instance to a destination. So, if you have four publish instances, you need four Replication Agents.



Details on the creation and management of Replication Agents can be found in the documentation at:
https://docs.adobe.com/docs/en/aem/6-2/deploy/configuring.html

13.4.2 Dispatcher

In the production delivery environment, the publish instance is joined by a web server module, called the Dispatcher. The Dispatcher is the AEM caching and/or load balancing tool.



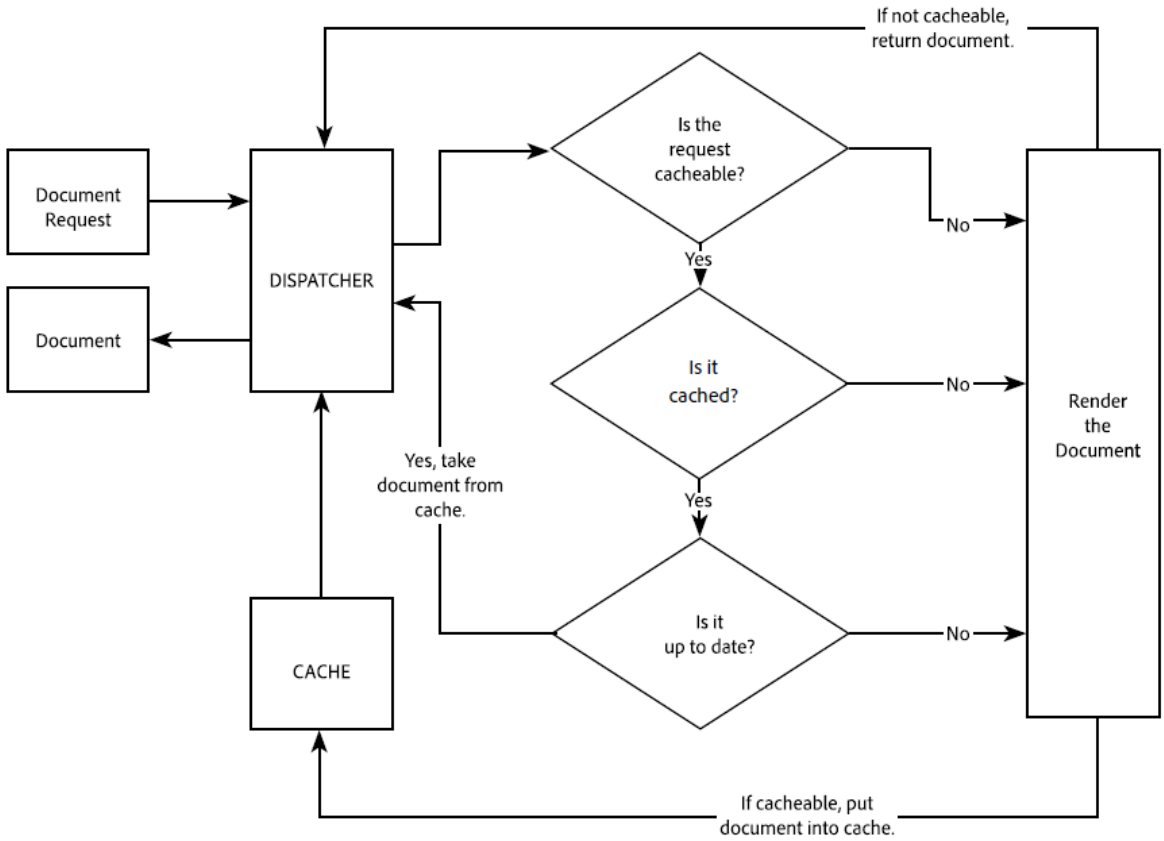
The Dispatcher helps realize an environment that is fast and dynamic. It works as part of a static HTML server, such as Apache, with the aim of:

- storing (or caching) as much of the site content as possible, in the form of a static website
- accessing the layout engine as little as possible

This means that:

- static content is handled with exactly the same speed and ease as on a static web server. Additionally, you can use the administration and security tools available for your static web server(s).
- dynamic content is generated as needed, without slowing the system any more than absolutely necessary

The Dispatcher contains mechanisms to generate and update static HTML, based on the content of the dynamic site. You can specify in detail which documents are stored as static files and which are always generated dynamically. The figure below demonstrates the algorithm that the Dispatcher uses to determine whether an object should be served from the web server cache or rendered dynamically.



Details regarding the management and configuration of the Dispatcher can be found at:
http://docs.adobe.com/docs/en/dispatcher.html

The Dispatcher algorithm for mobile content has an extra round-trip. Device evaluation is done with redirects, and device group-specific content is cacheable.