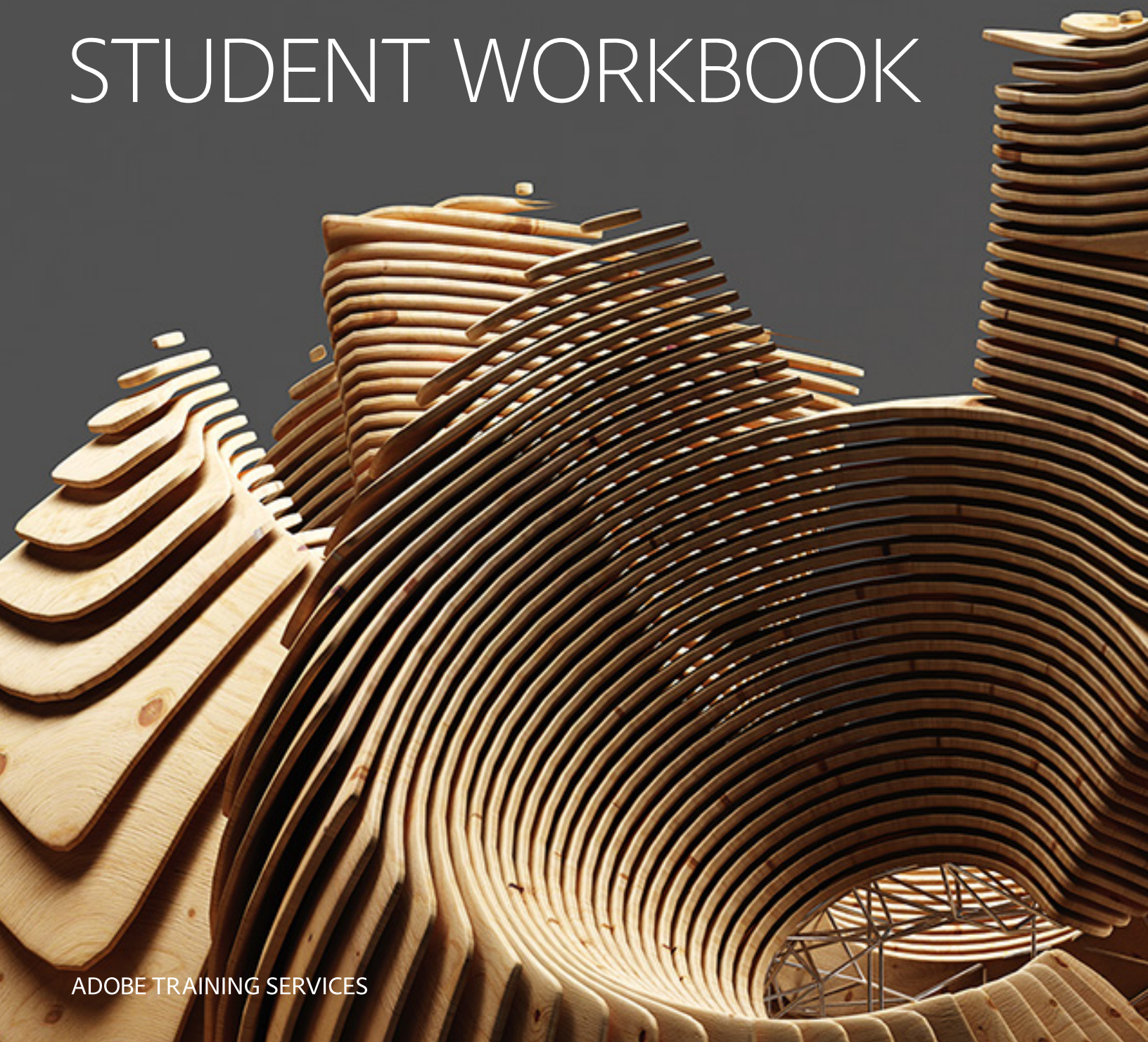
Adobe Experience Manager (6.x)
Sites: Developer

STUDENT WORKBOOK



ADOBE TRAINING SERVICES

Adobe Experience Manager 6.1 Developer

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

0515

Table of Contents

Getting Started 1-1

Introduction to AEM 1-2
AEM Platform 1-2
Installation and Deployment 1-3
What is an Author Instance? 1-3
What is a Publish Instance? 1-4
Installing AEM 1-4
Exercise 1.1: Install AEM 1-4
Exercise 1.2: Log in to AEM 1-7
AEM Consoles 1-8
Authoring in AEM 1-8
Exercise 1.3: Explore a Page and its Properties 1-8
AEM Web Consoles 1-15
Administration Interfaces 1-16
Developer Community 1-16

OSGi Framework 2-1

AEM Functional Building Blocks 2-2
Granite Platform 2-3
Architecture Stack 2-4
OSGi Framework 2-6
OSGi Bundles 2-7
Additional Information 2-7

Content Repository 3-1

JCR 3-2
Apache Jackrabbit 3-4
Jackrabbit Oak 3-5
Oak Architecture (Also Known as Hamburger Architecture) 3-5
MicroKernels 3-6
DocumentMK 3-6
SegmentMK 3-7
Adobe CRX 3-9
Repository Structure 3-9
Exercise 3.1: Familiarize Yourself With a Repository Structure 3-10
Exercise 3.2: Create a Node And Add Properties 3-12

Web Framework 4-1
REST 4-2
Apache Sling 4-3
Everything is a Resource 4-4
Exercise 4.1: Access Data in Different Formats 4-4
Sling Resolution 4-5
The Resolution Process 4-7

Developer Tools and Scripting Languages 5-1
Developer Tools 5-2
CRXDE Lite 5-2
Package Manager 5-3
Exercise 5.1: Create a content package of an existing project 5-3
Exercise 5.2: Upload and install a sample package 5-5
Brackets Sightly Extension 5-8
Exercise 5.3: Install the Brackets Plugin 5-9
AEM Developer Tools for Eclipse 5-10
Scripting Languages 5-11
JSP 5-11
Sightly 5-11
Exercise 5.4: Work on the Geometrixx project using Brackets plugin 5-16

AEM Authoring Framework — Templates 6-1
Creating Your Website 6-2
Structure Your Application 6-4
Exercise 6.1: Create the Structure of Your Website 6-4
Create Templates 6-5
Exercise 6.2: Create a Template for Your Website 6-6
Create a Page-Rendering Component 6-8
Exercise 6.3: Create a Page-Rendering Component 6-8
Create Pages 6-9
Exercise 6.4: Create a Website Structure 6-10
Modify Page-Rendering Scripts 6-11
Exercise 6.5: Modify the Page-Rendering Script to Use a Sightly Script 6-11
Use APIs to Display Basic Page Content 6-12
Exercise 6.6: Display Basic Page Content Using Available APIs (in Sightly) 6-13
Displaying Basic Page Content Using JSP 6-14
Recap of Sling framework 6-17
Exercise 6.7: Create Multiple Scripts for the Page Component 6-18

AEM Authoring Framework — Components and Design 7-1
Modularize the Page Component 7-2
Exercise 7.1: Modularize the Page Component 7-2
Inheriting Foundation Components 7-4
Types of Hierarchies 7-4
Overlays 7-5
Sling Resource Merger 7-5
Overlays vs. Sling Resource Merger 7-6
Exercise 7.2: Inherit the Sightly Foundation Component Page 7-6
Extra Credit Exercise: Add a New Navigation Item to The Admin UI 7-8

Add the Design 7-10
Exercise 7.3: Add a Design to the Page 7-10
Create Components and Include Them in a Script 7-14
Create a Top Navigation Component 7-14
Exercise 7.4: Create a Top Navigation Component and Include it in a Script 7-15
Exercise 7.5: Create a Top Navigation Component by Accessing the Root Node 7-17
Exercise 7.6: Create a Top Navigation Component Using Java 7-18
Add a Log Message From the Script 7-19
Exercise 7.7: Add a Log Message Using the JavaScript File of a Script 7-20

AEM Authoring Framework — Dialog Boxes 8-1
Create Dialog Boxes for Components 8-2
Touch-Optimized UI 8-2
Classic UI 8-4
Exercise 8.1: Create a Training Title Component 8-5
Exercise 8.2: Create a Dialog Box for a Classic UI 8-8
The Dialog Conversion Tool 8-10
Exercise 8.3: Convert a Classic UI Dialog Box to a Touch-Optimized UI Dialog Box 8-10
Exercise 8.4: Create a Dialog Box for Touch-Optimized UI 8-12
Use Design Dialog Boxes for Global Content 8-15
Exercise 8.5: Create a Logo Component 8-16
Use cq:EditConfig to Enhance the Component 8-21
Exercise 8.6: Enable In-place Editing in the Title Component 8-21

AEM Authoring Framework — Foundation Components, Internationalization, and Client Libraries 9-1
Work With the Foundation Components 9-2
Exercise 9.1: Include a Breadcrumb Foundation Component 9-2
Include the Paragraph System 9-4
Exercise 9.2: Include the Paragraph System Component 9-4
Exercise 9.3: Use the Toolbar Component 9-8
Exercise 9.4: Include the iParsys Component 9-12
Internationalize the Authoring Interface 9-14
Exercise 9.5: Internationalize the Title Component's GUI 9-14
Add Client Libraries 9-18
Client or HTML Libraries 9-18
Client Library Conventions 9-18
Examples of Client Libraries 9-19
Include Client Libraries 9-19
Exercise 9.6: Include a JavaScript Function From Client Libraries 9-20

Mobile Websites 10-1
Responsive Design 10-2
Pros and Cons of Responsive Design 10-3
Exercise 10.1: Preview the Site With Various Devices 10-4
Mobile Components 10-7
Exercise 10.2: Create a Mobile Time Component 10-8
Creation of a Mobile Website Using MSM 10-11
Exercise 10.3: Create a Mobile Website 10-11
Mobile Emulators 10-15
WURFL 10-15

Emulator Groups 10-16
Emulator Framework 10-17

Complex Components Using JSP 11-1
Working With Complex Components 11-2
Exercise 11.1: Create a Complex Component 11-4
End-User Search 11-10
Exercise 11.2: Create a Search Component 11-11
Using jQuery With Ajax and Apache Sling 11-14
Exercise 11.3: Create a Component to Display Dynamic Grid of User Accounts 11-14

OSGi Bundles and Workflow 12-1
Creating OSGi Bundles 12-2
What Exactly Is an OSGi Bundle? 12-2
Exercise 12.1: Consume an OSGi Bundle 12-3
Basics of the Workflow Console 12-5
Overview of the Main Workflow Objects 12-5
Starting a Workflow 12-7
Exercise 12.2: Explore Basic Workflow 12-7
Create a Workflow Implementation Step 12-10
Exercise 12.3: Define a Process Step Using Java 12-10
Exercise 12.4: Implement a Process Step 12-12

AEM Environment 13-1
Useful Tools 13-2
Developer Mode in Touch-Optimized UI 13-2
?debug=layout 13-3
?debugConsole=true 13-4
?debugClientLibs=true 13-4
Performance Consideration 13-6
Exercise 13.1: Monitor Page Response 13-9
Exercise 13.2: Find the Response Performance 13-10
Exercise 13.3: Monitor Component-based Timing 13-11
AEM Deployment 13-12
Replication 13-14
Reverse Replication 13-14

Extract for Brackets (Extra Credit) 14-1
Overview 14-1
Setting up the Development Environment 14-2
Exercise 14.1: Configure the Extracts Extension 14-2
Extracting from a PSD 14-4
Exercise 14.2: Modify Geometrixx Outdoors Page by Extracting Content From PSD 14-4

Getting Started

Overview

This chapter introduces you to Adobe Experience Manager (AEM).

Objective

In this chapter, you will learn how to do the following:

- Install and deploy AEM
- Work with User Interfaces (UIs)
- Work with various web consoles

Introduction to AEM

AEM helps you create, organize, and manage the delivery of creative assets and other content across your digital marketing channels, including web, mobile, email, communities, and video.

AEM provides digital marketers with easy-to-use, web-based applications for creating, managing, and delivering personalized online experiences. AEM provides out-of-the-box integration with other Adobe Marketing Cloud solutions.

Following are the major capabilities of AEM:

- Social communities: Include user-generated content and social media into your brand's message to create loyal communities on your own sites.
- Mobile content management: Integrate mobile into all your marketing efforts, maintaining consistency and confidence that the content functions on any device.
- Commerce: Quickly deliver branded, personalized shopping experiences to make the most of every customer interaction.
- Cloud management: Publish content to digital channels at a lower cost and with increased efficiency and security in the cloud.
- Marketing campaign management: Plan, design, launch, and optimize marketing campaigns across channels. Deliver profile-specific content and reuse effective collateral.
- Media publisher: Accelerate content marketing through simple workflows and by publishing to tablets with Adobe Digital Publishing Suite.
- Multisite management: Manage and publish templates and assets across multiple web and mobile sites for diverse regions and languages.
- Document services and security: Create and manage customer forms, share and track personalized client documents more securely, and publish documents on websites.

AEM Platform

AEM is implemented as a Java web application. It runs in any server that supports the Java servlet Application Program Interface (API) 2.5 (or higher). AEM comes preconfigured with its own built-in Servlet engine, but can also be installed in any compatible third-party application server. Because the system can be accessed from any computer with a modern web browser, no installation of client software is required. Therefore, large installations with thousands of users can be rolled out and upgraded easily.

Installation and Deployment

A Java web application (or web app) is a program that runs on a remote server and is accessed by users through a web browser. It is usually contrasted with a static website, which is a collection of documents that reside on the server and that can be viewed through a browser over the web. Alternately, a web application typically assembles the data to be sent back to the browser, dynamically with each request (or even using Ajax between requests). Most modern websites are essentially web applications. AEM is also a web application, but one that serves as a platform for building other web applications and managing the content they deliver.

As a Java web application, AEM runs on any server that supports the Java servlet API. For ease of installation, AEM comes bundled as a self-extracting Quickstart file that needs only to be double-clicked to install and start the server. For more details, see Supported Platforms. (http://docs.adobe.com/docs/en/aem/6-0/deploy/technical-requirements.html)

As AEM is Java based, it can run on any system for which a Java Runtime Environment (JRE) is available. This means, for example, that AEM can run on any mainstream operating system, including Windows, Macintosh, Linux, or other flavors of UNIX.

While different instances in different environments are all installations of the same AEM software—installed in different places in the overall system infrastructure—they differ mainly in the way they are configured. For example, it is that configuration, or run mode, that determines whether an AEM instance behaves as an author instance or a publish instance.

What is an Author Instance?

Author instances are usually located behind the internal firewall. This is the environment where you and your colleagues will perform authoring tasks, such as:

- administering the web properties
- inputting your content
- configuring the layout and designing your content
- activating your content to the publish environment

Content that has been activated is packaged and placed in the author environment's replication queue. The replication process then transports that content to the publish environment.

An author instance is the AEM installation that content authors will log in to and manage pages from. This includes creating, editing, deleting, moving, and so on. In addition, it is the instance run mode that you will use for development because you can easily observe Author and Publish views from this mode.

What is a Publish Instance?

A publish environment is usually located in the Demilitarized Zone (DMZ). This is the environment where visitors will access your website and interact with it, be it public or within your intranet.

- Holds content replicated from the author environment
- Makes that content available to visitors of your website
- Stores user data generated by your visitors, such as comments or other form submissions
- Can be configured to add such user data to the outbox for reverse replication back to the author environment

The publish environment generates your website's content pages dynamically in real time and the content can be personalized for each individual user.

Installing AEM

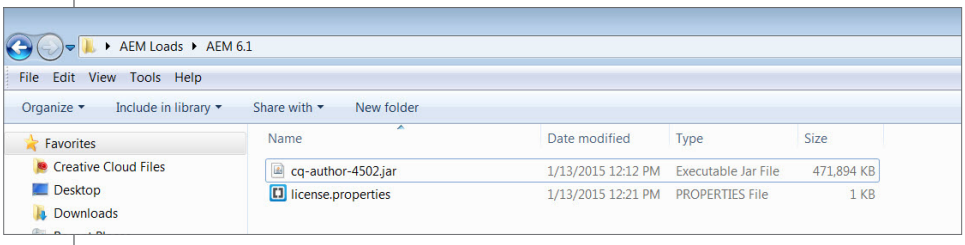
Unlike many other applications, you install AEM by using a quickstart, self-extracting JAR file. When you double-click the JAR file for the first time, everything you need is automatically extracted and installed. The AEM quickstart JAR file includes all files and repository structures required for the Content Repository eXtreme (CRX) (a fully JCR 2.0/JSR-283 compliant repository and Apache Sling), virtual repository services, index and search services, workflow services, security, and a web server.

You can run AEM without an application server, but you need a Servlet Engine. CRX and AEM Web Content Management (WCM) ship with a built-in Servlet engine (Jetty 8.1), which is fully supported and can be used for free. The first time you start the JAR file, it creates an entire JCR-compliant repository in the background, which may take several minutes. After this, startup is much quicker because the applications have been installed and the repository is already created.

NOTE: In production, you can install AEM in any directory structure. However, for this class, we recommend the structures described to the right, so that you can easily find the directory structures mentioned in the course materials.

Exercise 1.1
Install AEM

1. Create a folder structure on your file system where you will store, install, and start the AEM. For example:
   a. **Windows**: C:/adobe/AEM/author
   b. **MacOS X:** /Applications/adobe/AEM or ***x***: /opt/adobe/AEM/author
2. Copy the AEM quickstart `JAR` and `license.properties` files from <USB>/distribution/AEM into the newly created folder structure.

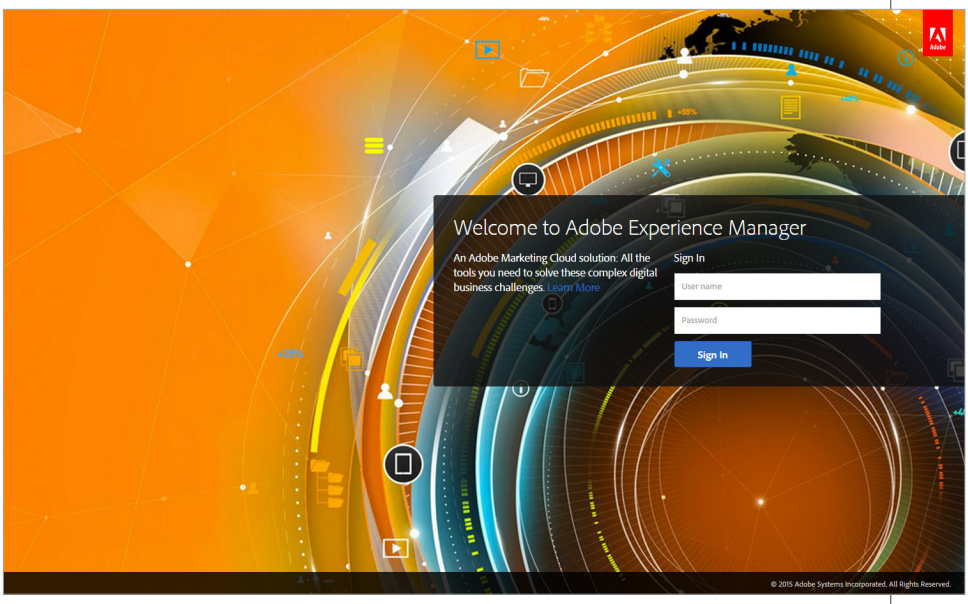


3. Rename the AEM quickstart JAR file to `cq-author-4502.jar`.
   a. cq = Application
   b. author = WCM mode it will run in; for example, author or publish
   c. 4502 = Port it will run in (any available port is acceptable)
4. In a Windows or MacOS X environment, double-click the `cq-author-4502.jar` file. Installation will take approximately 5–7 minutes depending on your system's capabilities.

NOTE: If no port number is provided in the file name, AEM will select the first available port from the following list:
1) 4502, 2) 8080, 3) 8081, 4) 8082, 5) 8083, 6) 8084, 7) 8085, 8) 8888, 9) 9362, 10) Random.

After AEM has started successfully, the startup screen will change to something similar to the following:

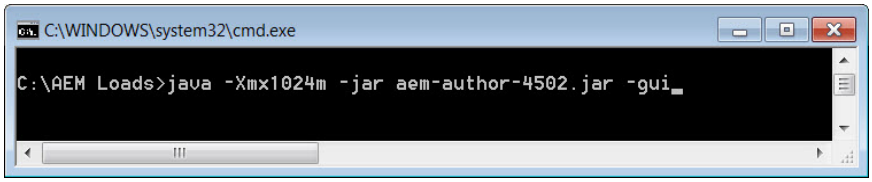
In addition, after AEM starts, your default browser will automatically open to AEM's start URL (where the port number is the one you defined on installation); for example, http://localhost:4502.



You have now successfully installed and started AEM. To start AEM in the future, double-click the renamed AEM quickstart JAR file; for example, `cq-author-4502.jar`.

Install or Start AEM Using a Command Line

There are two ways to install AEM: graphical and by command line. The latter is more powerful because the user has the possibility to provide additional performance-tuning parameters to the Java Virtual Machine (JVM). On Windows, MacOS X, or *x, you can also install or start AEM from the command line while increasing the Java heap size, which will improve performance. See the following image:



A typical command line start:

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

Tuning the JVM is a very 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 sufficient 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 the physical memory available and expected traffic
Syntax	-Xms512m (sets the initial heap size to 512 MB)

-Xmx --> assigns the maximum heap size

Default value	64 MB for a JVM running on 32-bit machines, or 83 MB for 64-bit machines
Recommended	Specific to the physical memory available and expected traffic, but should be equal to or greater than the initial size. To run AEM, it is recommended to allocate at least 1,024 MB of heap size.
Syntax	-Xmx1024m (sets the maximum heap size to 1,024 MB)

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

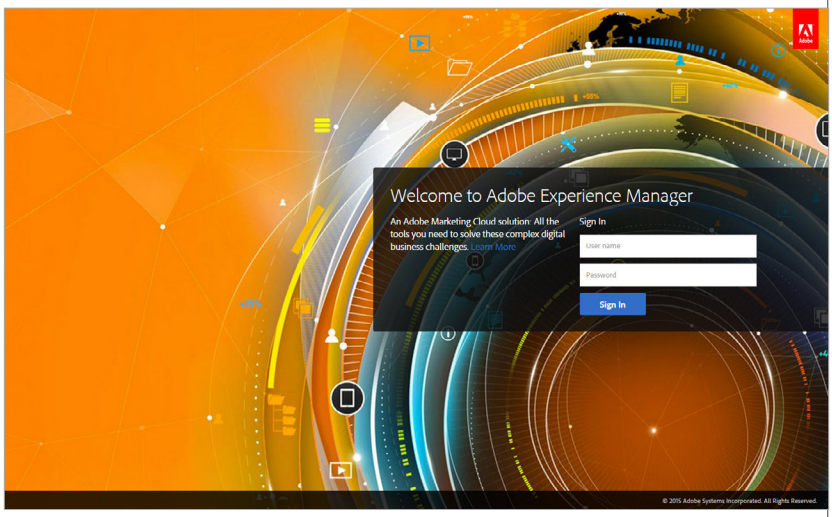
Default value	32 MB for a JVM running as a client, or 64 MB when running as a server
Recommended	Should be set to at least 128 MB for 'normal-sized' web apps or 256 MB for larger web apps with significant Java activity
Syntax	-XX:MaxPermSize=128m (sets the initial perm gen size to 128 MB)

Use the following command to install AEM, without installing the Geometrixx sites:

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

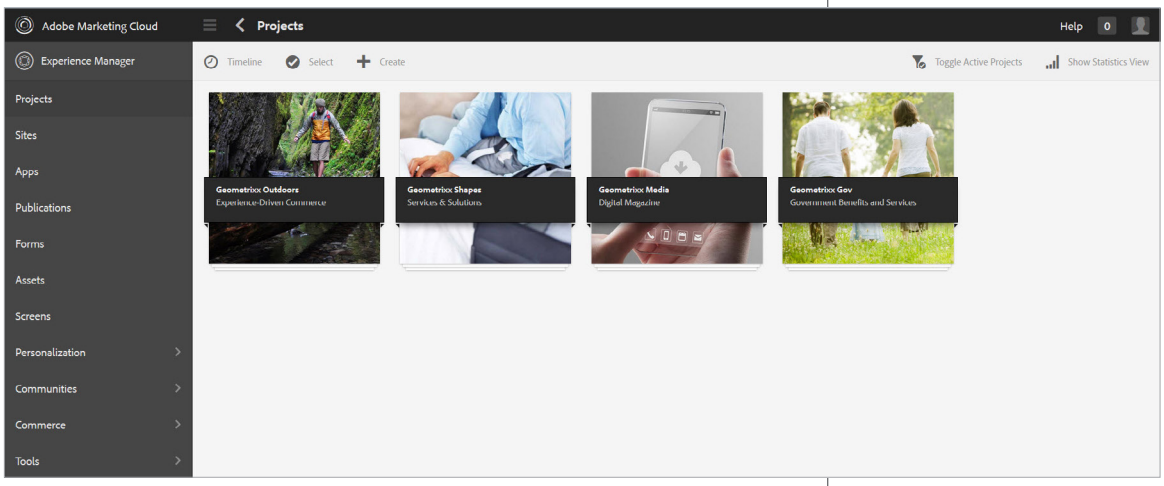
Exercise 1.2
Log in to AEM

1. Start AEM on a browser. The following screen appears:



2. Log in using the following credentials:
   user: **admin**
   password: **admin**

3. After a successful login, the following screen appears. You can see the Projects console by default. The Sites console displays various sites that you created.



NOTE: When you first login to AEM, you are prompted with a dialog box to integrate with Analytics and Targeting data. Click **Cancel**.

AEM Consoles

Following are the important consoles that you need to be familiar with:

- **Projects**: Manages the various aspects of the project
- **Sites**: Displays all existing sites. These sites are the sample implementation that you can refer to when you work with your actual sites.
- **Publications**: Manages publications of your organization
- **Forms**: Allows you to create forms using Adobe LiveCycle
- **Assets**: Provides you with your digital assets. It was formerly known as Digital Asset Management (DAM).
- **Communities**: Provides you with support through various communities.
- **Tools**: Provides you with various tools to manage operations and assets. It also provides you with a code inspection tool named CRXDE Lite and the Web Console.

Authoring in AEM

AEM provides you with the following distinct user experiences while editing your web content:

- **Classic UI:** Provides you with a desktop-like UI
- **Touch-optimized UI:** Provides you with UI that is optimized to use in various handheld devices, such as tablets and phones

You can use the UI based on your choice. Note that the touch-optimized UI works on a desktop too. At any point, you can switch from Classic UI to touch-optimized UI.

The Classic UI and administrative interfaces use Ajax to enable a desktop-like user experience. For example, when editing content on a website, authors can drag and drop elements like text paragraph and images right onto the page, and immediately see how their changes affect the appearance of the page.

Touch-Optimized UI

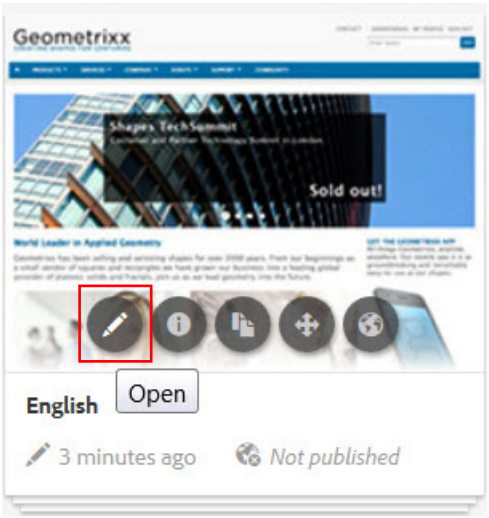
By default, when you open a page, it appears in the touch-optimized UI.

Exercise 1.3
Explore a Page and its Properties

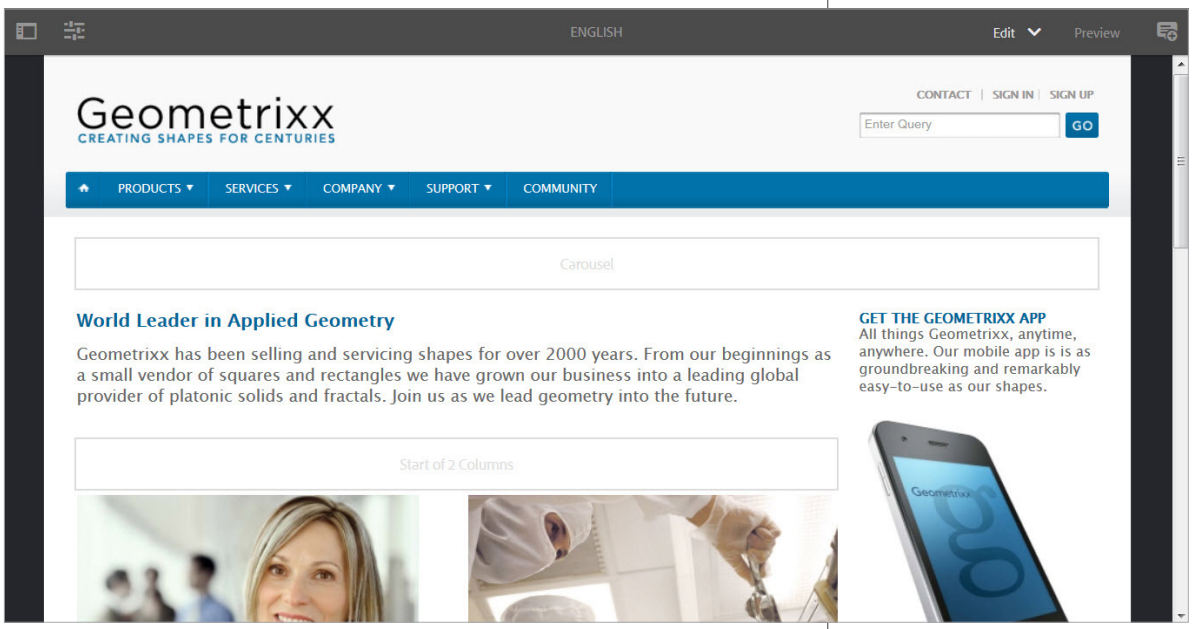
1. Log in to AEM.
   http://localhost:4502
2. From the left pane, select **Sites**. Click **Geometrixx Demo Site** to navigate into it.

NOTE: AEM 6.1 comes with a new Column view where navigation to child pages is much easier. To access this view, select the Column view in the upper right corner of the screen.

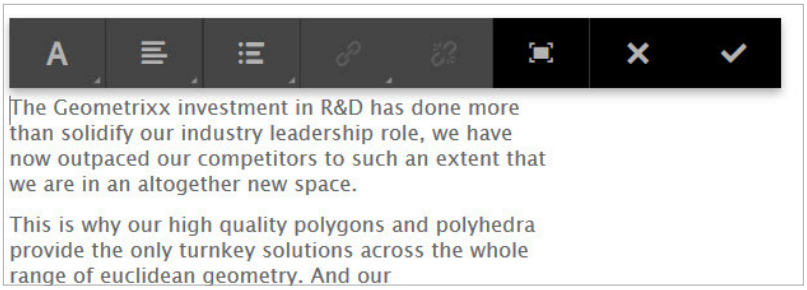
3. Hover the cursor over **English**, and click **Open**.



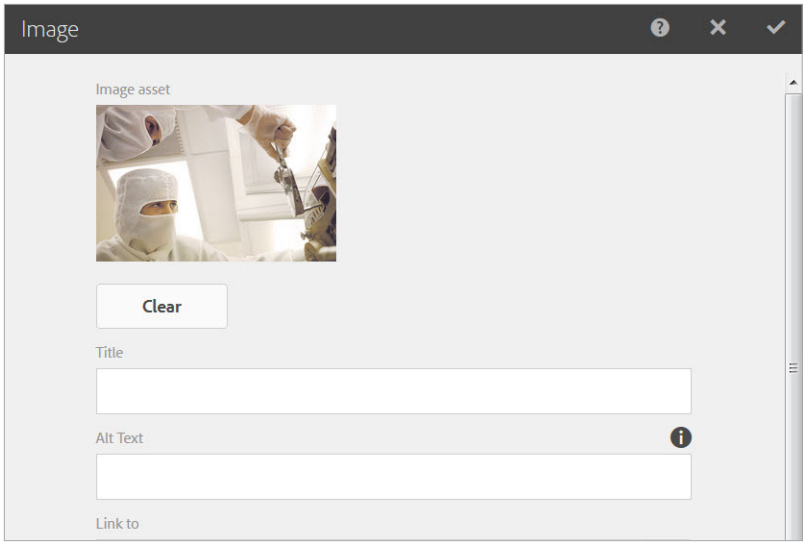
4. When you open a page for the first time, a demo screen appears that helps to familiarize you with the various features. Click **Skip Tour**. Note that the page appears in the touch-optimized UI.



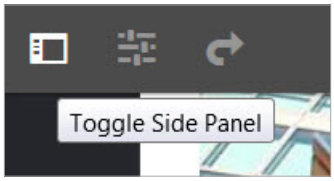
5. Click the paragraph below the second image to select it. Then, click the same paragraph once more to edit it.



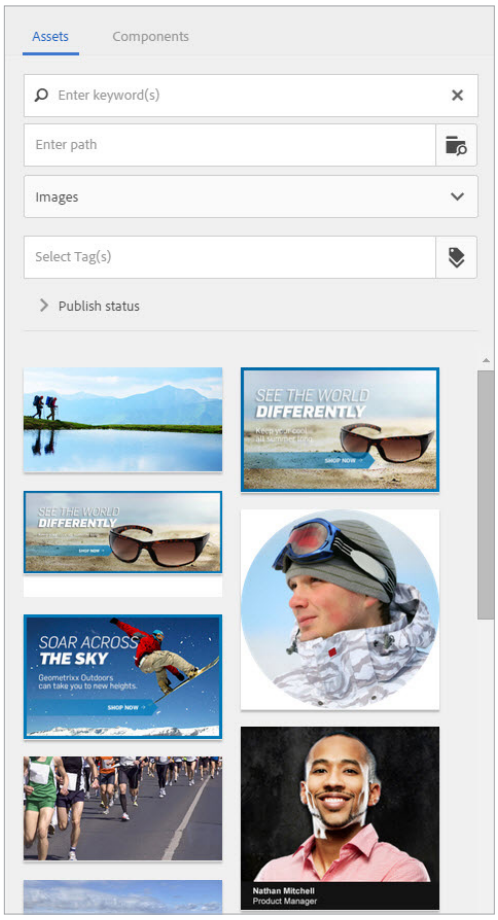
6. Select the image displayed by clicking it, and then click **Configure**. Note that the **Image** dialog box appears. You can update various properties of the image in the dialog box.



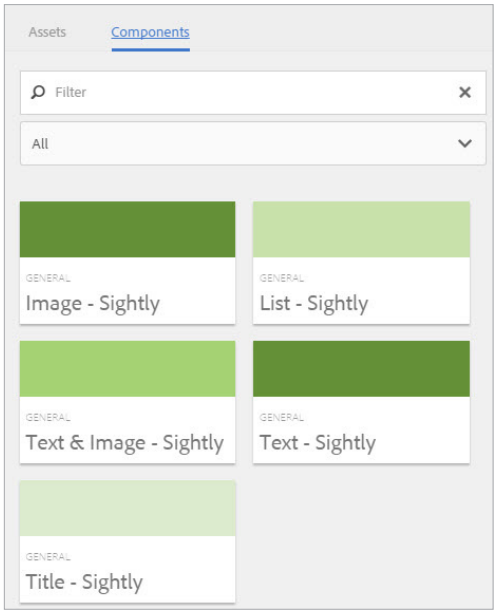
7. Update the **Description** field to `Leaders of Science`.
8. Enter the **Size** values as 500 and 400.
9. Click the check mark to save and close the window.
10. Click the toggle button in the upper-left corner.



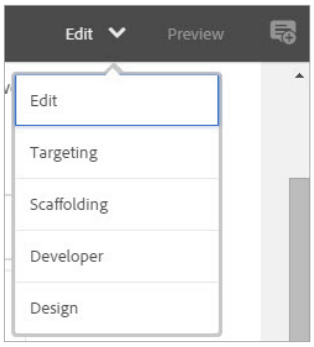
The **Assets** tab appears in the screen as shown here:



11. Click the **Components** tab. This tab displays all available components.



12. Click **Edit** in the upper-right corner of the page and select any of the following modes:

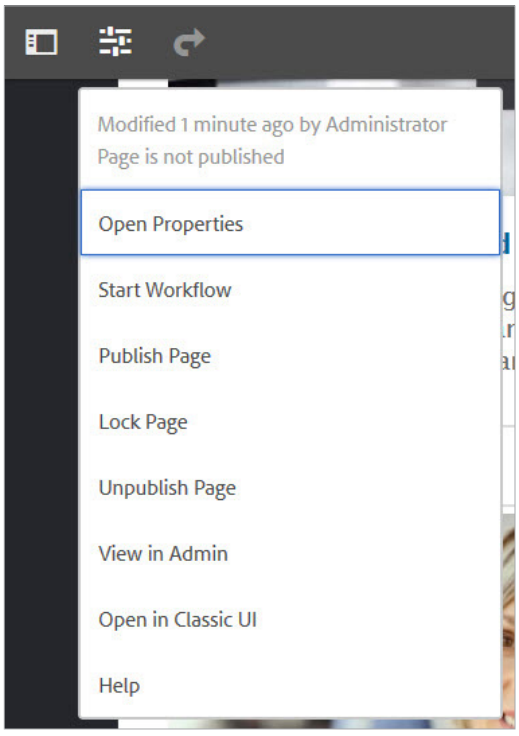


    a. Edit: The default mode that allows you to edit the page
    b. Targeting: Allows you to target your content for Adobe Target
    c. Scaffolding: Provides a form-like interface with content displayed in scaffolds. This enables you to create structurally similar content..
    d. Developer: Provides you with a list of components and related data that allows you to debug components. It also provides you with a debug console and a testing framework.
    e. Design: Enables you to design the page.

13. Click **Preview** to preview the page.

14. Click the **Annotate** icon to add an annotation to a selected location in the site.

15. Click the **Page Information** icon to view the list of actions that can be performed on the page.



16. Click **Open in Classic UI** to see the page in Classic UI.

Developer Mode

Developer mode provides you with the following options:

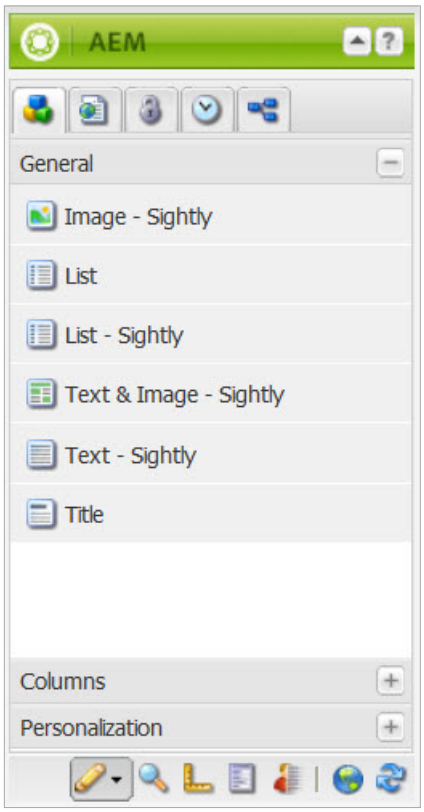
- **Components**: Displays the name of the page component used for the webpage. It also provides you with a list of components used on the page. For each component, the component script and the content path are provided with the time the component took to load. You can click the component script and navigate to the corresponding script in CRXDE Lite. This feature helps you to debug the components easily.
- **Tests**: Allows you to run GUI test cases. You can write test cases using CRXDE Lite and run them here. It displays the result page indicating the test cases that failed or passed.
- **Errors**: Indicates the errors on the webpage

Classic UI

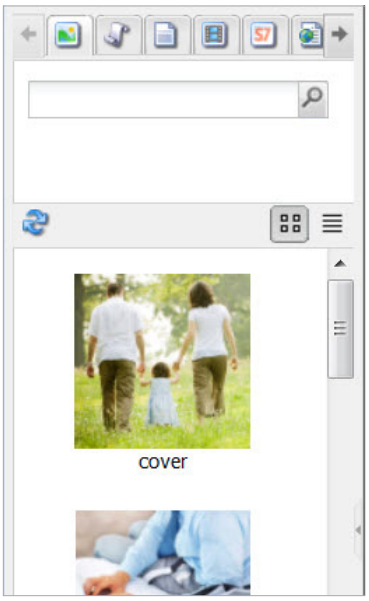
The page that you explored in the previous exercise is displayed as below in the Classic UI:

Classic UI has the following sections:

Sidekick: This is a floating 'inspector' window, sometimes known as a floating palette that appears on the editable page from which new components can be dragged and actions that apply to the page can be executed. The Sidekick window acts as the author's toolbox.



Content Finder: On the left side of each editable page, the Content Finder provides fully searchable, quick access to digital assets such as other images, Flash elements, and documents, as well as other pages and paragraphs. These items can be dragged to the page to position assets or create links to other pages, for example.



AEM Web Consoles

As mentioned above, the functionality of AEM is made available through various specialized web consoles:

- Websites classic console for creating and managing multiple websites
- Websites touch-friendly console for creating and managing website content from a mobile browser
- Digital Assets classic console for managing and organizing various digital assets
- Digital Assets touch-friendly console for managing and organizing digital assets from a mobile browser
- Campaigns console for managing marketing campaigns
- Community console for moderating content of the social network
- Inbox for managing your workflow and other inbox notifications
- Users console for managing user accounts and groups
- Tools console for maintaining and configuring the AEM system
- Tagging console for organizing the tagging taxonomy (tags and their namespaces)

Each of these web consoles is accessible from the AEM Welcome page.

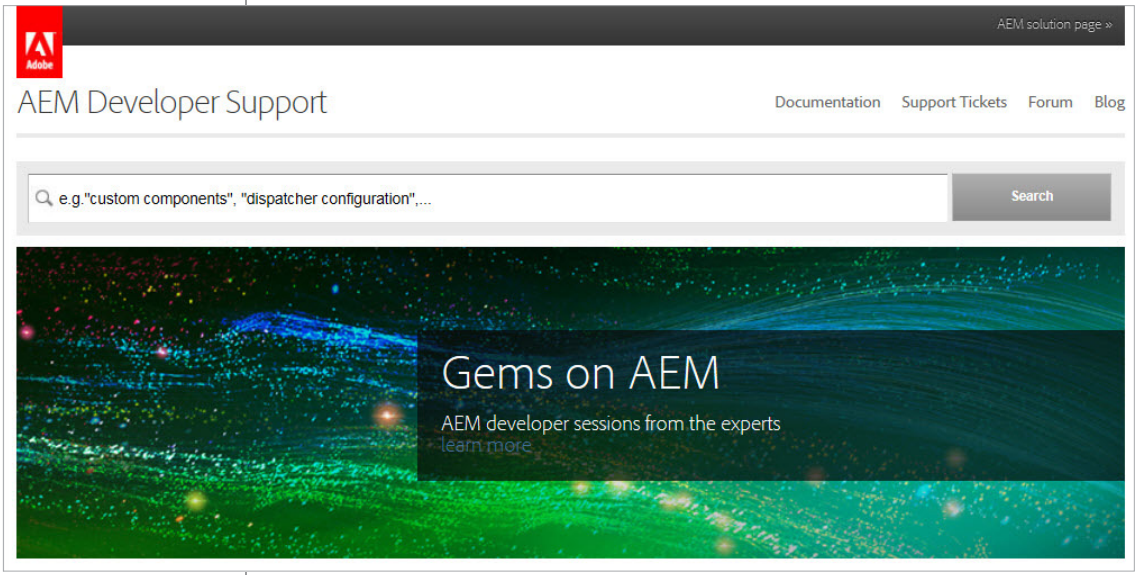
Administration Interfaces

Following are the major administration interfaces in AEM. You will learn more about them in the training:

- OSGi management console: Used to manage bundles and various configurations (http://localhost:4502/system/console)
- CRX Explorer: Used to view and update the CRX repository (http://localhost:4502/crx/explorer)
- CRXDE Lite: You will use this interface frequently in the training. It allows you to perform standard development tasks. It is recommended to use this interface when there is no direct access to the AEM/CRX server. (http://localhost:4502/crx/de)

Developer Community

The AEM and CRX developer community site is your one-stop-shop for all things AEM and CRX. You can access the developer community page at http://dev.day.com.



You have access to the:

- Documentation: Online documentation for authors, developers, and administrators
- Knowledge base: Technical articles focused on 'how to' in reply to specific technical questions
- Customer support portal: Support and ticket management
- Discussion groups: Access to forum discussions among the wide community of developers and administrators using AEM and CRX
- Content-centric applications on top of a Java Specification Request (JSR)

OSGi Framework

Overview

This chapter gives an introduction to the OSGi, Apache Sling frameworks, and Clustering.

OSGi and Apache Sling

AEM is built within an OSGi application framework. OSGi is a dynamic module system for Java that provides a framework within which small, reusable, standardized components can be composed into an application and deployed. The Apache Sling framework is designed to expose the Java Content Repository (JCR) through an HTTP-based REST API. AEM's native functionality and the functionality of any website built with AEM are delivered through this framework.

Objective

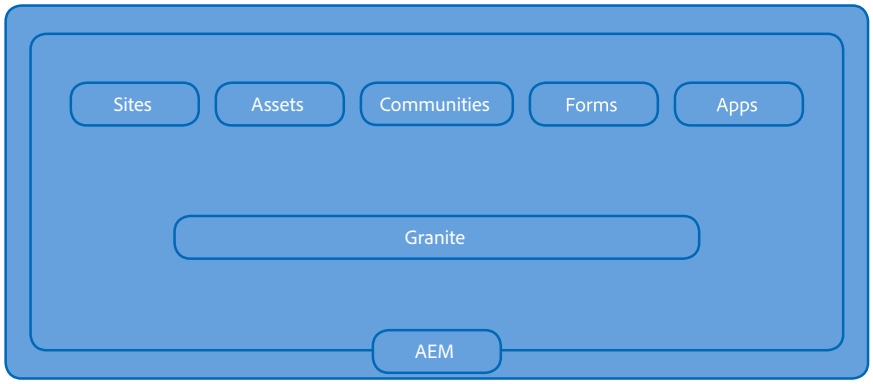
In this chapter, you will be able to:

- understand the concepts of OSGi and Apache Sling.
- describe the AEM functional building blocks.
- descrcibe the Granite platform.
- understand OSGi framework.
- understand about OSGi bundles.

AEM Functional Building Blocks

AEM consists of sets of OSGi bundles that are deployed on the CRX. The AEM application modules use the JCR Application Program Interface (API) to manipulate content in CRX. The AEM application modules sit on top of the AEM shared framework, which contains all application layer functionality that is shared among all the application modules; for example, mobile functionality, multisite manager, taxonomy management, and workflow. In addition to the shared application framework, the AEM applications (Sites, Assets, and so on) share the same infrastructure and same UI framework. With the installation of AEM, you get all applications because they are tightly integrated. Application functionality that is not used or not licensed can be disabled after the initial installation.

Third-party repositories can be integrated with JCR connectors that expose their content into CRX and are available to AEM. Notice that the connectors are plugged in at the content repository level, which allows the content in the external repositories to appear to authors as if the content existed in the local content repository—a true virtual repository.

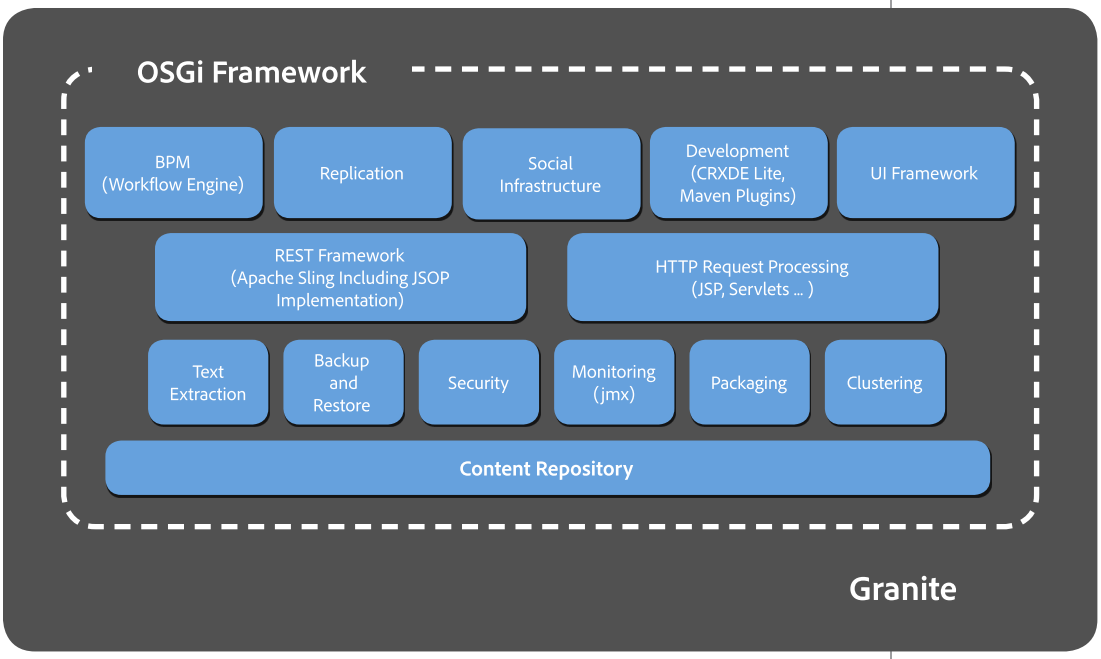


Granite Platform

The AEM application modules sit on top of the AEM shared framework (Granite), which contains functionality that is shared among all the application modules; for example, mobile functionality, multisite manager, taxonomy management, and workflow.

In addition to the shared application framework, the AEM applications (WCM, DAM, Mobile, and so on) share the same infrastructure and same UI framework.

Because 'Everything is Content' and all the content is in the content repository, you will note that clustering and backup is done at the repository level.

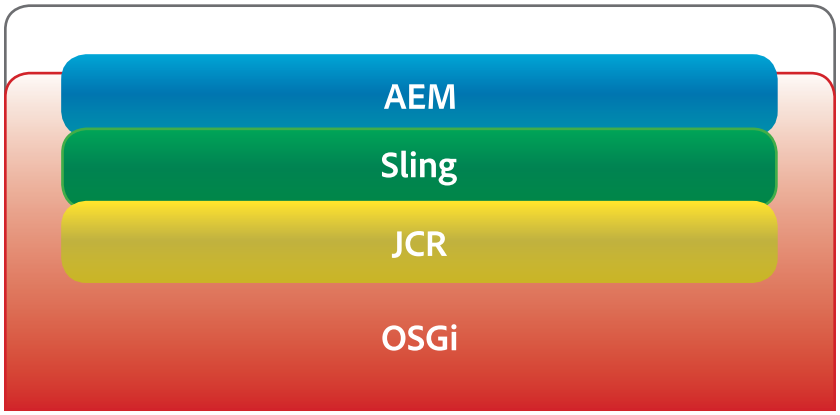


In the Application Runtime of OSGi, specifically Apache Felix, you cannot deploy any code that you wrap as an OSGi bundle. The OSGi runtime hosts Java applications that can access the repository using the JCR API.

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.

Architecture Stack

OSGi is a consortium that has developed a specification to build modular and extensible applications. The OSGi module system allows building applications as a set of reloadable and strongly encapsulated services.



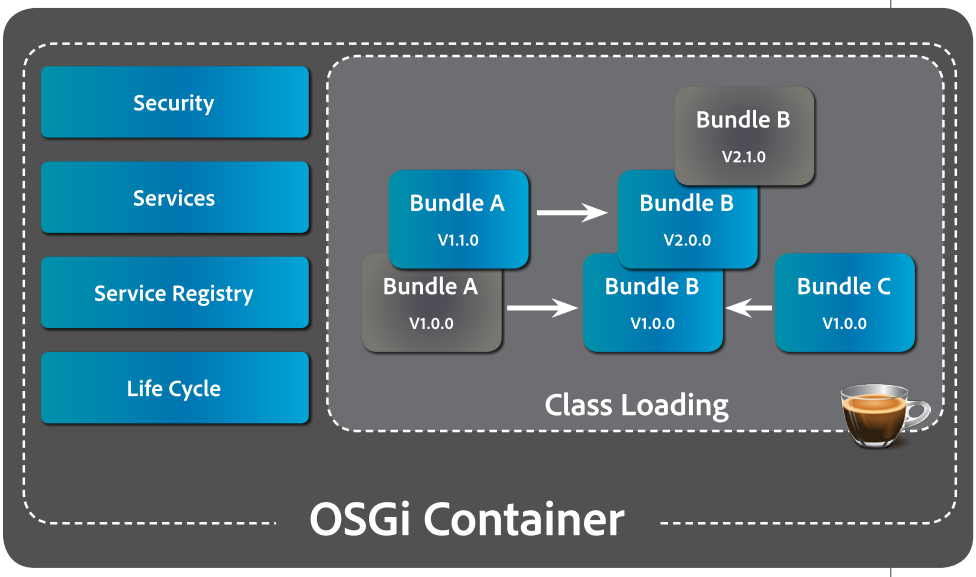
AEM Architectural Stack

Traditional Java application servers use a monolithic approach to application deployment. OSGi bundles run inside an OSGi container. This container manages relations among bundles, which are JAR files that contain extra metadata indicating what services they require and which they provide. The OSGi specifications define a dynamic component system for Java:

- OSGi specifications enable:
  - modularization by the use of a development model where applications are (dynamically) composed of many different (reusable) components.
  - components to hide their implementations from other components while communicating through services, which are objects specifically shared between components.
- Collaborative software environment:
  - The application being created emerges from assembling multiple, reusable modules that have no previous knowledge of each other.

OSGi is a solution to the new DLL-hell, which is the Java class loader. You can have more than one version of any bundle running in the container at the same time. This container resolves any version dependencies.

OSGi is a platform in the Java stack that allows large development teams to be more efficient and provides the ability to replace code pieces (bundles) during normal operation of the application—in other words, without taking the server down.



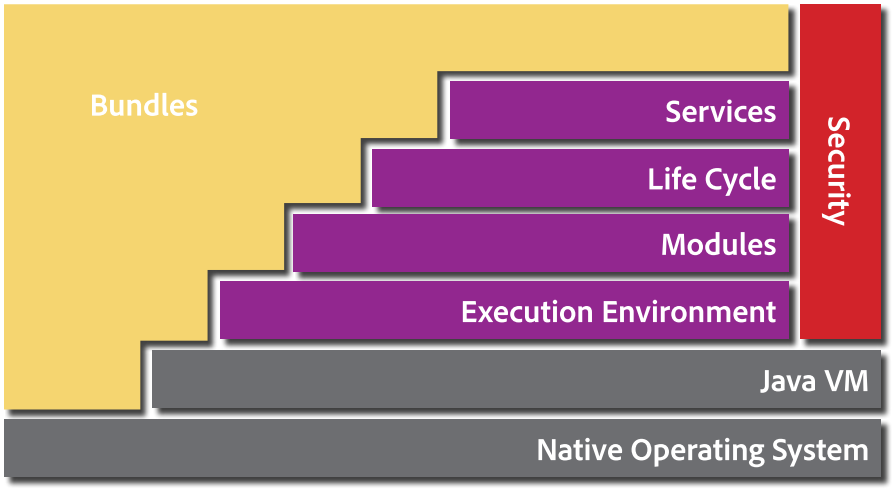
The OSGi container (Apache Felix) that is included in CRX is compliant to version 4.2 of the OSGi specification.

Using an OSGi solution has the following benefits:

- Code is easier to write and test
- Increases reuse
- Build systems become significantly simpler
- Deployment is more manageable
- Bugs are detected early
- Runtime provides an enormous insight into what is running

OSGi Framework

The OSGi Framework is made up of three layers—Module, Life Cycle, and Services—that define how extensible applications are built and deployed.



The responsibilities of the layers are as follows:

- **Module:** The Module layer defines how a module or bundle in OSGi-speak is defined. A bundle is a JAR file whose manifest file has some defined entries. These entries identify the bundle with a symbolic name, a version, and so on. In addition, there are headers that define what a bundle provides—Export-Package; and what a bundle requires to be operative—Import-Package, and Require-Bundle.
- **Life Cycle:** The Life Cycle layer defines the states a bundle may be in and describes the state changes. By providing a class, which implements the Bundle-Activator interface, and which is named in the Bundle-Activator manifest header, a bundle may hook into the life cycle process when the bundle is started and stopped.
- **Services:** For the application to be able to interact, the OSGi Core specification defines the Service layer. This describes a registry for services, which may be shared.

The key principles of OSGi are:

- Universal middleware, dynamic module system
- Service oriented, component-based environment

- Standardized software life cycle management

The implementation of OSGi that the AEM platform makes use of is Apache Felix.

OSGi Bundles

- OSGi bundles can contain compiled Java code, scripts, and content that is to be loaded into the repository, in addition to configuration and/or other files, as needed.
- Bundles can be loaded and installed during normal operations.
- For AEM, bundles are dropped into specially named folders (.../install) in the repository. The Apache Felix Management Console can also manage them.

Additional Information

To learn more about OSGi, go to:

- http://www.osgi.org
- http://en.wikipedia.org/wiki/OSGi

To learn more about Apache Felix specifically, go to:

- http://felix.apache.org

Content Repository

Overview

This chapter provides you with an overview of the content repository in AEM and the best practices involved.

Objective

In this chapter, you will:

- learn about the Java Content Repository (JCR).
- understand the concepts of Apache Jackrabbit.
- explore Adobe CRX.
- understand the underlying repository structure.
- learn about the best practices.

JCR

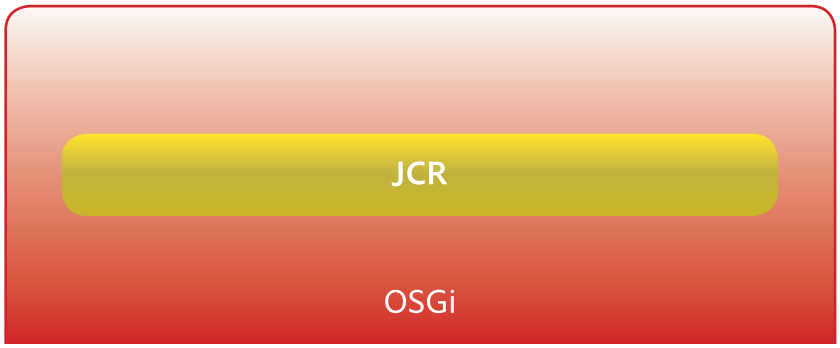
According to JSR-283, the JCR API defines an abstract model and a Java API for data storage and related services commonly used by content-oriented applications.

A JCR is an object database that provides various services for storing, accessing, and managing content. In addition to a hierarchically structured storage, common services of a content repository are versioning, access control, full-text searching, and event monitoring.

The JCR provides a generic application data store for structured and unstructured content. File systems provide excellent storage for unstructured, hierarchical content. Databases provide excellent storage for structured data due to transactional services and referential integrity functionality. The JCR provides the best of both data storage architectures, plus observation, versioning, and full-text search.

An additional advantage of the JCR is support for 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.

AEM is designed to store and retrieve content from any JCR-compliant content repository. In its default configuration, content storage is handled by the CRX repository that comes bundled with AEM. CRX is Adobe System's implementation of the JCR standard, and the version of CRX that comes with AEM supports JCR 2.x.



In addition to CRX, AEM can also work with other JCR repositories such as Apache Jackrabbit and with a number of non-JCR data stores through connectors. Because AEM 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.

JCR defines a Java API for a class of data storage systems called content repositories. A content repository, as defined by JCR, combines features of the traditional relational database with those of a conventional file system, as well as additional services that content-centric applications often need—but that neither file systems nor databases typically provide. See the CRX and JCR documentation for more information about this topic.

JCR Structure

The JCR consists of a set of one or more workspaces, each containing a tree of nodes with associated properties. Each node can have one primary Node type that defines the characteristics of the node. Each node may also be associated with any other node, zero or more mixins, which define additional node characteristics and behaviors.

Beginning with the root node at the top, the hierarchy descends much like the directory structure of a file system. Each node can have zero or more child nodes and zero or more properties. Properties cannot have children but do have values.

The values of properties are where the actual pieces of data are stored. These can be of different types—strings, dates, numbers, binaries, and so forth. The structure of nodes above the properties serves to organize this data according to principles employed by the application using the repository.

Nodes may point to other nodes by using a special reference type property. In this way, nodes in a JCR offer referential integrity and the object-oriented concept of inheritance.

Content Services of the JCR

- Search
- Indexing
- Observation
- Versioning
- Access control/security
- Transactions

Key principles behind the specification of JSR-283 are:

- A common programmatic interface to content repositories
- An API not tied directly to the underlying architecture, data source, or protocol
- Content organization in a repository model
- Hierarchical modeling

The reference implementation for JSR-283 is Apache Jackrabbit. To learn more about Apache Jackrabbit, go to http://jackrabbit.apache.org.

The Adobe implementation of JSR-283 is the Content Repository eXtreme (CRX).

Apache Jackrabbit

The Apache Jackrabbit content repository is a fully conforming implementation of the content repository for Java Technology API (JCR, specified in JSR 170 and 283).

A content repository is a hierarchical content store with support for structured and unstructured content, full-text search, versioning, transactions, observation, and more.

Jackrabbit Oak

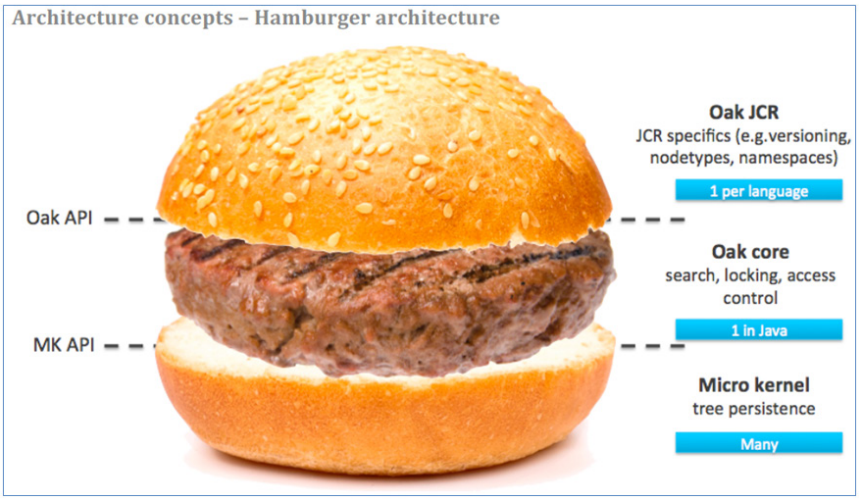
Jackrabbit Oak is an effort to implement a scalable and efficient hierarchical content repository for use as the foundation of modern, efficient websites and other demanding content applications.

Jackrabbit Oak implements the JCR specification. The most used parts of JSR-283 are implemented, but Jackrabbit Oak does not aim to be a reference implementation of JSR-283. AEM 6.1 is built on top of Jackrabbit Oak. Following are the goals of Jackrabbit Oak:

- Scalability
  - Big repositories
  - Distributed repository with many cluster nodes
- Improved write throughput
  - Parallel writes
- Support for many child nodes
- Support for many Access Control Lists (ACLs)

Oak Architecture (Also Known as Hamburger Architecture)

The repository implements standards like JCR, WebDAV, and CMIS are easily accessible from various platforms, especially from JavaScript clients running in modern browser environments. The implementation provides more out-of-the-box functionalities than typical NoSQL databases while achieving comparable levels of scalability and performance.



The burger's top bun: Oak JCR

- Implements the JCR API

The burger's patty: Oak core

- Where most of the heavy lifting takes place
- Adds to the MicroKernel's (MK) tree model (ACLs, search, and indexing)
- Observation
- Exposes essentially a decorated tree model
- Mostly transforms JCR semantics into tree operations
- Also contains 'Commit hooks' that implement JCR constraints; for example, Node types
- Is now implemented for JCR. However, non-Java implementations are possible and are part of the concept.

The burger's bottom bun: MicroKernel

- Implements a tree model (nodes and properties)
- Exposes the MicroKernel API

MicroKernels

The Oak Microkernel API provides an abstraction layer for the actual storage of the content. The Microkernel API is completely based on Strings. It uses the JSOP format, which is a lightweight HTTP protocol for manipulating JSON-based object models.

Currently, Oak has two main Microkernel implementations:

- DocumentMK
- SegmentMK

DocumentMK

The Oak Document MicroKernel manages JCR content by storing each content node in a separate document.

Implementation

DocumentMK is a concept and is not something that is directly implemented. Apart from MongoMK, AEM 6.1 includes more options for DocumentMK such as DB2 and Oracle 12c. AEM 6.1 also supports MySQL, MariaDB, and Microsoft SQL Server on experimental support.

Documents

Documents in a DocumentMK are usually stored as JSON. A sample JSON representation of a document in DocumentMK is:

```
{
    "_deleted" : {
        "r13f3875b5d1-0-1" : false" }
    "_id" : "1:/node",
    "_lastRev" : {
        "r0-0-1" : "r13f38818ab6-0-1"
},
"_modified" : NumberLong(274208516), "_modCount" :
    NumberLong(2), "_revisions" : {
    "r13f3875b5d1-0-1" : "c", "r13f38818ab6-0-1" : "c"
},
"prop" : {
    "r13f38818ab6-0-1" : "\"foo\""  }
}
```

Document nodes have two types of fields:

- Simple fields are stored as key/value pairs. In the example above, _id, _modified, and _modCount are simple fields.
- Versioned fields are kept in sub-documents where the key is a revision paired with the value of this revision.

SegmentMK

SegmentMK is an Oak storage backend that stores content as various types of records within larger segments. One or more journals are used to track the latest state of the repository. These are the principles on which SegmentMK is designed:

- Immutability: The segments are immutable, and therefore, it is easy to cache frequently accessed segments. This feature simplifies backups.
- Compactness: The size is optimized in a way that reduces the IO costs and fits the content in caches as much as possible.
- Locality: One segment usually stores related records. With this, the cache misses are avoided; for example, if a client wants to access more nodes that are related per session.

Implementation

TarMK is the supported persistence mechanism of the SegmentMK. It is based on tar files in the local files system.

Segments

Each segment usually contains a continuous subset of a content tree; for example, a node with its properties and closest child nodes. Each segment is defined by UUID and can be up to 256 KB. Each segment keeps a list of the UUIDs of all other segments it references. The UUID contains data about the type of the segment, like in this example:

`xxxxxxxx-xxxx-4xxx-Axxx-xxxxxxxxxxxx`: data segment UUID

`xxxxxxxx-xxxx-4xxx-Bxxx-xxxxxxxxxxxx`: bulk segment UUID

As you can see from the example, there are two types of segments—data and bulk segments. Data segments can contain any type of records and may refer to content in other segments. Bulk segments are only used for storing large binary values and can contain only raw binary data, interpreted as a sequence of block records.

Adobe CRX

CRX implements the Content Repository API for Java Technology (JCR). This standard defines a data model and application programming interface (that is, a set of commands) for content repositories. Content is available through a standardized API:

- JSR-283
- JCR API
  - Node node.addNode("JCR")
  - Node node.getNode("JCR")
  - Node node.setProperty("opinion","excellent and recommended")
  - Node node.getProperty("opinion").getString()

Built-in Protocols/APIs for the CRX Platform

You can add, consume, and manage content with these interfaces:

- Java Content Repository API—a complete JCR 2.0 implementation
- Content Management Interoperability Services—CMIS 1.0
- WebDAV—with versioning, access control, and search
- Windows Network File Share—CIFS/SMB
- RESTful Web API for JavaScript and Flash/Flex
- Java Remoting with RMI and HTTP
- LDAP and any JAAS plugin
- Native repository interface through a Virtual Repository, for example, Microsoft SharePoint

Repository Structure

You can use CRXDE Lite to explore the logical structure of the repository as defined by the JCR API. This interface is going to be useful to you as a developer. You will be able to look at the structures that your code writes into the repository. If you take a close look at the repository structure, you will notice that the repository has several major areas:

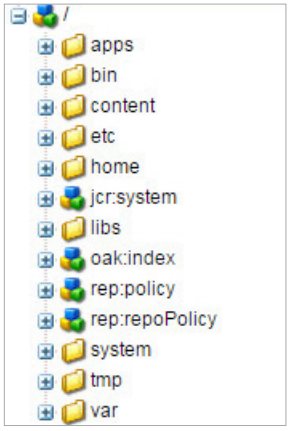
- **/var:** Files that change and are updated by the system, such as audit logs, statistics, and event handling
- **/libs:** Libraries and definitions that belong to the core of AEM. The sub-folders in /libs represent the out-of-the-box AEM features, such as search or replication.

The content in /libs should not be modified because it affects the way AEM works. Features specific to your website should be developed under /apps.

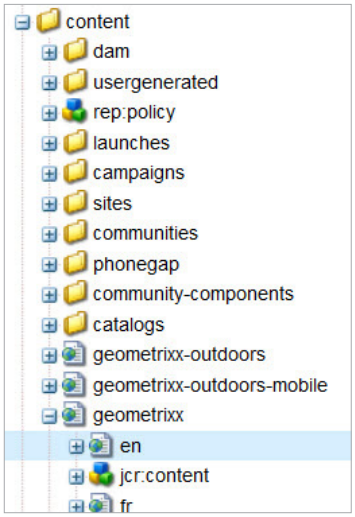
- **/etc:** Utilities and tools. See the documentation for the Tools Console for detailed information.
- **/apps:** Application related. The custom templates and component definitions specific to your website. The components that you develop may be based on out-of-the-box components available at `/libs/foundation/components`.
- **/content:** Content created for your website
- **/tmp:** A temporary working area
- **/home:** User and group information

Exercise 3.1
Familiarize Yourself With a Repository Structure

1. Log in to CRXDE Lite (http://localhost:4502/crx/de/index.jsp).
2. Observe the left pane to see the content structure that was mentioned earlier. You can expand the nodes to see the contents.

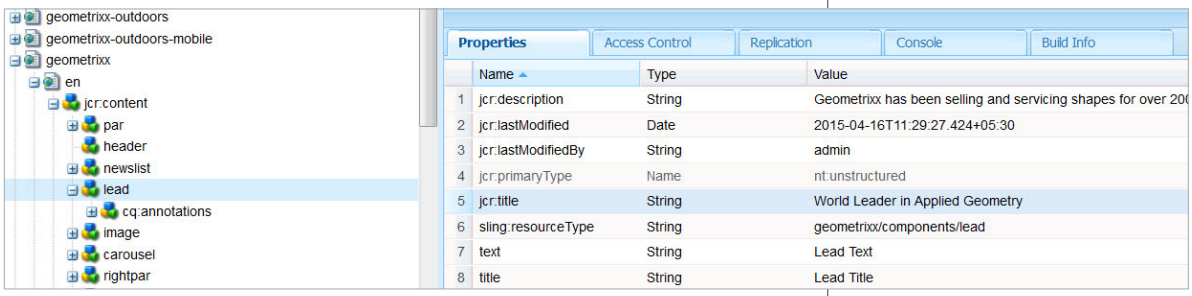


3. Navigate to **content** > **geometrixx** > **en**.



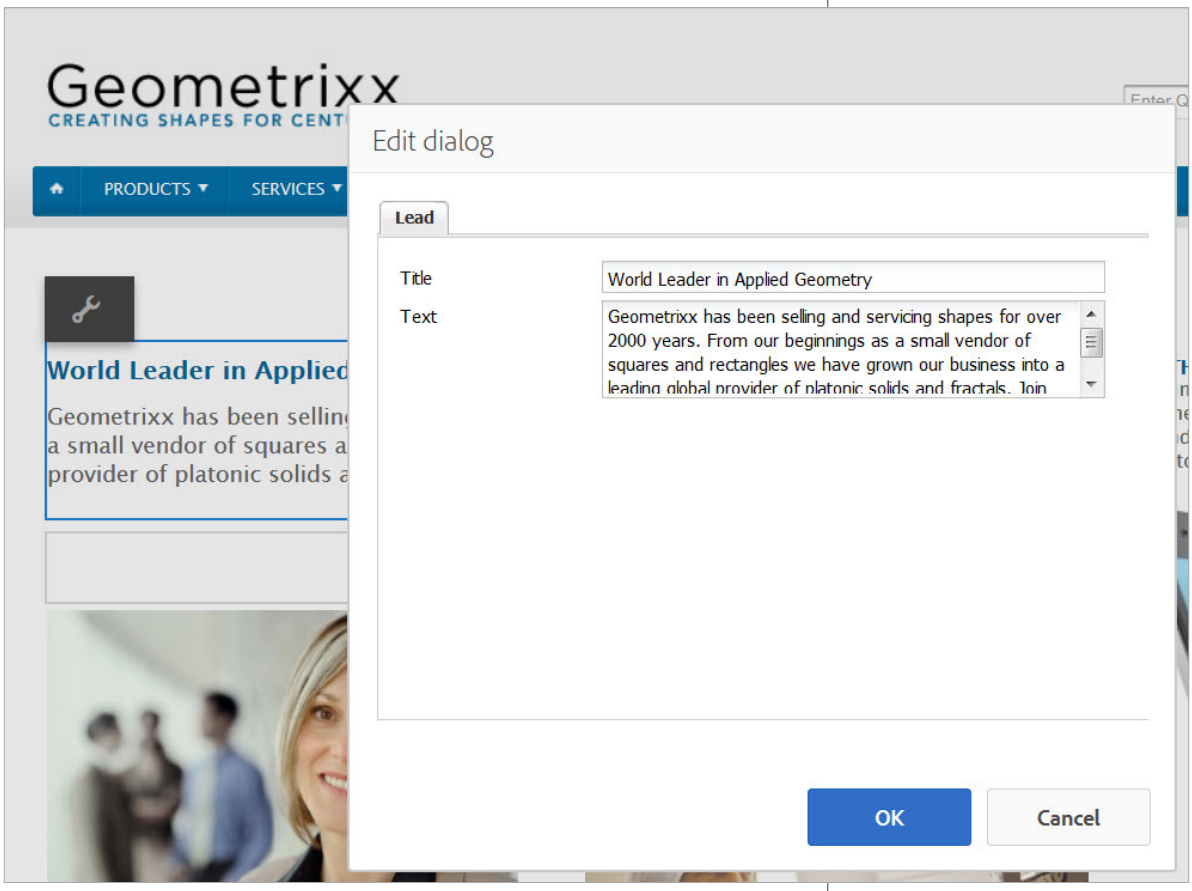
4. Expand the `en` node, and then expand the `jcr:content` node within it.

5. Under the `jcr:content` node, select the `lead` node. Note that all the content in the page appears as various nodes. Observe that it displays the leading title of your page.



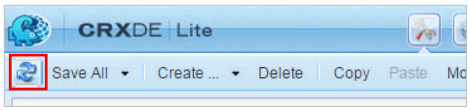
6. Open the page in a UI of your choice (classic or touch-optimized UI). Following is the URL for touch-optimized UI:
http://localhost:4502/editor.html/content/geometrixx/en.html

7. Double-click the lead component to open the **Edit dialog** box.

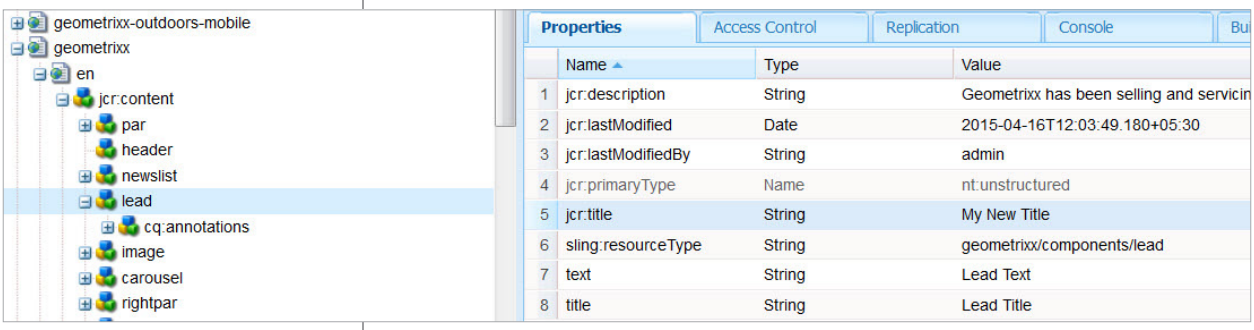


8. Update the title to `My New Title` and click **OK**.

9. Go to CRXDE Lite, and select the `lead` node. Click the **Refresh** icon displayed in the upper-left corner.



Note that the `jcr:title` property is changed in CRXDE Lite.



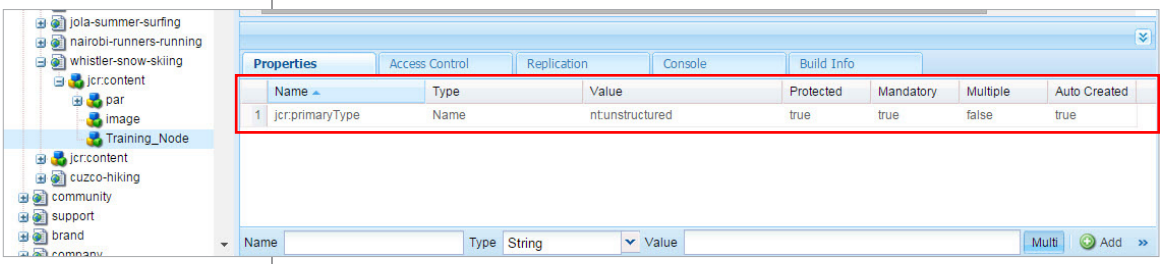
Exercise 3.2
Create a Node And Add Properties

The goal of this exercise is to learn how to create a node in CRXDE Lite, and add properties to it. There would be no change to any content of the site.

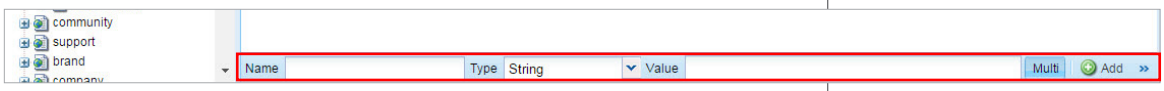
1. Open **CRXDE Lite**.
2. Navigate to `/content/geometrixx-outdoors/en/activities/whistler-snow-skiing/jcr:content`.
3. Right-click the `jcr:content` node and select **Create Node...**.
4. Enter the following details for the node and click **OK**.

Property	Value
Name	Training_Node
Type	nt:unstructured

5. Click **Save All**.
6. Select **Training_Node**. You can see its properties in the panel on the right.



To add a property to the node, enter the values in the lower section of the **Properties** tab.



7. Add the following property to the node:

Property	Value
Name	SingleValueProperty
Type	String
Value	Hello World

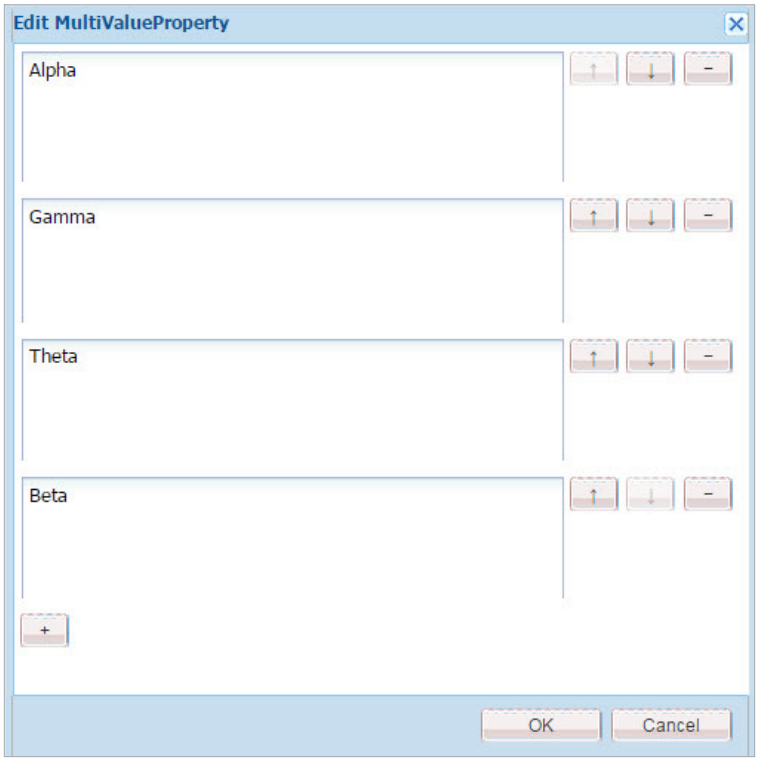
8. Click **Add**.

9. Add another property to the node with the following details:

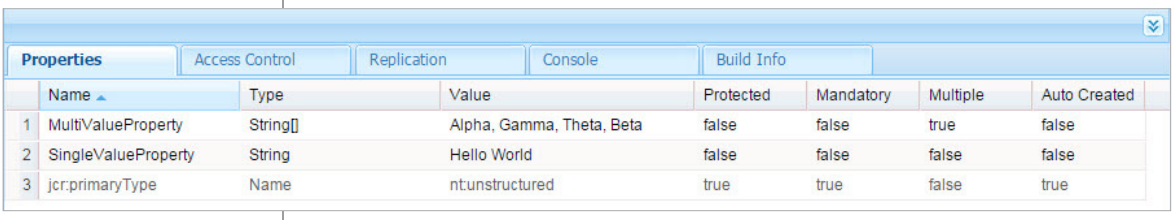
Property	Value
Name	MultiValueProperty
Type	String
Value	Alpha

10. Click **Multi**, and then click **Add**.

11. In the **Edit** dialog box, add additional values by clicking the **+** button. Add the values "Gamma," "Theta," "Beta," and click **OK**.



12. Click **Save All**. The node now has two additional properties.



	Name	Type	Value	Protected	Mandatory	Multiple	Auto Created
1	MultiValueProperty	String[]	Alpha, Gamma, Theta, Beta	false	false	true	false
2	SingleValueProperty	String	Hello World	false	false	false	false
3	jcr:primaryType	Name	nt:unstructured	true	true	false	true

Web Framework

Overview

This chapter includes a detailed description and exercises on Representational State Transfer (REST) and Apache Sling.

Objective

In this chapter, you will:

- understand REST architectural style.
- learn about Apache Sling.

REST

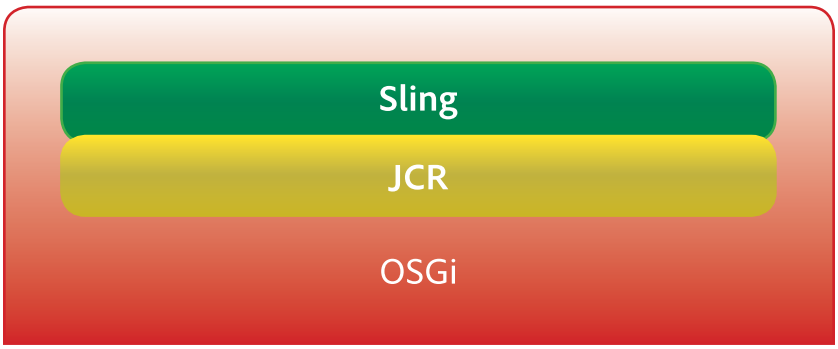
Addressable resources present a uniform interface that allows transfers of state; for example, reading and updating of the resource's state. The best example of a RESTful architecture is the web, where resources have URIs and the uniform interface is HTTP.

For most cases, a framework like HTTP (addressable resources plus 'verbs' plus standard ways to transmit metadata) is all you need to build a distributed application. HTTP is a rich application protocol that gives you capabilities like content negotiation and distributed caching. RESTful web applications try to leverage HTTP in its entirety using specific architectural principles.

1. Resource-oriented: Any piece of information (content object)—news entry, product description, or photo is a resource.
2. Addressable resources: With REST over HTTP, every object will have its own specific URI. From the URI, you know how to communicate with the object, find its location in the network, and identify it on the server it resides.
3. A uniform, constrained interface: When using REST over HTTP, stick to the methods provided by the protocol. This means following the meaning of GET, POST, PUT, and DELETE as defined with no additional methods or services.
4. Representation oriented: You interact with services using representations of that service. An object referenced by one URI can have different formats or renderings available. Different clients need different formats—Ajax may need JSON, a Java application may need XML, a browser may need HTML. You may also need a print-friendly rendering.
5. Communicate statelessly: REST, as implemented in HTTP, tends to be stateless; for example, it does not use cookies, and clients need to re-authenticate on every request. Client session data is not stored on the server. Session-specific information is held and maintained by the client and transferred to the server on each request, as needed. Stateless applications are easier to scale.

Apache Sling

Apache Sling is a web framework that uses a JCR, such as Apache Jackrabbit or Adobe CRX to store and manage content. Sling applications use scripts or Java servlets, selected based on simple naming conventions, to process HTTP requests in a RESTful way.



REST refers to the software architectural style on which the World Wide Web is based. It describes the key elements that make the web work, and so provides a set of principles for designing web-based software. When designing an API to be used over the web, it therefore makes sense to adhere to these best practices. Sling does just that.

The Sling application is built as a series of OSGi bundles and makes heavy use of a number of OSGi core and compendium services. The embedded Apache Felix OSGi framework and console provide a dynamic runtime environment, where code and content bundles can be loaded, unloaded, and reconfigured at runtime.

As the first web framework dedicated to JSR-283 JCRs, Sling makes it easy to implement simple applications, while providing an enterprise-level framework for more complex applications.

Being a REST framework, Sling is oriented around resources, which usually map into JCR nodes. Traditional web applications select a processing script based on the URL and then attempt to load data to render a result. Sling request processing, however, takes what seems like an inside-out approach to handling requests. A request URL is first resolved to a resource, and then based on the resource (and only the resource), it selects the actual Servlet or script to handle the request.

Everything is a Resource

The resource is one of the central parts of Sling. Extending from JCR's 'Everything is Content', Sling assumes that 'Everything is a Resource'. A resource is Sling's abstraction of the object addressed by the URI. Sling resources usually map to a JCR node.

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.

Exercise 4.1
Access Data in Different Formats

1. Access the following page:
   http://localhost:4502/content/geometrixx/en.html
2. Change the URL as follows, and access the page:
   http://localhost:4502/content/geometrixx/en.xml
   Note that you are getting an xml representation of the page.

This XML file does not appear to have any style information associated with it. The document tree is shown below.


```
− <jcr:content jcr:primaryType="cq:PageContent" cq:designPath="/etc/designs/geometrixx"
  cq:lastModified="2015-05-22T18:12:18.522+05:30" cq:lastModifiedBy="admin"
  cq:template="/apps/geometrixx/templates/homepage"
  jcr:created="2015-05-22T17:40:41.257+05:30" jcr:createdBy="admin" jcr:title="English"
  pageTitle="GeoMetrixx" sling:resourceType="geometrixx/components/homepage">
   − <par jcr:primaryType="nt:unstructured" sling:resourceType="foundation/components
     /parsys">
       <colctrl jcr:primaryType="nt:unstructured"
       jcr:created="2010-08-23T22:02:24.480+02:00" jcr:createdBy="admin"
       jcr:lastModified="2010-08-23T22:02:35.400+02:00" jcr:lastModifiedBy="admin"
       layout="2;cq-colctrl-lt0" sling:resourceType="foundation/components/parsys/colctrl"/>
     − <image jcr:primaryType="nt:unstructured" fileReference="/content/dam/geometrixx
       /portraits/jane_doe.jpg" jcr:created="2010-08-23T22:03:39.419+02:00"
       jcr:createdBy="admin" jcr:lastModified="2010-10-31T20:39:50.135Z"
       jcr:lastModifiedBy="admin" sling:resourceType="foundation/components/image"
       width="340">
        − <file jcr:primaryType="nt:file" jcr:created="2015-05-22T17:40:41.269+05:30"
          jcr:createdBy="admin">
            <jcr:content jcr:primaryType="nt:resource" jcr:uuid="996a0214-c7ee-4356-9202-
            bf14bc7379bc" jcr:data="/9j/4AAQSkZJRgABAQEASABIAAD
```

3. Change the URL as follows, and access the page.
   http://localhost:4502/content/geometrixx.5.json
   Note that you are getting a JSON representation of the page. The selector 5 in the URL provides you with five levels of JSON.

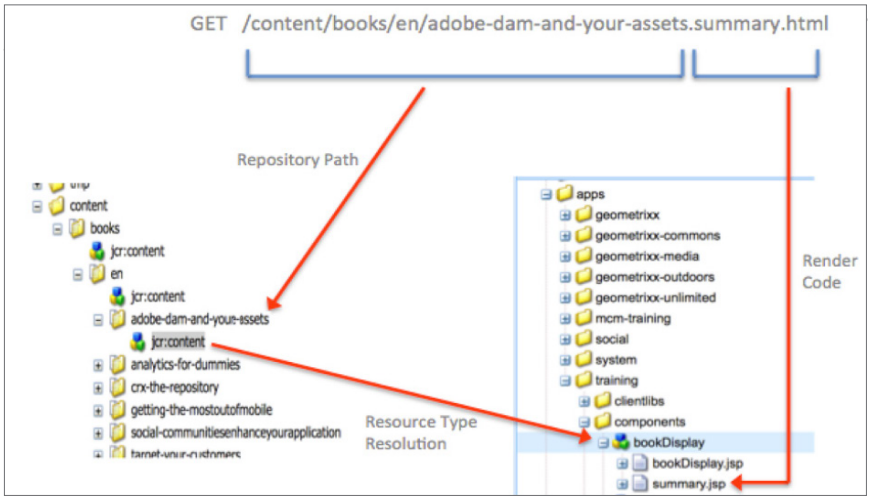

```
{
    "jcr:primaryType": "cq:Page",
    "jcr:createdBy": "admin",
    "jcr:created": "Fri May 22 2015 17:40:41 GMT+0530",
  ▾ en: {
        "jcr:primaryType": "cq:Page",
        "jcr:createdBy": "admin",
        "jcr:created": "Fri May 22 2015 17:40:41 GMT+0530",
      ▾ "jcr:content": {
            "jcr:primaryType": "cq:PageContent",
            "jcr:createdBy": "admin",
            "jcr:title": "English",
            "cq:template": "/apps/geometrixx/templates/homepage",
            "jcr:created": "Fri May 22 2015 17:40:41 GMT+0530",
            "cq:lastModified": "Fri May 22 2015 18:12:18 GMT+0530",
            pageTitle: "GeoMetrixx",
            "sling:resourceType": "geometrixx/components/homepage",
            "cq:designPath": "/etc/designs/geometrixx",
            "cq:lastModifiedBy": "admin",
          ▾ par: {
                "jcr:primaryType": "nt:unstructured",
                "sling:resourceType": "foundation/components/parsys",
              ▾ colctrl: {
                    "jcr:primaryType": "nt:unstructured",
                    "jcr:createdBy": "admin",
                    "jcr:lastModifiedBy": "admin",
                    layout: "2;cq-colctrl-lt0",
                    "jcr:created": "Tue Aug 24 2010 01:32:24 GMT+0530",
                    "jcr:lastModified": "Tue Aug 24 2010 01:32:35 GMT+0530",
                    "sling:resourceType": "foundation/components/parsys/colctrl"
                },
```


Sling Resolution

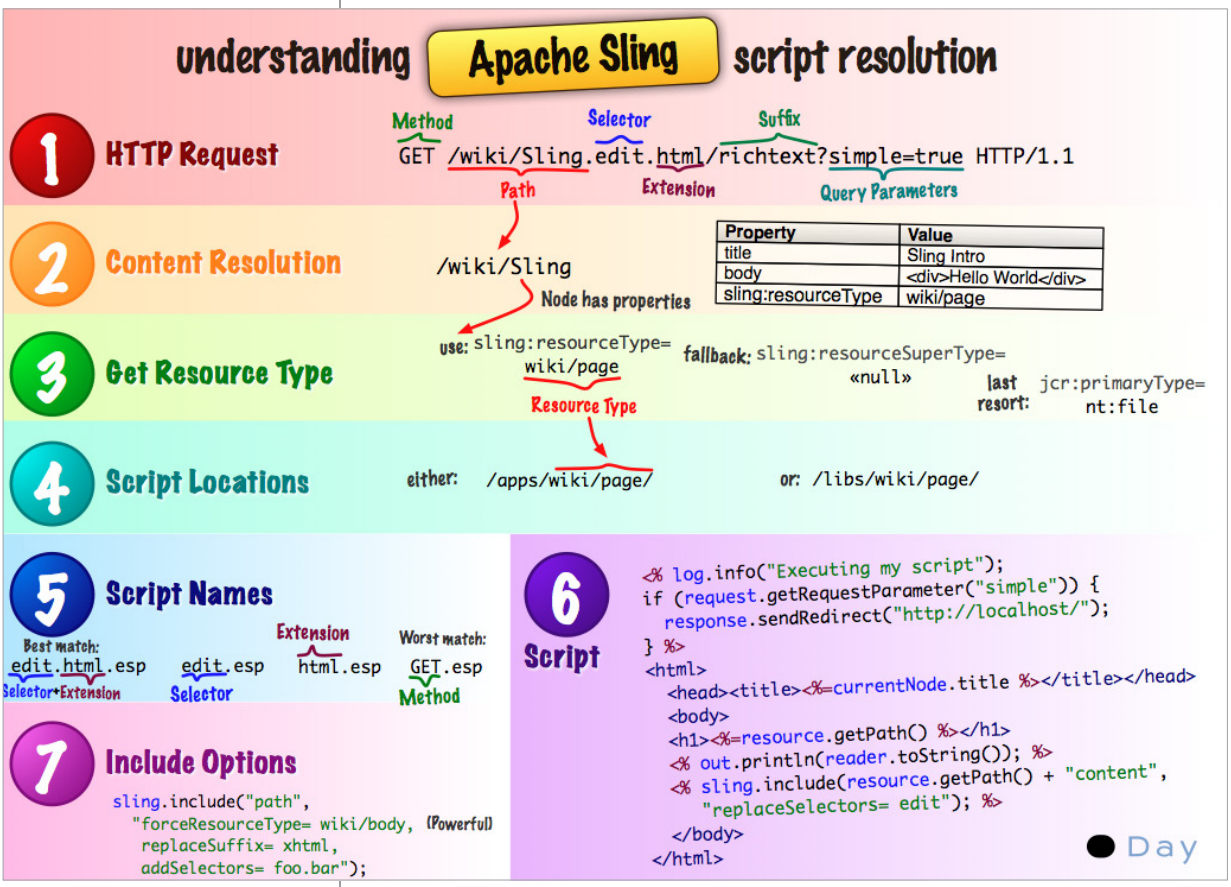
AEM is built using Apache Sling, a web application framework based on REST principles that provides easy development of content-oriented applications. Sling uses a JCR repository, such as Apache Jackrabbit or Day's CRX, as its data store.

Sling is included in the installation of AEM. Sling was originally designed and implemented by Day Software (now Adobe Systems). Sling has since been contributed to the Apache Software Foundation—further information can be found at Apache (http://sling.apache.com).

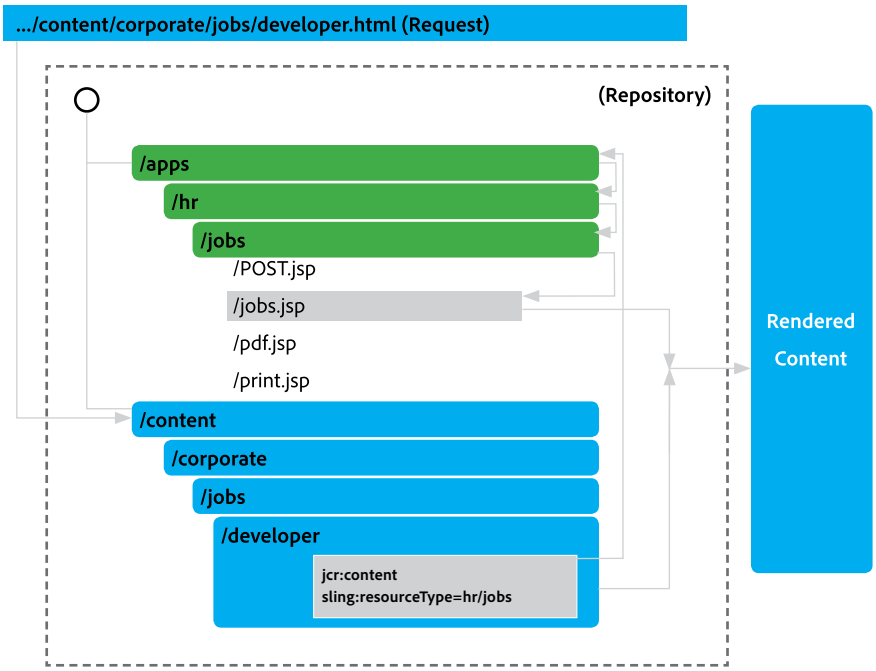
When using Sling, the type of content to be rendered is not the first processing consideration. Instead, the main consideration is whether the URL resolves to a content object for which a script can then be found to perform the rendering. This provides excellent support for web content authors to build pages, which are easily customized to their requirements. The advantages of this flexibility are apparent in applications



The following diagram explains the Sling script resolution. It shows how to get from HTTP request to content node; from content node to resource type; from resource type to script—and what scripting variables are available.



The following diagram illustrates another example of how a script is resolved.



With Sling, you specify which script renders a certain entity (by setting the sling:resourceType property in the jcr:content node). This mechanism offers more freedom than one in which the script accesses the data entities (as an SQL statement in a PHP script would do) because a resource can have several renditions. If multiple scripts apply for a given request, the script with the best match is selected. The more specific a match is, the better it is; in other words, the more selector matches the better, regardless of any request extension or method name match.

The Resolution Process

The Servlet Resolution processes four elements of a SlingHttpServletRequest:

1. The **request selectors** as retrieved through `request.getRequestPathInfo().getSelectorString()`.

   The selector string is turned into a relative path by replacing all separating dots with forward slashes. For example, the selector string print.a4 is converted into the relative path print/a4.

2. The **request extension** as retrieved through `request.getRequestPathInfo().getExtension()` if the request method is GET or HEAD and the request extension is not empty.

3. The **resource** type as retrieved through `request.getResource().getResourceType()`. Because the resource type may be a Node type such as nt:file, the resource type is mangled into a path by replacing any colons with forward slashes. Also, any backslashes are replaced with forward slashes. This should give a relative path. Of course, a resource type may also be set to an absolute path.

4. The **request method** name for any request method except GET or HEAD or if the request extension is empty.

When dealing with multiple selectors, it can often be simplified. Instead of creating multiple folders matching selectors, use one single JSP, get all the selectors on the request by using SlinghttpServletRequest.gerSectionRequestPathInfo.getSelectors(), and then in that one JSP, use a series of "if..else" statements.

```
if (selector1.equals("xxx")) then….
else if (selector2.equals("yyy")) then…
else if (selector3.equals("zzz")) then…
```

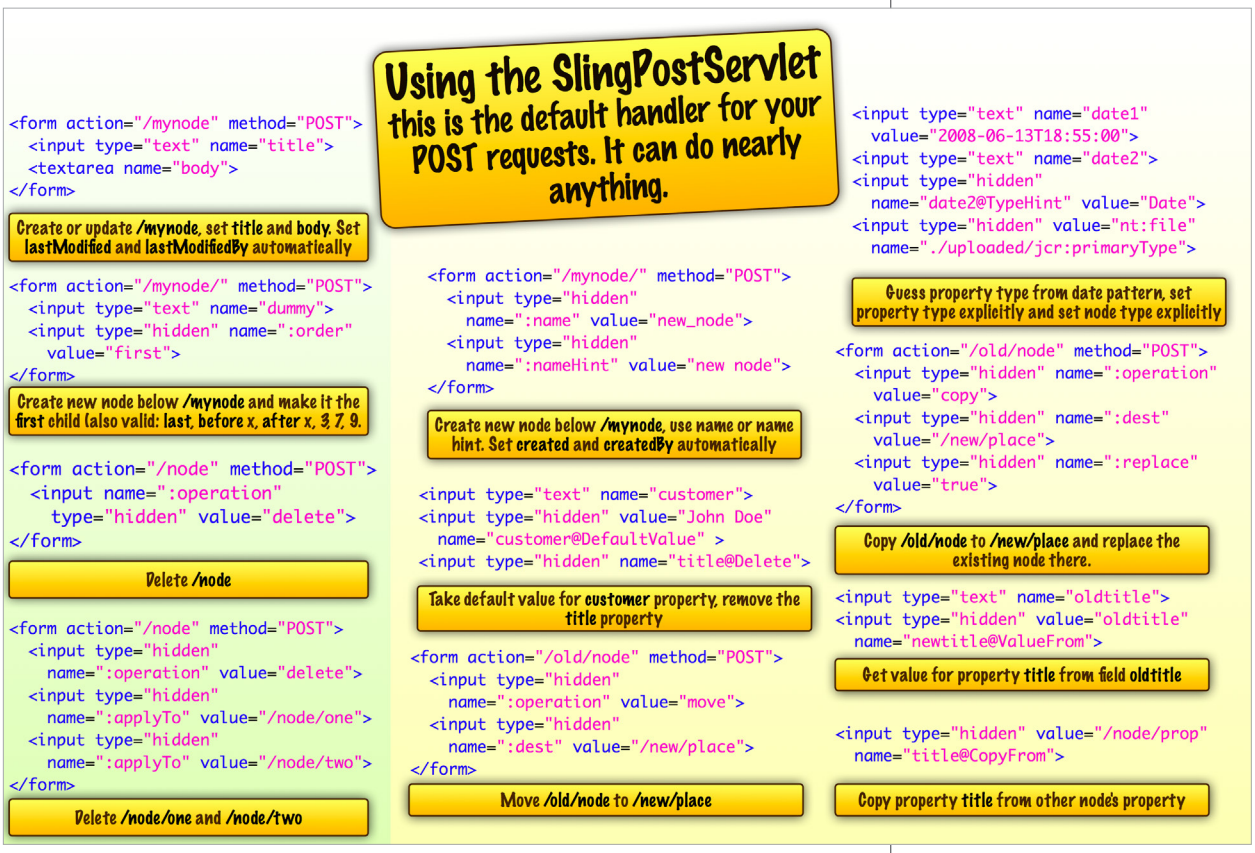
Create one single JSP to test all the different cases.

URI	Resource Path	Selectors	Extension	Suffix Path
/a/b	/a/b	null	null	null
/a/b.html	/a/b	null	html	null
/a/b.s1.html	/a/b	s1	html	null
/a/b.s1.s2.html	/a/b	s1.s2	html	null
/a/b/c/d	/a/b/c/d	null	null	null
/a/b./c/d	/a/b	null	null	/c/d
/a/b.html/c/d	/a/b	null	html	/c/d
/a/b.s1.html/c/d	/a/b	s1	hmtl	/c/d
/a/b.s1.s2.html/c/d	/a/b	s1.s2	html	/c/d
/a/b/c/d/s.txt	/a/b/c/d	s	txt	null

The resource type is used as a (relative) parent path to the Servlet while the request extension or request method is used as the Servlet (base) name. The Servlet is retrieved from the Resource tree by calling the ResourceResolver.getResource(String) method, which handles absolute and relative paths correctly by searching relative paths in the configured search path.

SlingPostServlet

The following diagram explains all the hidden, but powerful, request parameters you can use when dealing with the SlingPostServlet, the default handler for all POST requests that gives you endless options for creating, modifying, deleting, copying, and moving nodes in the repository.



Developer Tools and Scripting Languages

Overview

The AEM platform supports a variety of developer tools and scripting languages to meet different needs. Sightly was the new AEM templating system introduced with AEM 6.0. It replaces Java Server Pages (JSP) and ECMAScript Server Pages (ESP) as the preferred UI templating system for AEM. Note that AEM supports JSP and ESP as templating languages.

Objective

In this chapter, you will:

- explore the features and functionalities of CRXDE Lite.
- work with packages using the Package Manager.
- install and work with Brackets plugins.
- learn about Sightly and its syntax and features.

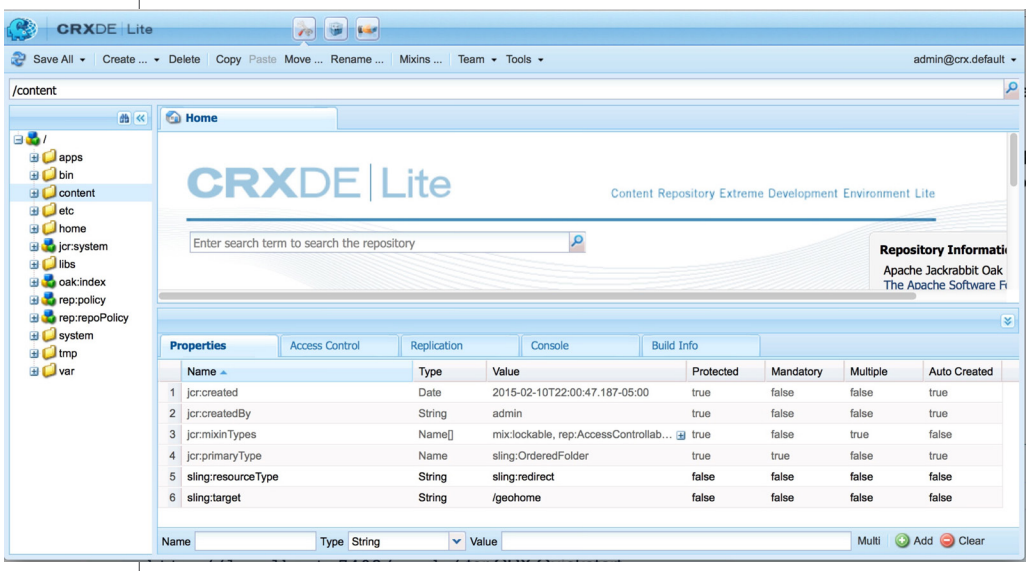
Developer Tools

There are 3 key developer tools available for the AEM platform:

- CRXDE Lite
- The Sightly Extension for Brackets
- The AEM Developer Tools for Eclipse

CRXDE Lite

CRXDE Lite is embedded into AEM and enables 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 lightweight development tasks.

CRXDE Lite lacks some features that are desirable for extensive development tasks such as code completion, syntax highlighting, and a dedicated debugger.

From the touch-optimized UI, CRXDE Lite is available at **Tools** > **CRXDE Lite**. It can also be accessed directly at http://localhost:4502/crx/de/index.jsp.

Package Manager

A package is a zip file that contains the content in the form of a file-system serialization (called vault serialization), which represents the content from the repository as an easy-to-use-and-edit representation of files and folders. Packages can include any or all project-related data.

Additionally, it contains vault meta information, including a filter definition and import configuration information. Additional content properties can be included in the package, such as a description, a visual image, or an icon. These properties are for the content package consumer and are for informational purpose only.

You can perform the following actions with packages:

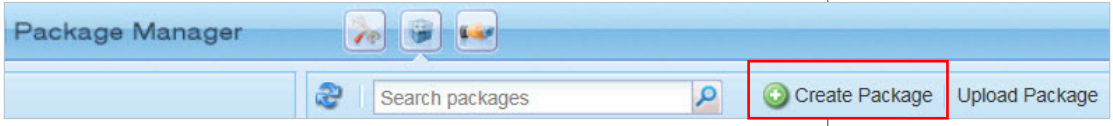
- Create new packages
- Modify existing packages
- Build packages
- Upload packages
- Install packages
- Download packages from the package share library
- Download packages from AEM to a local machine
- Apply package filter

In the following exercises, you will learn how to create packages, as well as upload and install packages.

Exercise 5.1
Create a content package of an existing project

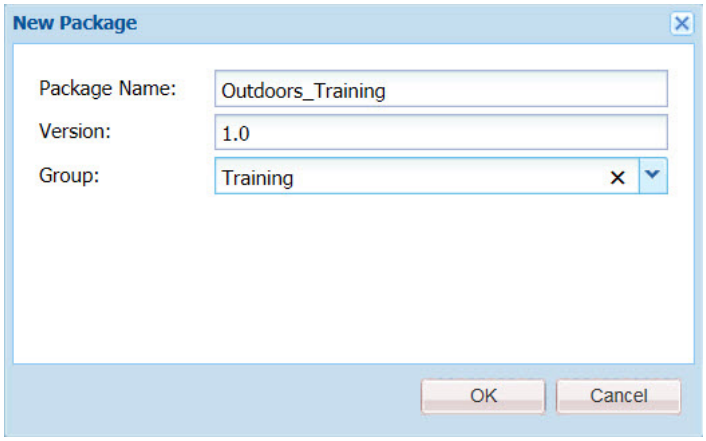
In this exercise, you will create a package of the Geometrixx project. The purpose of this exercise is to learn how to create and build packages using CRXDE Lite.

1. Navigate to the Projects console or click the following link: http://localhost:4502
2. In the left rail, select **Tools** > **Operations** > **Packaging** > **Packages**. This opens the Package Manager of CRXDE Lite.
3. Click **Create Package**.

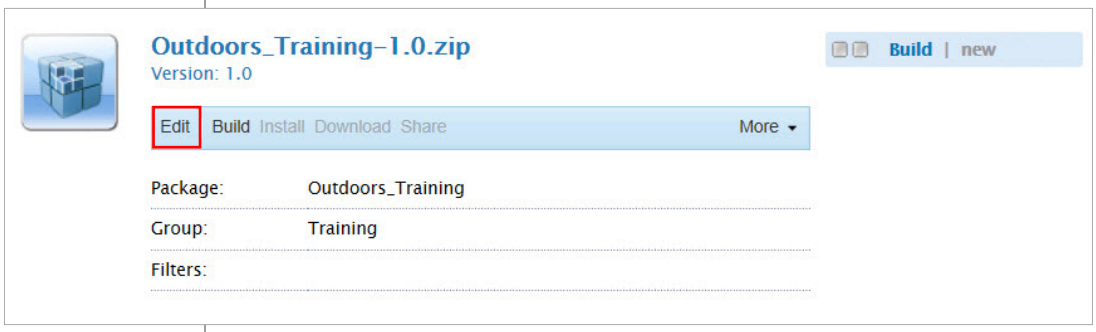


4. In the **New Package** dialog box, enter the following details:

Property	Value
Package Name	Outdoors_Training
Version	1.0
Group	Training

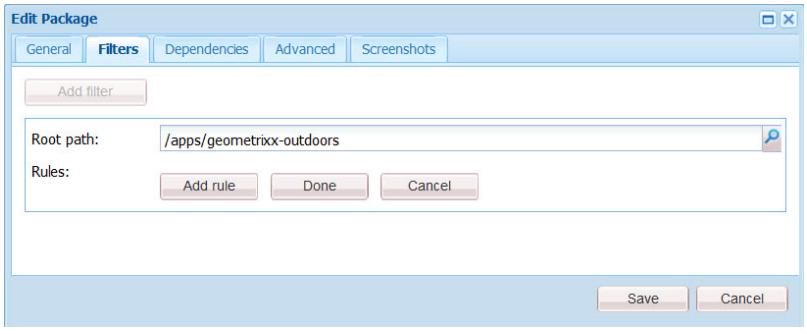


5. Click **Edit** on the newly created package.

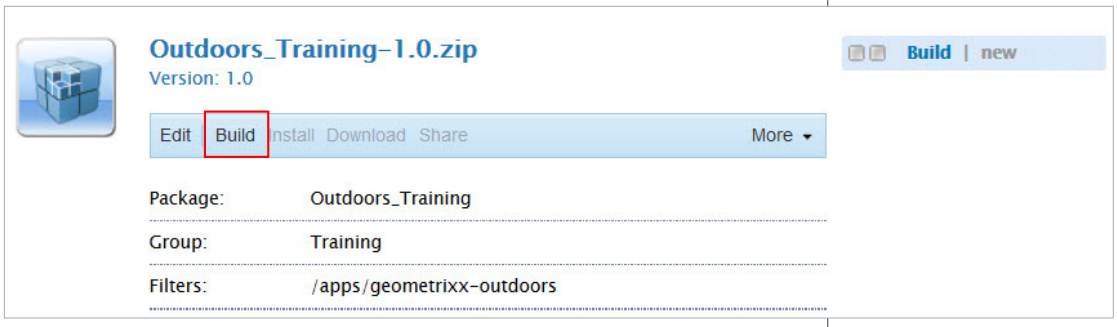


6. You need to add filters to the package. In the **Edit Package** dialog box, select the **Filters** tab, and click **Add Filter**.

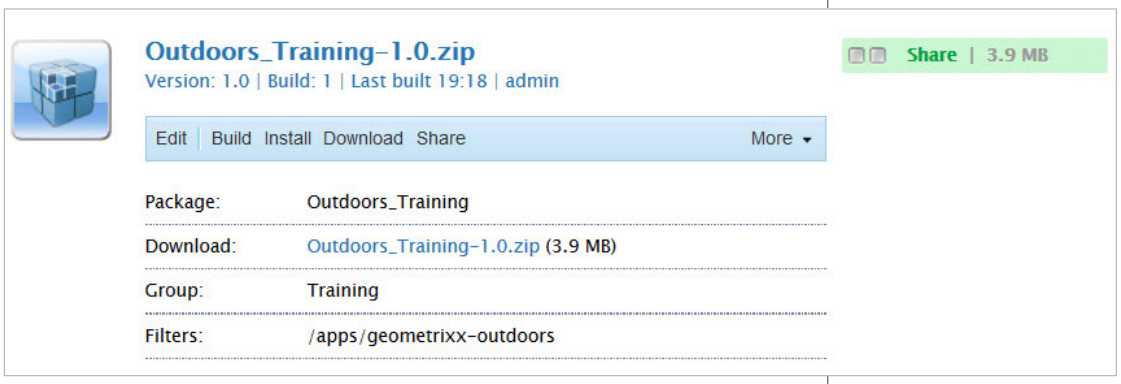
7. For the `Root path`, browse and select the `geometrixx-outdoors` project under `apps`, and click **Done**, and then click **Save**.



8. Select **Build** to build the package, and then click **Build** in the confirmation dialog box.



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

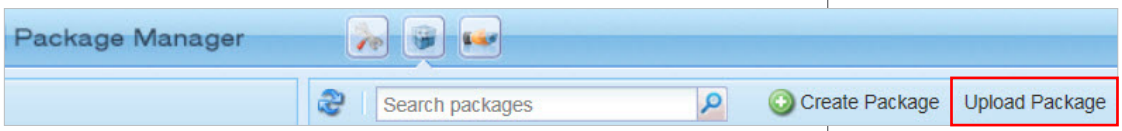


Exercise 5.2
Upload and install a sample package

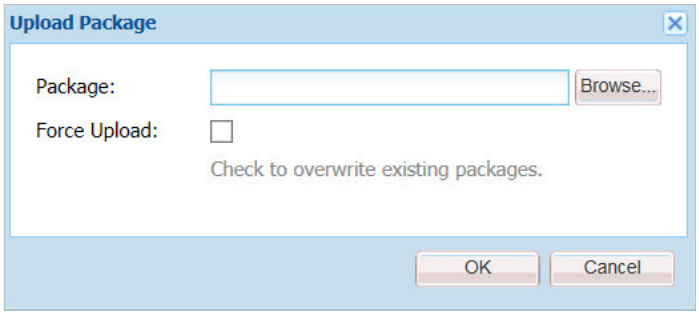
The following instructions explain how to upload a package using the Package Manager in CRXDE Lite.

1. Navigate to the **Projects** console, or click the following link: http://localhost:4502
2. In the left rail, select **Tools** > **Operations** > **Packaging** > **Packages**. This opens the Package Manager of CRXDE Lite.

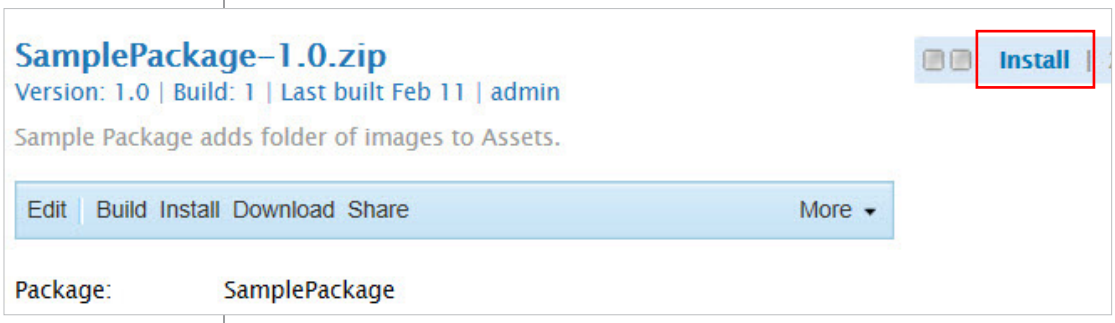
   As an alternative, click the following link to open the page directly: http://localhost:4502/crx/packmgr/index.jsp
3. Click **Upload Package**.



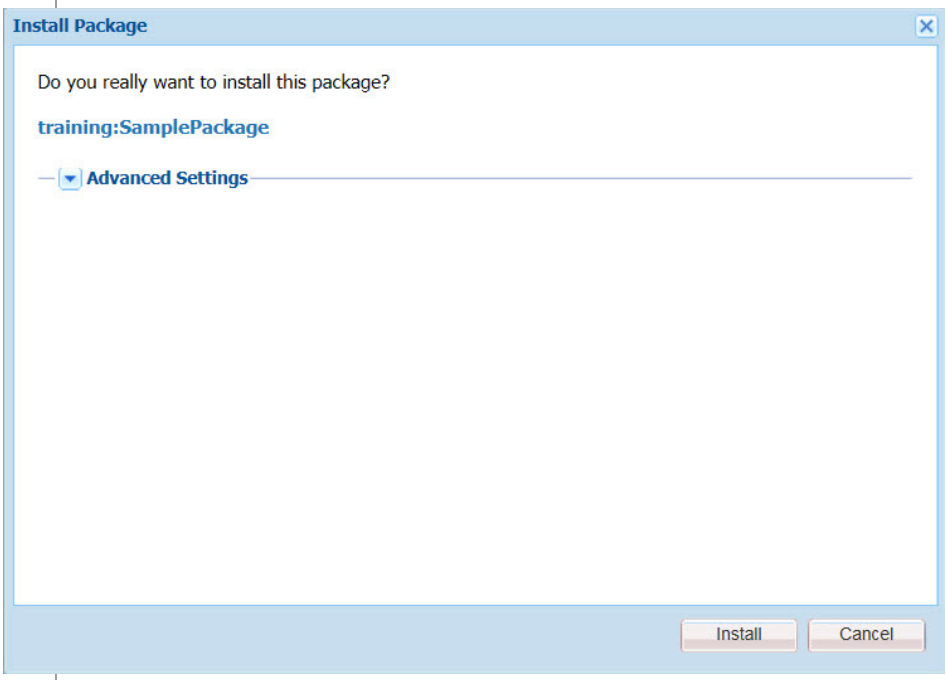
4. In the **Upload Package** dialog box, click **Browse** and select the **SamplePackage-1.0.zip** package from the exercise files provided to you. Click **OK**.



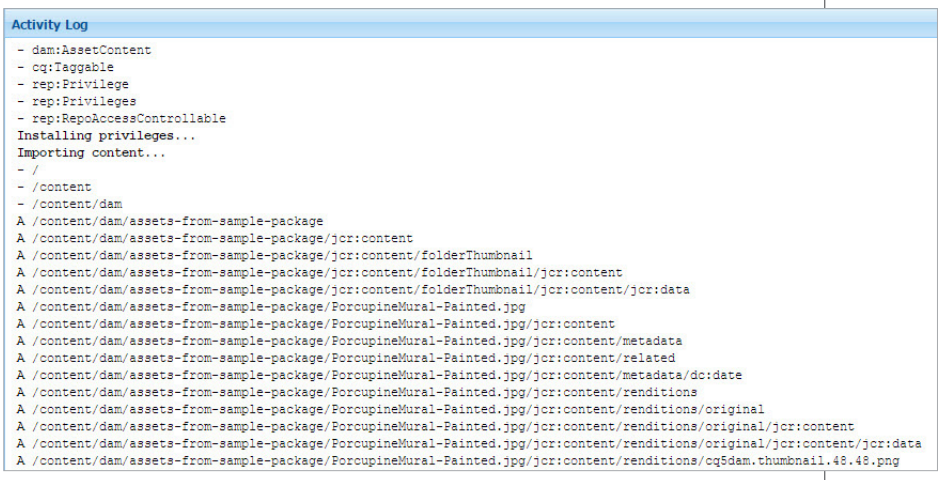
5. After the package is uploaded, click **Install**.



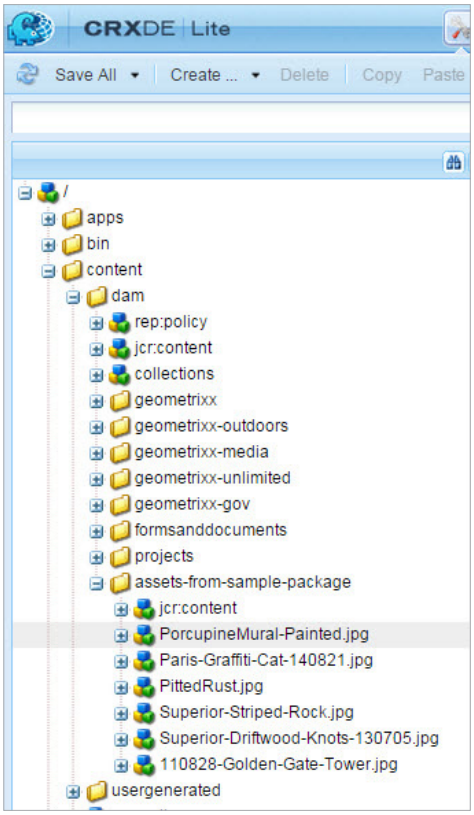
6. In the **Install Package** dialog box, click **Install**.



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



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



Brackets Sightly Extension

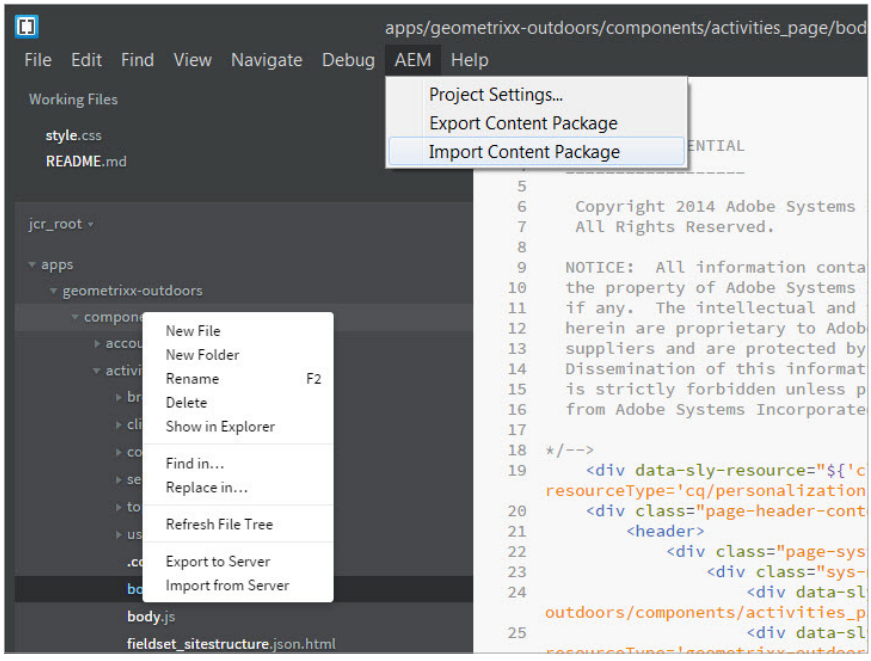
Brackets is a free, open-source editor written in HTML, CSS, and JavaScript with a primary focus on Web Development. It was created by Adobe Systems, licensed under the MIT License, and is currently maintained on GitHub. Brackets is available for cross-platform download on Mac, Windows, and Linux at:
http://brackets.io/

Brackets features live preview, in-line editors, pre-processor support, and a wide variety of add-on extensions.

Sightly is the new HTML templating system introduced with AEM 6.0. The AEM Sightly Brackets Extension provides a smoother development workflow for front-end display code.

The Sightly Brackets Extension can be installed directly from inside Brackets and features:

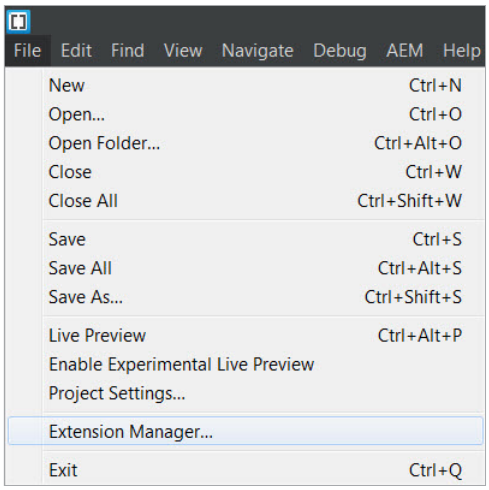
- Sightly syntax-highlighting
- Sightly code-completion
- Automatic, bidirectional synchronization with the JCR repository



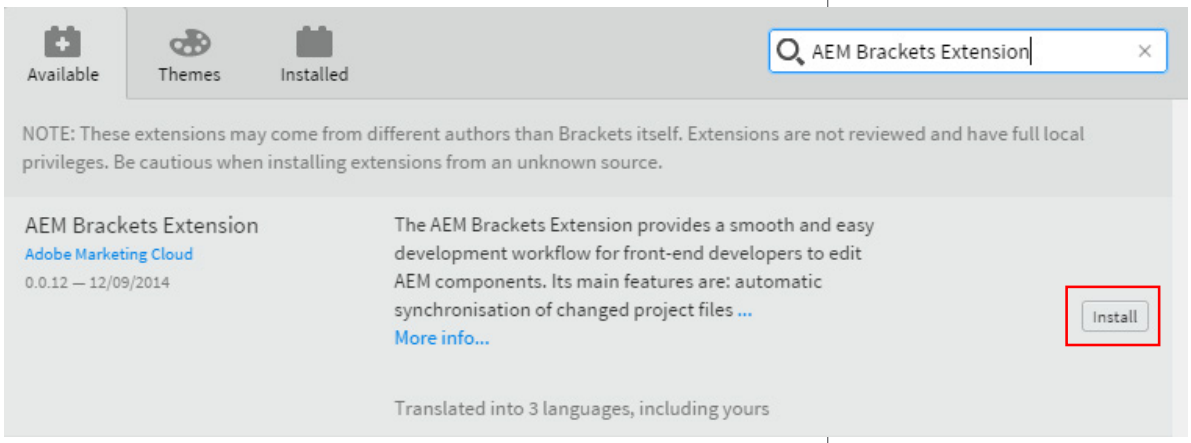
Brackets with the AEM Sightly Extension is the preferred tool for front-end developers building components with Sightly.

Exercise 5.3
Install the Brackets Plugin

1. Download the latest version of Brackets from http://brackets.io/
   The installation files are provided in the USB.
2. Install Brackets and open it.
3. Click **File** > **Extension Manager...**.



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



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

AEM Developer Tools for Eclipse

Eclipse is an Integrated Development Environment (IDE) popular with Java developers. It contains a base workspace and an extensible plugin system for customizing the environment.

The AEM Developer Tools for Eclipse is an Eclipse plugin based on the Eclipse plugin for Apache Sling released under the Apache License 2.

It offers several features that make AEM development easier:

- Seamless integration with AEM
- Synchronization with both content and OSGI bundles
- Debugging support with code hot-swapping capability
- Simplified creation of AEM projects
- Easy access to JCR properties

The AEM developer tools for Eclipse is generally preferred by developers working with JSP display code and building Java-based extensions to AEM in OSGi bundles.

Currently, the developer tools support AEM 5.6.1, 6.0, 6.1, and can be used with the Kepler and Luna versions of Eclipse. Documentation for the AEM developer tools for Eclipse is available at https://docs.adobe.com/docs/en/dev-tools/aem-eclipse.html. The developer tools can be downloaded from https://eclipse.adobe.com/aem/dev-tools/

Scripting Languages

In the front-end of an AEM website, rendering code is contained in components in the form of scripts. Sling defines components and allows scripts to be written in JavaScript, JSP, Ruby, Velocity, and Scala. Sling also allows additional scripting languages to be added.

In versions of AEM before 6.0, Java and JSP were the primary scripting languages used in AEM. With version 6.0, a new scripting language was introduced called Sightly. Sightly takes the place of JSP as the preferred scripting language for component scripts.

To ease migration, JSP and Sightly scripts can be used together. To provide ongoing support for different development models, both JSP and Sightly will continue to be supported.

JSP

Java Server Pages (JSP) is a server-side scripting language that combines HTML with Java program code and JSP tag libraries to create dynamic web pages. JSP is powerful and the pages can be flexible but they can be complex to develop. As it requires Java and JSP syntax knowledge, creating JSP scripts usually requires Java developer-level skills.

As all of the back-end code for AEM is already written in Java, JSP can be a good fit for a development team working on the platform. Developing AEM scripts in JSP is enabled by several AEM and Sling tag libraries. The JSTLs (Java Server Pages Standard Tag Libraries) are also available.

More information on JSP development and the AEM Tag Libraries is available in the documentation at https://docs.adobe.com/docs/en/cq/5-6/howto/taglib.html.

Sightly

A Sightly 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.

Sightly differs from other templating systems in three main ways:

- Sightly is HTML5: A template created in Sightly is a valid HTML5 file. All Sightly-specific syntax is expressed within a data attribute or within HTML text.

Any Sightly 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.

- Separation of concerns: The expressiveness of the Sightly 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 Sightly's `Use` API defines the structure of the external helper.
- Secure by default: Sightly automatically filters and escapes all text being output to the presentation layer to prevent cross-site-scripting vulnerabilities.

Moving to Sightly

Components written in Sightly are compatible with components written in JSP or ESP. Templates created in Sightly can also be used alongside JSPs and ESPs, even within the same component. For example, a JSP can include a Sightly script like this:

```
<cq:include script="footer.html"/>
```

A Sightly template can include a JSP script like this:

```
<div data-sly-include="footer.jsp"/>
```

Note: All components developed using JSP will work without any changes. Adobe recommends you use Sightly to develop new components.

Markup

Every Sightly template is a valid HTML5 document or fragment, augmented with specific syntax that supports Sightly functionality.
For example:

```
<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>
```

As you can see, the above snippet is valid HTML5 document. Additionaly, it also includes:

- Sightly expression language: Sightly 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.

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

Sightly Syntax

For details, visit:
http://docs.adobe.com/content/docs/en/aem/6-0/develop/sightly.html.

Comments

Sightly comments are HTML comments with additional syntax. They are delimited like this:

```
<!--/* A Sightly Comment */-->
```

Expressions

Sightly expressions are used to access the data structures that provide the dynamic elements of the HTML output. A Sightly expression is delimited by ${ and }. The expression syntax includes literals, variables, operators, and options.

Literals

Boolean:

```
${true} ${false}
```

Integers (including exponentiation). Floating point numbers are not supported:

```
${42} ${42e2}
```

Strings:

```
${'foo'} ${"bar"}
```

Variables

Variables are accessed as below:

```
${properties.text}
```

Enumerable Objects

These objects provide convenient access to commonly used information. They can be iterated using data-sly-list.

- `properties`: List of properties of the current resource. Backed by org.apache.sling.api.resource.ValueMap.
- `pageProperties`: List of page properties of the current page. Backed by org.apache.sling.api.resource.ValueMap.
- `inheritedPageProperties`: List of inherited page properties of the current page. Backed by org.apache.sling.api.resource.ValueMap.

Java-backed Objects

These objects provide the standard AEM execution context (as global.jsp does for JSPs, for example). Each object is backed by the corresponding Java object.

For example:

```
component backed by com.day.cq.wcm.api.components.Component.
```

URI Manipulation

The URI manipulation options work for expressions that are outside of block statements as well as for data-sly-text and data-sly-attribute. URI manipulation can be performed by adding any of the following options to an expression:

Option	Description
scheme	Allows adding or removing the scheme part of the URI. Example: ${'example.com/path/page.html' @ scheme='http'} Output: http://example.com/path/page.html
domain	Allows adding or replacing the host and port (domain) for a URI Example: ${'http://www.example.com/path/page.html' @ domain='www.example.org'} Output: http://www.example.org/path/page.html
path	Modifies the path that identifies the resource. Example, ${'http://example.com/this/one.selector.html/suffix?key=value#fragment' @ path='that/two'} Output: http://example.com/that/two.selector.html/suffix?key=value#fragment
selectors	Modifies or removes the selectors from a URI. Example: ${'path/page.woo.foo.html' @ addSelectors='foo.bar'} Output: path/page.woo.foo.bar.html
extension	Adds, modifies, or removes the extension from a URI. Example, ${'path/page.json' @ extension='html'} Output: path/page.html
suffix	Adds, modifies, or removes the suffix part from a URI. Example, ${'path/page.html/some/suffix' @ suffix='my/suffix'} Output: path/page.html/my/suffix
fragment	Adds, modifies, or replaces the fragment segment of a URI. Example, ${'path/page#one' @ fragment='two'} Output: path/page#two

NOTE: The Selectors are the URI segments between the part that identifies a resource and the extension used for representing the resource.

Sightly Block statements

Following are the Sightly block statements. For details, visit: http://docs.adobe.com/content/docs/en/aem/6-0/develop/sightly.html.

use

`data-sly-use`: Initializes a helper object (defined in JavaScript or Java) and exposes it through a variable. Initialize a JavaScript object, where the source file is located in the same directory as the template. Note that the filename must be used:

```
<div data-sly-use.nav="navigation.js">${nav.foo}</div>
```

Initialize a Java class, where the source file is located in the same directory as the template. Note that the classname must be used, not the file name:

```
<div data-sly-use.nav="Navigation">${nav.foo}</div>
```

unwrap

`data-sly-unwrap`: Removes the host element from the generated markup while retaining its content. This allows the exclusion of elements that are required as part of Sightly presentation logic but are not desired in the actual output.

However, this statement should be used sparingly. In general, it is better to keep the Sightly markup as close as possible to the intended output markup. In other words, when adding Sightly block statements, try as much as possible to annotate the existing HTML, without introducing new elements.

text

`data-sly-text`: Replaces the content of its host element with the specified text

element

`data-sly-element`: Replaces the element name of the host element

test

`data-sly-test`: Conditionally removes the host element and its content. A value of false removes the element; a value of true retains the element.

list

`data-sly-list`: Repeats the content of the host element for each enumerable property in the provided object

Here is a simple loop:

```
<dl data-sly-list="${currentPage.listChildren}">
        <dt>index: ${itemList.index}</dt>
<dd>value: ${item.title}</dd></dl
```

resource

`data-sly-resource`: Includes the result of rendering the indicated resource through the sling resolution and rendering process

A simple resource includes:

```
<article data-sly-resource="path/to/resource"></article>
```

include

`data-sly-include`: Replaces the host element with the markup generated by the indicated HTML template file (Sightly, JSP, ESP, and so on) when it is processed by its corresponding template engine. The rendering context of the included file will not include the current Sightly context (that of the including file). Consequently, for inclusion of Sightly files, the current data-sly-use would have to be repeated in the

included file. In such a case, it is usually better to use data-sly-template and data-sly-call.

A simple include statement:

```
<section data-sly-include="path/to/template.html"></section>
```

repeat

`data-sly-repeat`: Iterates over the content of each item in the attribute value and displays the containing element as many times as items in the attribute value. That is, the data-sly-repeat attribute works similar to the data-sly-list, but instead of repeating the content of the element, it repeats the element itself.

A simple repeat statement:

```
<ul>
    <li data-sly-repeat="${currentPage.listChildren}">${item.
    title}</li>
</ul>
```

Special HTML Tags

<sly>

The `<sly>` HTML tag can be used to remove the current element, allowing only its children to be displayed. Its functionality is similar to that of `data-sly-unwrap`.

A simple example of `<sly>` tag:

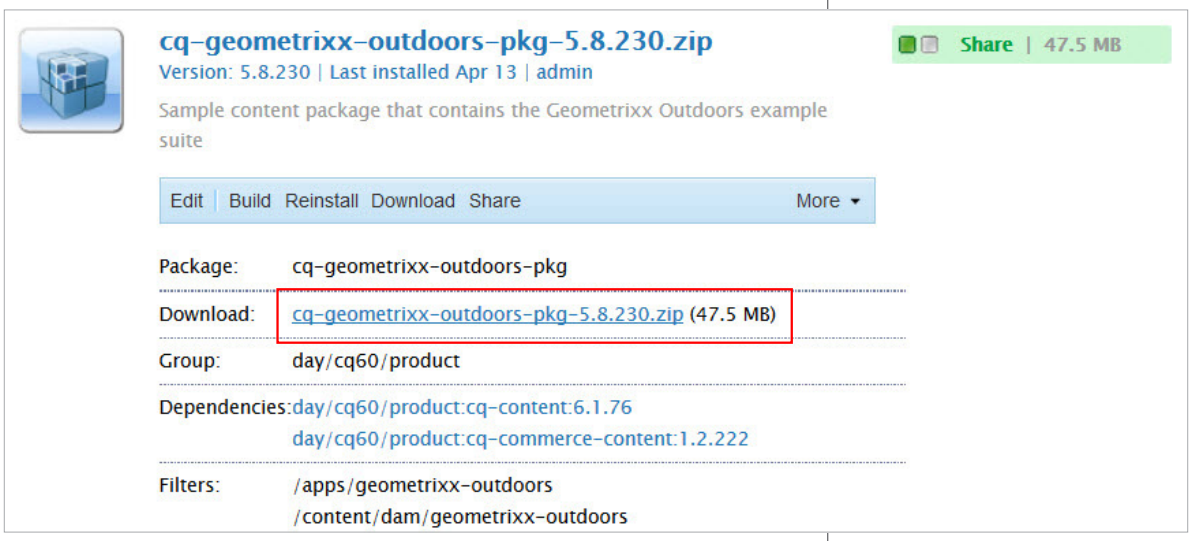
```
<sly data-sly-resource="node"/>
```

Exercise 5.4
Work on the Geometrixx project using Brackets plugin

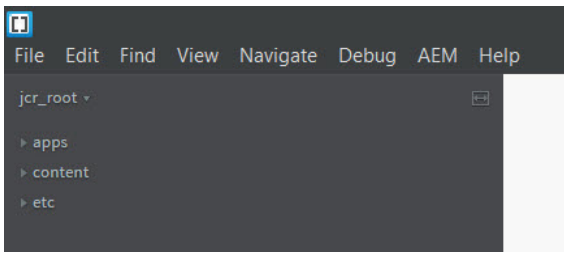
The goal of this exercise is to edit the contents of the Geometrixx Outdoors project through Brackets. To perform this exercise, you will first need to download a package from the Package Manager.

1. Open the **Package Manager** available in CRXDE Lite. http://localhost:4502/crx/packmgr/index.jsp
2. Search for the package `cq-geometrixx-outdoors-pkg` (for example, `cq-geometrixx-outdoors-pkg-5.8.160.zip`)
3. Select the package to expand it.

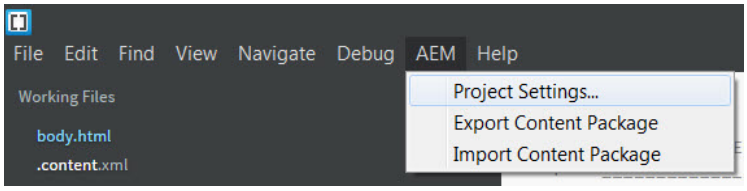
4. Click the link in **Download** to download the package to your local disk.



5. Unzip the file.
6. Open Brackets.
7. Navigate to **File** > **Open Folder...**.
8. 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.

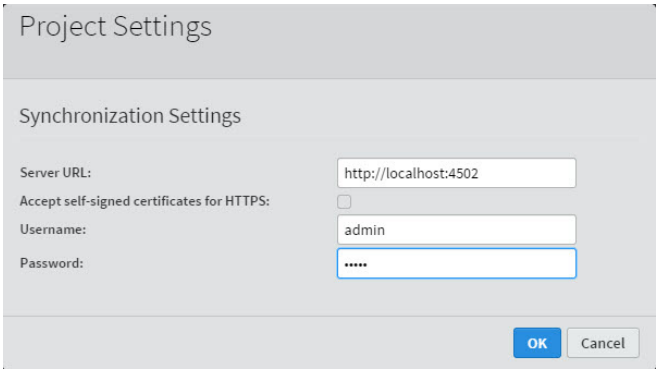


9. Configure the Project Settings. On the menu bar, click **AEM**, and select **Project Settings...**.



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

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



11. In the left panel, expand the `apps` node and navigate to `apps/geometrixx-outdoors/components/activities _ page/body.html`.

12. 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}
```

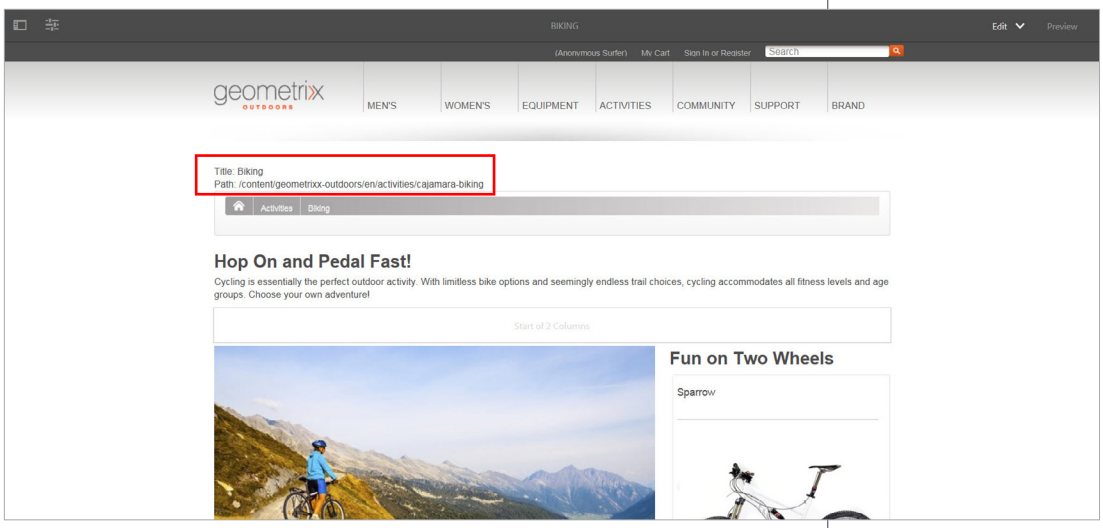
The inserted code is shown in the figure below.

```
22              <div class="page-systemnav">
23                  <div class="sys-nav-wrapper">
24                      <div data-sly-resource="${'search' @ resourceType='geometrixx-
    outdoors/components/activities_page/search'}" class="search"></div>
25                      <div data-sly-resource="${'userinfo' @
    resourceType='geometrixx-outdoors/components/activities_page/userinfo'}"
    class="userinfo"></div>
26                  </div>
27              </div>
28              <div data-sly-resource="${'topnav' @ resourceType='geometrixx-
    outdoors/components/activities_page/topnav'}" class="topnav"></div>
29          </header>
30      </div>
31      <div id="main" class="page-main">
32          <div class="page-content">
33          Title: ${currentPage.title}<br>
34          Path: ${currentPage.path}
35              <div data-sly-resource="${'breadcrumb' @ resourceType='geometrixx-
    outdoors/components/activities_page/breadcrumb'}" class="breadcrumb"></div>
36              <div data-sly-resource="${'par' @
    resourceType='wcm/foundation/components/parsys'}"></div>
37          </div>
38          <div class="page-footer" sly-use.footer="footer.js">
39              <footer>
40                  <nav>
41                      <div data-sly-resource="${'toolbar' @
    resourceType='foundation/components/toolbar'}" class="toolbar"></div>
42                  </nav>
```

13. Save your changes. This would automatically synchronize the file with AEM. 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.



AEM Authoring Framework — Templates

Overview

In this chapter, you will begin the project of creating your own website. To create your own website, you need to understand the layout of your site and the elements needed on each page.

Objective

In this chapter, you will create a project using the authoring framework in AEM. You will specifically learn how to do the following:

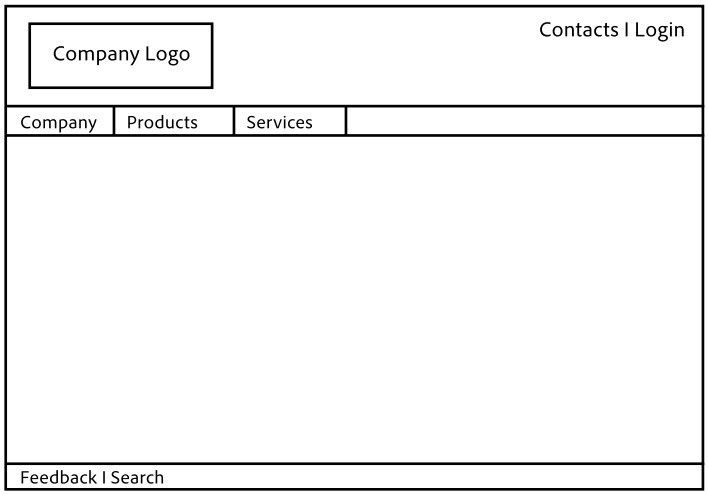
- Create an appropriate directory structure
- Create a template and a rendering component
- Display basic page information

Creating Your Website

In this section, you will create a website in AEM. Typically, you will create a new website in AEM in the following scenarios:

- There is an existing website. You want to create it in AEM.
- You do not have a website; you are developing one from the beginning. You may have a prototype or a wireframe that helps you to understand the different elements needed in the website.

You need to understand the elements needed in your website. The following is a prototype of a website that you will create.



Every page in the site should have the following elements:

- A company logo. The company logo does not change from page to page. You do not want the author to set the company logo.
- A contacts link: This should take users to a page that has the contact information.
- A login link: This should take users to a page where they can log in.
- It should have a navigation bar as shown above. It displays three links: Company, Products, and Services. These links are essentially the pages that you have in the root folder of your website for a specific region.
- A feedback link: This should take users to the feedback page.

When starting with multiple wireframes, you can use the following process to determine the templates and components that will make up the pages.

- Examine all pages and begin grouping them by structure similarities. Ignore the content at this point and look only at structure.
- Sketch or whiteboard similarities and differences.
- Identify the unique templates.
- Catalog the required components.
- Identify inheritance between the foundation SuperTypes, local SuperTypes, and unique template structure.
- Map the required components to the out-of-the-box components.

Identify and separate header, body, and footer sections of the pages. The structure of the page would be:

- Header
- Content
- Footer

After you have identified the templates, you can begin to deconstruct the pages by identifying the individual structure and components that will make up the pages. This decomposition of the major structural pieces into components will help you to identify which items can be reused and which items are unique to the template. For example, most of the time, the pages will use the same or a similar header and footer. This might mean that the header and footer can be reused. The page can have the following structural elements:

Header

- Logo
- Search box
- Navigation menu

Body

- Subsection text and image
- Text paragraph
- Image
- Forms

Footer

- Copyright notice
- Image

Now, you have identified the structure. In the following sections, you will develop the page.

TIP: Remember that having as few templates as possible is good practice.

Structure Your Application

When you develop a website, as a first step, create a structure to store various elements of your application/project. These elements include your templates, components, OSGi bundles, and static files. Typically, you create the directory structure inside the */apps* folder.

Adobe recommends the following directory structure for your projects:

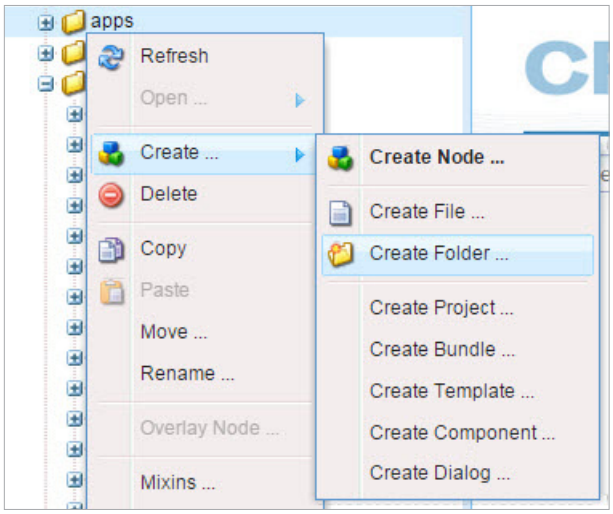
Structure	Description
/apps/*application-name*	The website's main folder
/apps/*application-name*/components	The folder to hold components
/apps/*application-name*/templates	The folder to hold templates

NOTE: Organize your website in a way that allows you to maintain it easily.

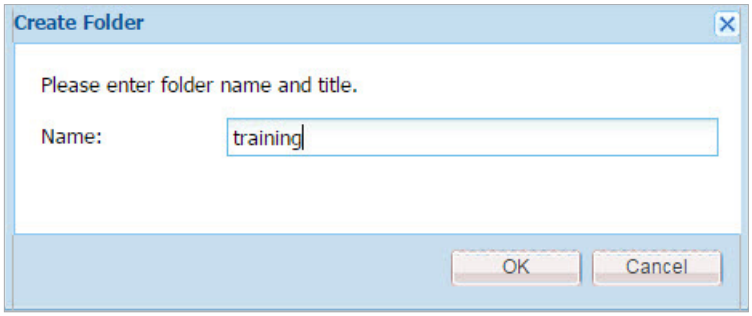
Exercise 6.1
Create the Structure of Your Website

In the following exercise, you will create a directory structure for the website that you will develop.

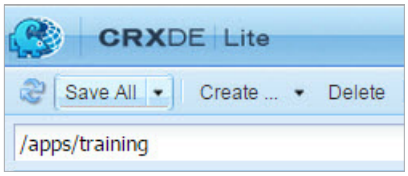
1. Log in to CRXDE Lite. In the left pane, navigate to the `apps` folder.
2. Right-click the `apps` folder, select **Create**, and click **Create Folder...**.



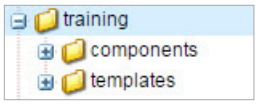
3. Enter the name as `training`, and click **OK**. As a best practice, provide the node names in lowercase because they become part of the URL.



4. From the top bar, click **Save All**.



5. Repeat the same process to create a directory structure as follows:



You have created the basic structure of your website.

Create Templates

A template is used to create a page and defines which components can be used within the selected scope. A template is a hierarchy of nodes that has the same structure as the page to be created, but without any actual content. Each template will present you with a selection of components available for use. Templates are built up of components. Components use and allow access to widgets, which are used to author or render the content.

A template is the basis of a page. To create a page, the template's content must be copied (*apps/<application name>/templates/<template name>*) to the corresponding position in the site tree. This occurs automatically if the page is created using AEM. This copy action 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.

When you create a template, the following information is saved in the repository. The template creation widget provides you with options to enter the below details:

- Label: cq:Template node name
- Title: jcr:title property

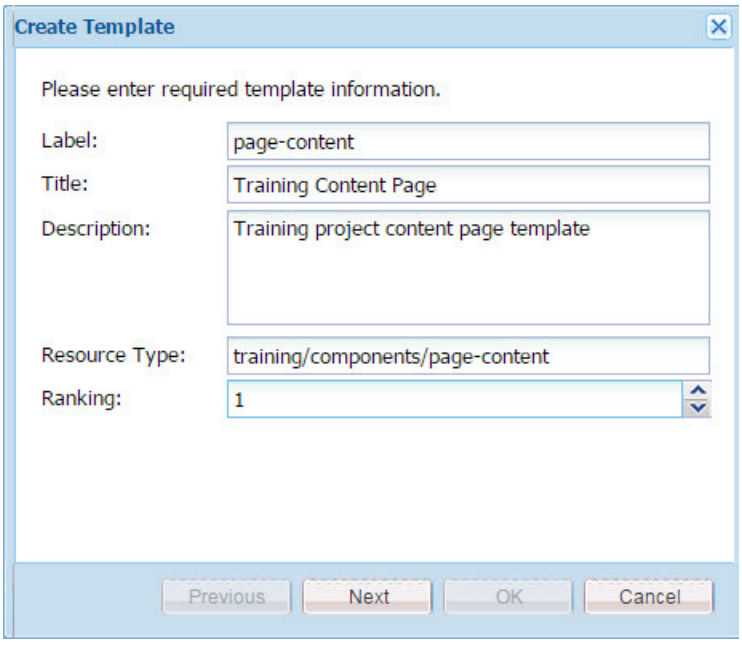
- Resource Type: sling:resourceType property
- Ranking: ranking property
- Allowed Paths: allowedPaths property

Exercise 6.2
Create a Template for Your Website

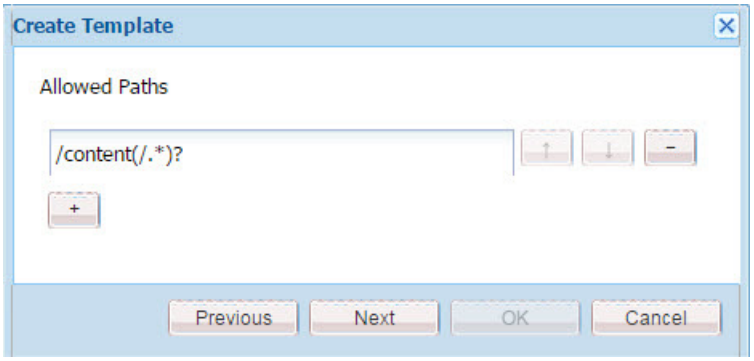
In the following exercise, you will create a template.

1. Right-click the `templates` folder created in the previous exercise.
2. Click **Create** > **Create Template**.
3. Enter the following details, and click **Next**:

Name	Value
Label	page-content
Title	Training Content Page
Description	Training project content page template
Resource Type	training/components/page-content
Ranking	1 (ranking indicates the order in which the template appears on the page creation page. Setting the rank to 1 ensures that the template appears first in the list



4. Click the + symbol provided with the **Allowed Paths** property. The **Allowed Paths** property defines the path where this template is to be used to create pages. Add the following value: `/content(/.*)?`

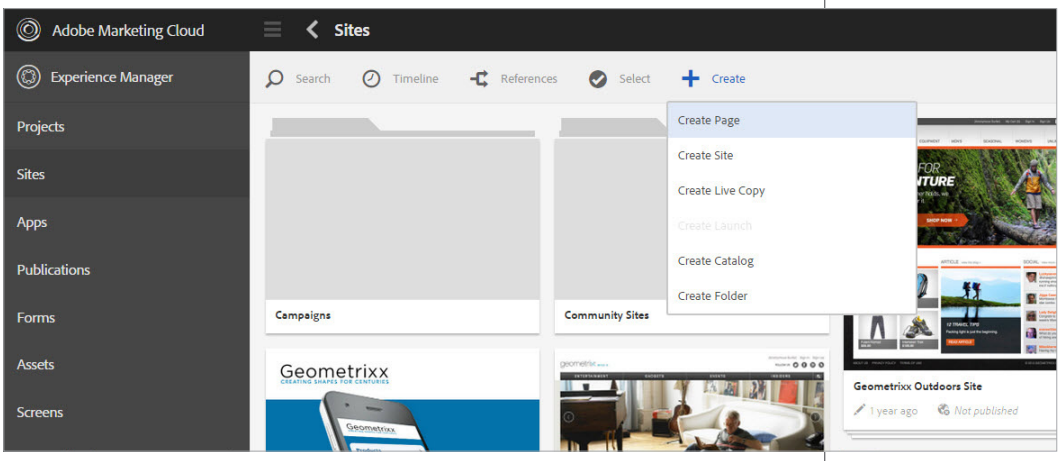


5. Click **Next** on the Allowed Parents screen.
6. Click **OK** on the Allowed Children screen.
7. Click **Save All** from the upper-left bar to save the changes.

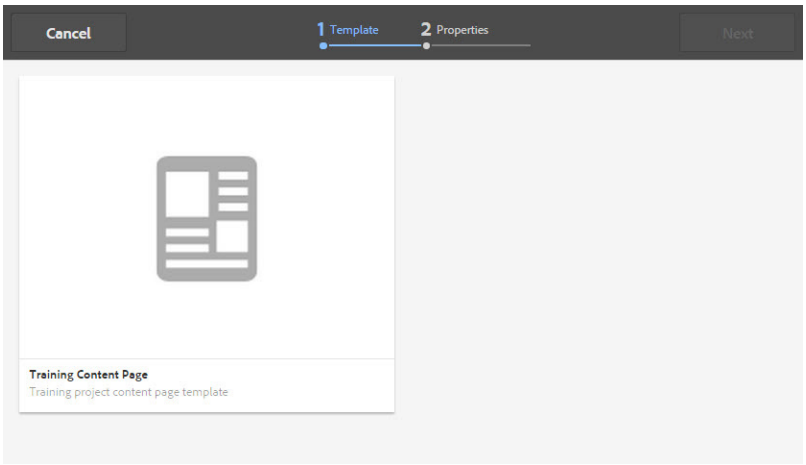
Testing the Template

In the previous exercise, you created a template; however, you have not yet created a component to render the page. Therefore, this template does not render any content. You can test if the template was created successfully by following these steps:

1. Go to **Site** console, or click the following link: http://localhost:4502/sites.html/content
2. On the toolbar, click **Create**, and select **Create Page**.



3. See the template you created—as highlighted in the image below. Note that it does not have a thumbnail. Observe the description you provided while creating the template. Also, note that it appears as the first template in the list because you specified the rank as 1.



4. Click **Cancel** to close the window.

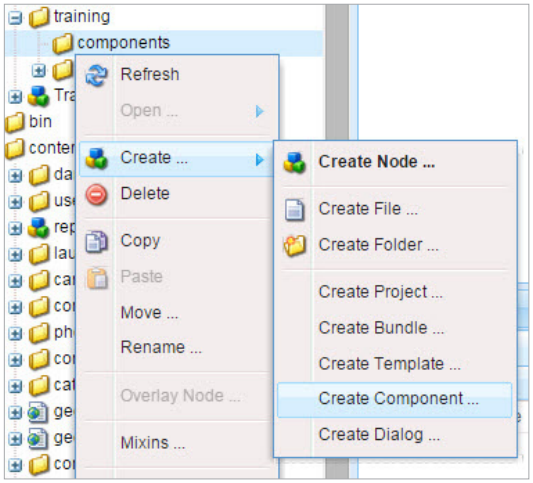
Create a Page-Rendering Component

Components are modular, reusable units that implement specific functionality or logic to render the content of your website. They have no hidden configuration files, can include other components, and can run anywhere within AEM or in isolation (for example, a portal). A component can be described as a collection of scripts (for example, Sightly files, JSPs, Java servlets, and so on) that completely realize a specific function. More specifically, a template typically references a 'page' component.

The **Create Component** wizard allows you to enter the information necessary to create the complete component structure. Typically, a component ('page' or otherwise) will have at least one default script, identical to the name of the component (for example, *contentpage.html* or *contentpage.jsp*).

Exercise 6.3
Create a Page-Rendering Component

1. Right-click the `/apps/training/components` folder, and click **Create** > **Create Component**.



2. Enter the following details:

Name	Value
Label	page-content
Title	Training Content Page
Description	The Training Content Page Component

Note that the '*page*' in the title component name is used to identify that it is associated with a template. In other words, it is a page component. You can also create a page folder and create the component inside. If you create a page folder, change the Resource Path accordingly in Exercise 6.2.

3. Click **Next** until you reach the last screen, and then click **OK**. Save the changes. Notice that the page-content component is created with a `page-content.jsp` script.

4. Open the script by double-clicking it. The script appears with some sample code that you can delete for now. Enter the following HTML code, and then click **Save All**.

```
<html>
        <head>
            <title>Hello World!!</title>
        </head>
        <body>
            <h1>Hello world!!</h1>
            <h2>I am your rendering script!!</h2>
        </body>
</html>
```

You have successfully created a page-rendering component.

Create Pages

A page is where content authors create and edit content that most likely will be published and viewed by site visitors. It is an exact copy of the template from which it was created.

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

Exercise 6.4
Create a Website Structure

In this exercise, you will create the basic structure of your website.

1. Log in to AEM.
   http://localhost:4502/siteadmin
2. Click **Websites** in the left pane.
3. Click **New** > **New Page**...
4. Enter the title of the page as `Training Site`.
5. Select the **Training Content Page** as the template, and click **Create**.
6. Double-click the page to open it. The page does not appear in the touch-optimized UI.
7. Move to the Classic UI by replacing `editor.html` with `cf#` in the URL as shown here:
   http://localhost:4502/cf#/content/training-site.html
   The page appears as follows:

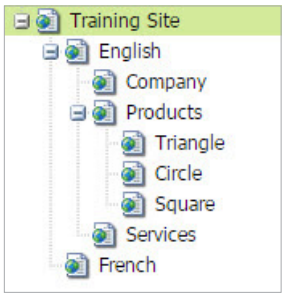
   Note that the page displays the texts added in the default script of the component.
8. Go to the Site Admin again, and select the Training Site page in the left panel.
9. Click **New** > **New Page**. Enter the details as follows:

Name	Value
Title	English
Name	en

10. Create one more page for the French locale, with the following details:

Name	Value
Title	French
Name	fr

11. Select the English page, and create the following pages underneath:
    a. Company
    b. Products
    c. Services
12. Select the Products page and create the following pages underneath:
    a. Triangle
    b. Circle
    c. Square
13. Ensure that you create the website structure as follows:



You have created the pages of your website.

Modify Page-Rendering Scripts

In the previous example, you noticed that when a page-rendering component was created, a rendering script was also created by default. When you view a page, the output is displayed from the rendering script.

By default, the rendering script is created in JSP language. The file name of the rendering script is the name of the component with a *.jsp* extension. AEM 6.0 introduces a new language for creating rendering scripts—Sightly. Sightly is a markup language that enables you to separate the logic and content. Adobe recommends using Sightly for front-end web developers to build AEM components.

You can use JSP or Sightly for developing the rendering scripts. In this course, you will use Sightly as the rendering scripts to develop the Training website. The course will also provide you with additional information that you need if you are developing rendering scripts using JSP.

Exercise 6.5
Modify the Page-Rendering Script to Use a Sightly Script

In this exercise, you will modify the page-rendering script. You will use a script that is developed in Sightly to display the content.

1. In CRXDE Lite, navigate to the `apps/training/components` folder.
2. Delete the `page-content.jsp` file.
3. Right-click the `page-content` node, and click **Create** > **Create File**.
4. Type the name of the file as `page-content.html`.

5. Double-click `page-content.html` if it is not already opened in the right pane.
6. Type the following code in the editor:

```
<html>
        <head>
            <title>Hello World!!</title>
        </head>
    <body>
        <h1>Hello World!!</h1>
        <h2>I am your Sightly rendering script!!</h2>
    </body>
</html>
```

7. Click **Save All**.
8. In the Site Admin, open the newly created Training Site page by double-clicking it. Move to the Classic UI by replacing the `editor.html` with the `cf#` tag in the URL.
   http://localhost:4502/cf#/content/training-site/en.html

Note that the page is updated with the new text.

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. As discussed previously, 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.

You can access the content in AEM in three ways:

- 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:
    Sightly
    `${currentPage.Title}`
    JSP
    `String pageTitle = currentPage.getTitle();`
- The properties object
  - The properties object is an instance of the ValueMap (see Sling API) class and contains all properties of the current resource.

- Sightly

  ```
  <p> Title : ${currentPage.properties.jcr:title}</p>
  ```

- JSP

  ```
  String pageTitle = properties.get("jcr:title", "NO
  TITLE");
  ```

- 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.
  - Sightly

    ```
    ${currentNode.Name}
    ```

  - JSP

    ```
    String pageTitle = currentNode.getProperty("jcr:title").
    getString();
    ```

Exercise 6.6
Display Basic Page Content Using Available APIs (in Sightly)

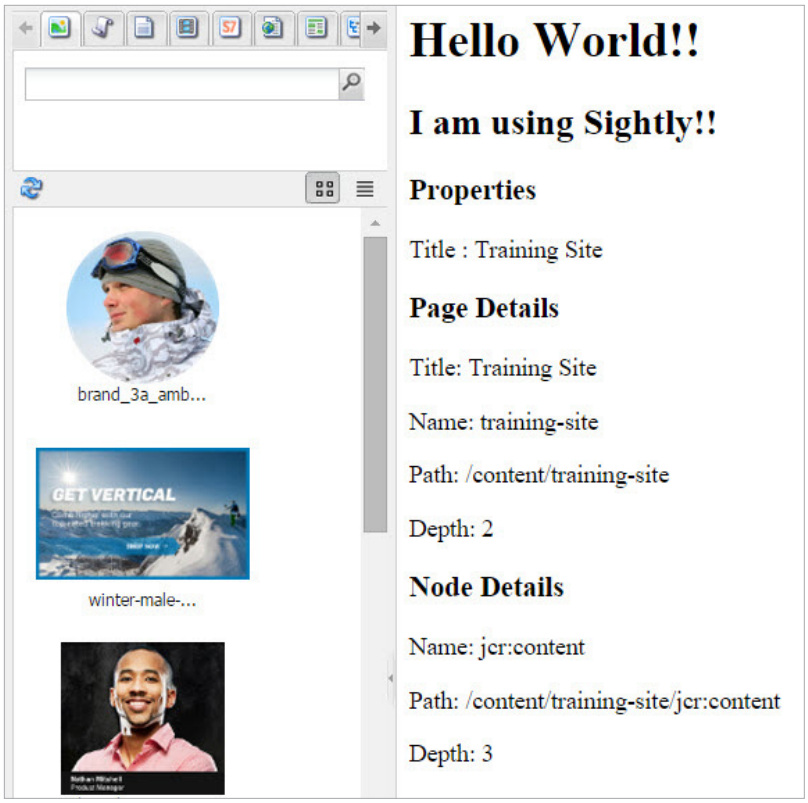
In this exercise, you will update the page-rendering script to use the existing APIs to display some of the page properties.

1. Navigate to `/apps/training/components`, and open `page-content.html` script.
2. Enter the following code, and then click **Save**.

```
<html>
    <head> <title>Hello World!!</title> </head>
    <body>
       <h1>Hello World!!</h1>
        <h2>I am using Sightly!!</h2>
       <h3>Properties</h3>
           <p> Title : ${currentPage.properties.jcr:title}</p>
       <h3>Page Details</h3>
           <p>Title: ${currentPage.Title}</p>
           <p>Name: ${currentPage.Name}</p>
           <p>Path: ${currentPage.Path}</p>
           <p>Depth: ${currentPage.Depth}</p>
       <h3>Node Details</h3>
           <p>Name: ${currentNode.Name}</p>
           <p>Path: ${currentNode.Path}</p>
           <p>Depth: ${currentNode.Depth}</p>
   </body>
  </html>
```

NOTE: To use the above APIs, in JSP, you need to include the following script: */libs/foundation/global.jsp*. However, including *global.jsp* is not required if you are using Sightly.

3. Test your script by opening one of the pages you created in Classic UI. (For example, English.)



Displaying Basic Page Content Using JSP

When you develop the JSP script of an AEM component, it is required to include the following code at the top of the script:

```
<%@include file="/libs/foundation/global.jsp"%>
```

The Adobe-provided *global.jsp* script declares the Sling, AEM, and JSTL taglibs and exposes the regularly used scripting objects defined by the *<cq:defineObjects/>* tag. This shortens and simplifies the JSP code of your component.

The *<cq:defineObjects/>* tag exposes the following, regularly used, scripting objects that can be referenced by the developer. It also exposes the objects defined by the *<sling:defineObjects>* tag.

The following objects are added by default in Sightly:

- componentContext
  - The current component context object of the request (*com.day.cq.wcm.api.components.ComponentContext* interface)
- component
  - The current AEM component object of the current resource (*com.day.cq.wcm.api.components.Component* interface)

- currentDesign
  - The current design object of the current page (*com.day.cq.wcm.api.designer.Design* interface)
- currentPage
  - The current AEM WCM page object (*com.day.cq.wcm.api.Page* interface)
- currentNode
  - The current JCR node object (*javax.jcr.Node* interface)
- currentStyle
  - The current style object of the current cell (*com.day.cq.wcm.api.designer.Style* interface)
- designer
  - The designer object used to access design information (*com.day.cq.wcm.api.designer.Designer* interface)
- editContext
  - The edit context object of the AEM component (*com.day.cq.wcm.api.components.EditContext* interface)
- pageManager
  - The page manager object for page-level operations (*com.day.cq.wcm.api.PageManager* interface)
- pageProperties
  - The page properties object of the current page (*org.apache.sling.api.resource.ValueMap*)
- properties
  - The properties object of the current resource (*org.apache.sling.api.resource.ValueMap*)
- resource
  - The current Sling resource object (*org.apache.sling.api.resource.Resource* interface)
- resourceDesign
  - The design object of the resource page (*com.day.cq.wcm.api.designer.Design* interface)
- resourcePage
  - The resource page object (*com.day.cq.wcm.api.Page* interface)

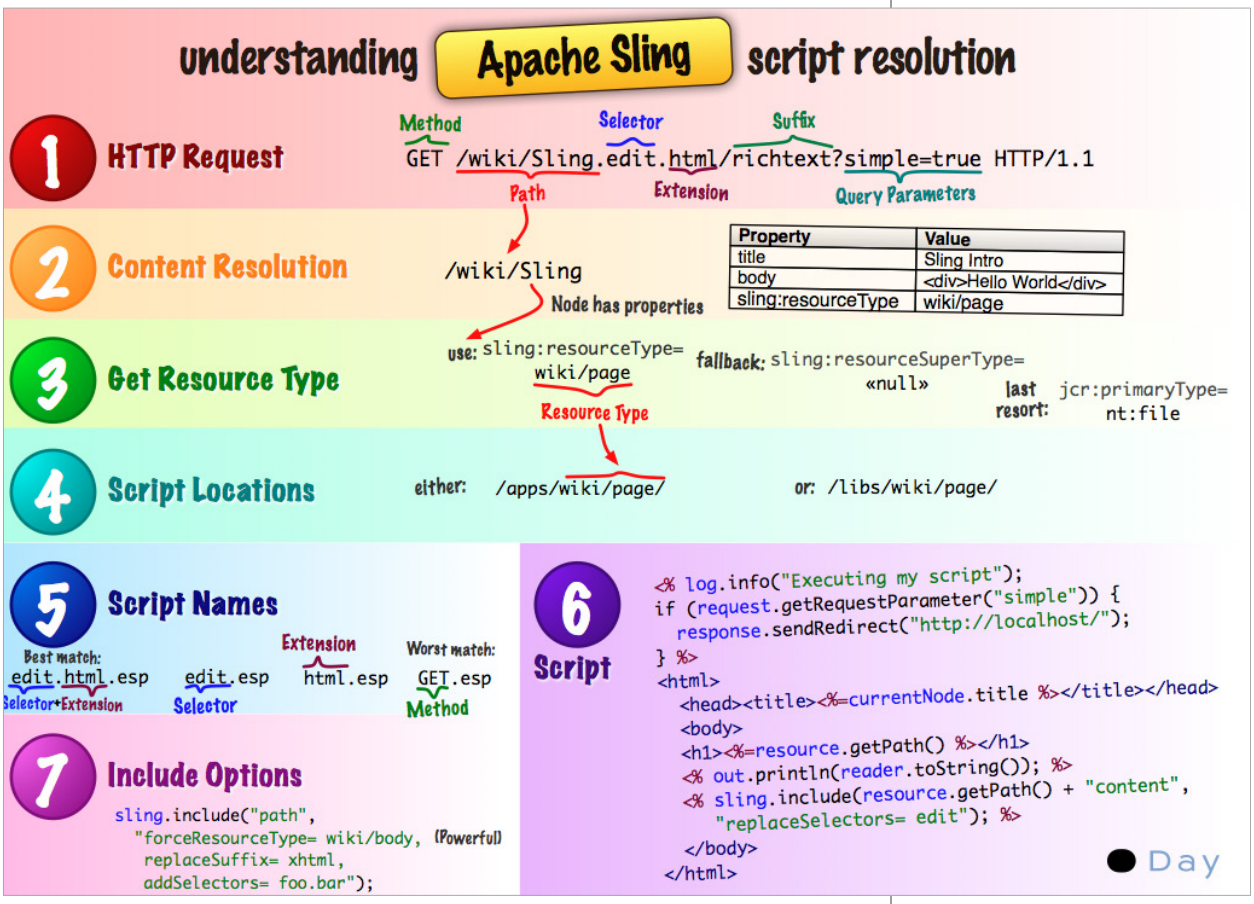
Extra Credit

Use the following code to test the same using a JSP script. Create `page-content.jsp`, and then insert the following code. Refresh the page you created.

```
<%--
      Training Content page component.
--%><%
    %><%@include file="/libs/foundation/global.jsp"%><%
    %><%@page session="false" %><%
    %>
    <html>
        <head>
            <title>
                <%= currentPage.getTitle() == null ? currentPage.getName()
    :currentPage.getTitle() %>
            </title>
        </head>
        <body>
            <h2>
                properties
            </h2>Title: <%= properties.get("jcr:title") %><br>
            <h2>
                currentPage
            </h2>Title: <%= currentPage.getTitle() %><br>
            Name: <%= currentPage.getName() %><br>
            Path: <%= currentPage.getPath() %><br>
            Depth: <%= currentPage.getDepth() %><br>
            <h2>
                currentNode
            </h2>Title: <%= currentNode.getProperty("jcr:title").getString()
    %><br>
            Name: <%= currentNode.getName() %><br>
            Path: <%= currentNode.getPath() %><br>
            Depth: <%= currentNode.getDepth() %><br>
            <br>
        </body>
    </html>
```

Recap of Sling framework

Now that you have created a template and webpages based on the template, let us take a quick recap of the Sling framework that you learned in the previous chapter. AEM is built using Apache Sling, a web application framework based on REST principles that provides easy development of content-oriented applications. The following image represents how script resolution happens in Sling.



Exercise 6.7
Create Multiple Scripts for the Page Component

In this exercise, you will create multiple scripts for the page component.

1. In CRXDE Lite, navigate to `/apps/training/components`, and locate the `page-content` node.
2. (If you created a JSP file) Ensure that you deleted the `page-content.jsp` script you created by right-clicking and clicking **Delete**.
3. Right-click the `page-content` node, and click **Create** > **Create File**.
4. Create a file named `html.html`, and save the changes.
5. Add the following code and save the changes:

```
<html>
        <head>
            <title>Hello World!!</title>
        </head>
        <body>
            <h1>This is the HTML script</h1>
        </body>
</html>
```

6. Create a file named `m.html`.
7. Add the following code, and save the changes:

```
<html>
        <head>
            <title>Hello World!!</title>
        </head>
        <body>
            <h1>This is the Mobile script</h1>
        </body>
</html>
```

8. Open one of the pages you created.
   http://localhost:4502/cf#/content/training-site/en/company.html
   Observe that HTML script is rendered.
9. Now, open the same page using the following URL:
   http://localhost:4502/cf#/content/training-site/en/company.m.html
   Note that the mobile script is rendered.
10. Delete the scripts (`html.html` and `m.html`) because we will not use these scripts to build the sample website.

AEM Authoring Framework — Components and Design

Overview

This chapter will introduce you to template modularization and guide you through the creation of components and designs. You will create a website using the authoring framework in AEM.

Objective

In this chapter, you will specifically learn how to do the following:

- Modularize the template
- Extend the component hierarchy
- Assign a design
- Create and include components in scripts

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 include scripts to support modularization in this manner. There are different ways in which you can include a file to the script. For example, the following code in JSP includes a file at compilation time: `<%@ include file="myScript.jsp" %>`

Adobe recommends that you include a file at runtime using the following methods:

- In Sightly, using the `<data-sly-include>` tag

  For example, `<div data-sly-include="myScript.html"/>`

- In JSP, using the `<cq:include>` tag

  For example, `<cq:include script="myScript.jsp">`

You can also use the `<sling:include>` tag to include a script at runtime. However, Adobe recommends that you use `<cq:include>`.

Exercise 7.1
Modularize the Page Component

In this exercise, you will modularize the page-rendering component. In the wireframe that you saw before, it makes sense to place the header and footer in two separate files. In that way, you can reuse the header or footer and overlay them in another template if needed.

- *body.html*: Holds the content that changes from page to page
- *header.html*: Holds the content that should appear in the header. It does not change from page to page.
- *footer.html*: Holds the content that should appear in the footer. It does not change from page to page.

As a best practice, split the template (script) only if you want to overlay that part later. Avoid creating more scripts by splitting the templates.

1. In CRXDE Lite, navigate to `/apps/training/components`, and locate the `page-content` component.
2. Open `page-content.html` and add the following code:
   Note that you use the `data-sly-include` tag to include an additional file in your template.

```
<div data-sly-include="body.html"></div>
```

3. Right-click the `page-content` component, and click **Create** > **Create File**. Create a file named `body.html`.

4. Open `body.html`, and add the following code:

```
<div data-sly-include=”header.html”></div>
    <div class=”body_content container_16”>
        <div class=”content_title grid_16”>Body content</div>
    </div>
<div data-sly-include=”footer.html”></div>
```

5. Create a new file, `header.html`, and add the following code:

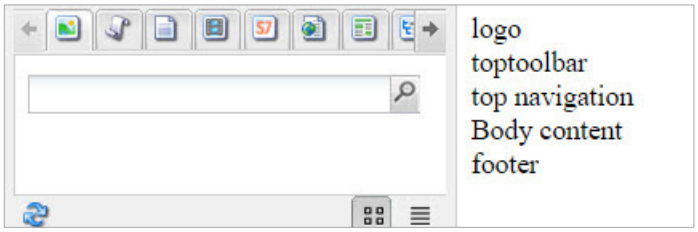
```
<div class=”body_header header container_16”>
    <div class=”header_logo grid_8”>logo</div>
        <div class=”header_topnav grid_8 search_area”>
            <div>toptoolbar</div>
        </div>
    <div class=”content_topnav toptoolbar toolbar”>top navigation</div>
</div>
```

6. Create a new file, `footer.html`, and add the following code:

```
<div class=”body_footer footer container_16”>
    <div class=”grid_6”>footer</div>
</div>
```

7. Click **Save All**.

8. Refresh the web page that you created:
   http://localhost:4502/cf#/content/training-site/en.html
   The page appears as shown below:



Example: Including additional files in JSP

```
<%@include file=”/libs/foundation/global.jsp”%>
<html>
  <head>
    <title><%= currentPage.getTitle() == null ? currentPage.getName():currentPage.getTitle()
    %></title>
    <cq:include script=”header.jsp”/>
  </head>
</html>
```

Inheriting Foundation Components

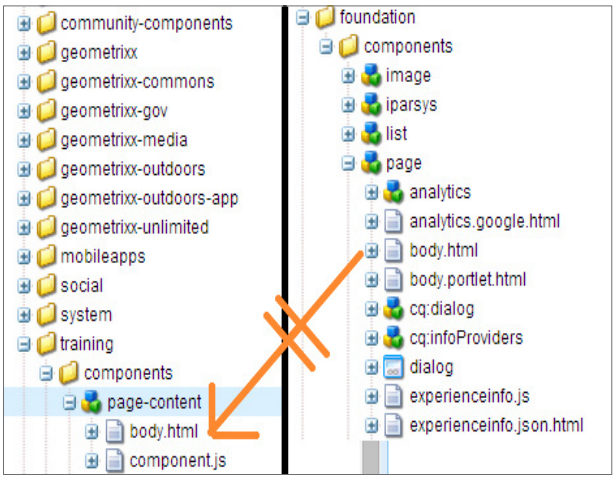
Components can be given a hierarchical structure to implement the inheritance of included script files, dialog boxes, and so on. Therefore, it is possible for a specific 'page' component (or any component) to inherit from a 'base' component. For example, allowing inheritance of a script file for a specific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

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.

Overlays

Sling's Resource Resolver searches for resources in a list of specific paths, as defined by the Apache Sling JCR Resource Resolver. The default search path is first `/apps` and then `/libs`. You can modify the 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.

Sling Resource Merger

Earlier in AEM, overlays enabled you to extend and customize components by copying the full JCR subtree under apps. That included the nodes as well as their properties. 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.

NOTE: The Sling Resource Merger can be used only with Granite. That is, with the touch-optimized UI.

The 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.

The Sling Resource Merger can be used to customize the AEM consoles, or to customize page authoring.

Overlays vs. Sling Resource Merger

The following table describes the basic difference between Overlays and Sling Resource Merger. However, it must be noted that both concepts are supported in AEM 6.1

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.

For more information on Overlays and Sling Resource Merger, refer to the online documentation here.

In the following example, you will learn how to extend the foundation page component. As mentioned earlier, by extending the component, you are extending some of the additional features that the component offers. You can find foundation components in two locations:

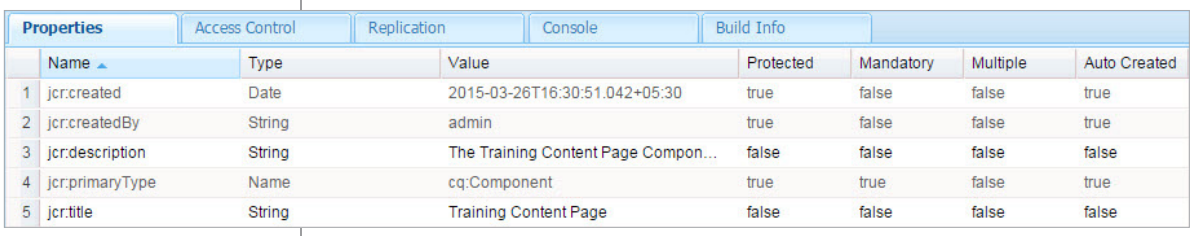
- *libs/foundation/components*: All components developed using JSP are available here.
- *libs/wcm/foundation/components*: All components developed using Sightly are available here.

You can select the component that you want to extend based on the templating language you use (Sightly/JSP). In the following exercise, you will extend a Sightly foundation page component.

Exercise 7.2
Inherit the Sightly Foundation Component Page

In this exercise, you will inherit the foundation component. This provides your page with additional functionalities.

1. In CRXDE Lite, select **training** > **components** > **page-content**. The properties of the component appear as shown below:

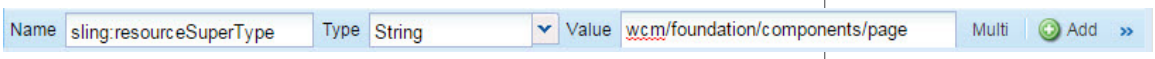


Properties | Access Control | Replication | Console | Build Info

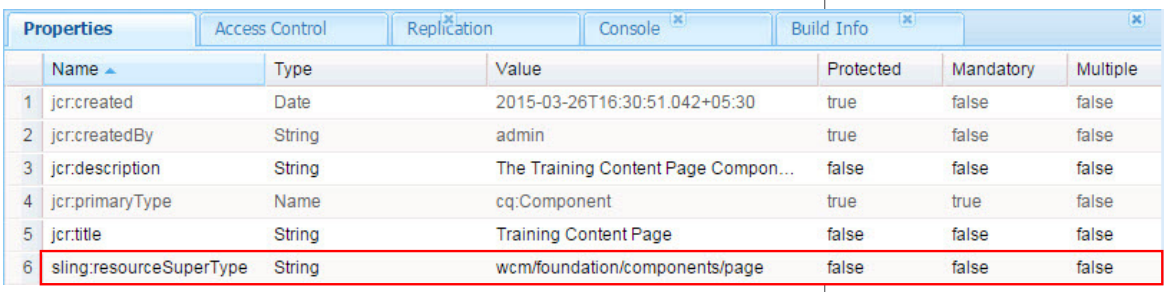
	Name	Type	Value	Protected	Mandatory	Multiple	Auto Created
1	jcr:created	Date	2015-03-26T16:30:51.042+05:30	true	false	false	true
2	jcr:createdBy	String	admin	true	false	false	true
3	jcr:description	String	The Training Content Page Compon...	false	false	false	false
4	jcr:primaryType	Name	cq:Component	true	true	false	true
5	jcr:title	String	Training Content Page	false	false	false	false

2. Enter the following properties in the dialog box, and then click **Add**.

Name	Type	Value
sling:resourceSuperType	String	wcm/foundation/components/page

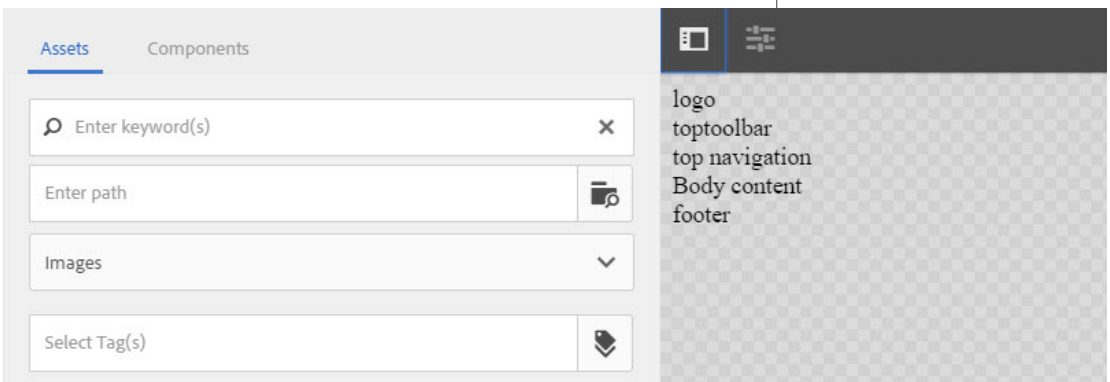


Observe that the property you added appears on the list.

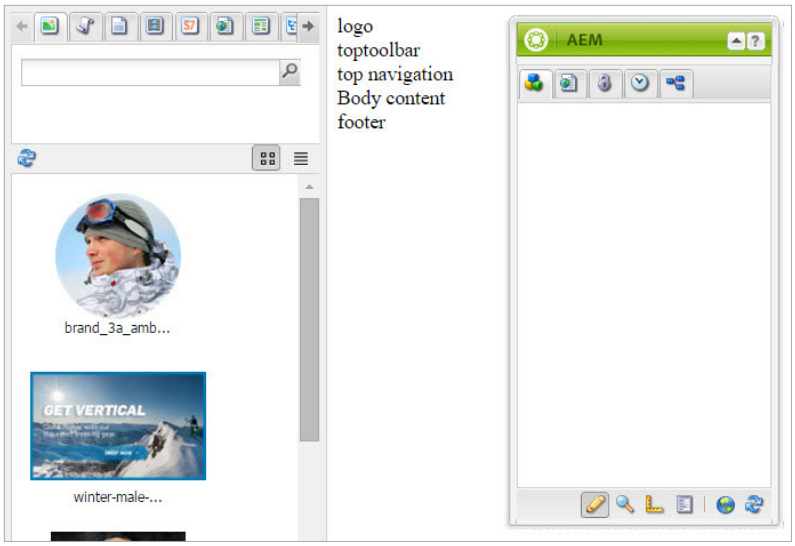


3. Now delete `page-content.html` that you created. This script has become redundant. The intention is to inherit the foundation page component. See the `/libs/wcm/foundation/components/page` component. Open `page.html`. In your component, `body.html` overrides the page component's `body.html`. It includes `head.html` from the page component and provides additional functionalities needed for performing various operations.

4. Go to Site Admin, and open the page you created in touch-optimized UI.
   - If successful, the webpage starts appearing normally with the content you entered in the scripts.
   - Touch-optimized UI displays Assets, Components, and other options. Note that the page starts appearing in touch-optimized UI after inheriting the page component.



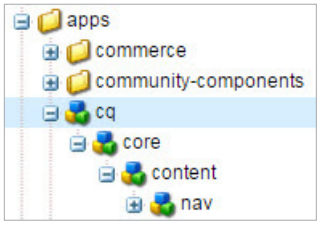
5. Now, open the page in the Classic UI. Note that the page now displays the sidekick.



Extra Credit Exercise
Add a New Navigation Item to The Admin UI

The goal of this exercise is to create a new navigation item called Users, in the left rail of the touch-optimized UI. This item would navigate to the user admin screen of AEM.

1. Open **CRXDE Lite**. Alternatively, click the following link: http://localhost:4502/crx/de/index.jsp
2. Create the following node structure under the `apps` node.



   a. Right-click the `apps` node, and create a new node called `cq`, of type `nt:unstructured`.
   b. Right-click the `cq` node, and create a new node called `core`, of type `nt:unstructured`.
   c. Right-click the `core` node, and create a new node called `content` of type `nt:unstructured`.
   d. Right-click the `content` node, and create a new node called `nav` of type `nt:unstructured`.

NOTE: If the node structure already exists, you can ignore step 2 and its sub-steps.

NOTE: These nodes can all be of type nt:unstructured and do not have to reflect the original node type as in /libs.

3. Right-click the `nav` node, and create a new node called `Users` of type `nt:unstructured`.

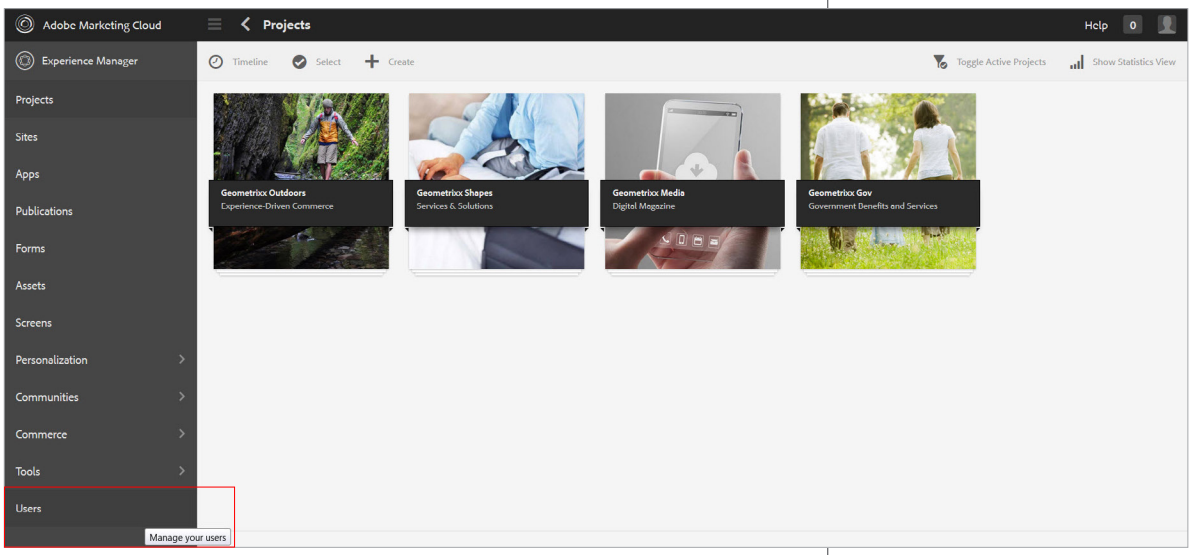
4. Add the following properties for the `Users` node.

Name	Type	Value
id	String	aem-nav-users
href	String	/libs/granite/security/content/useradmin.html
jcr:description	String	Manage your users
jcr:title	String	Users

5. Click **Save All**.

6. Open the **Projects** console, or click the following link: http://localhost:4502/projects.html

   The new navigation item **Users** is added to the left rail.



Add the Design

Creating and assigning a design(er) in AEM 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.

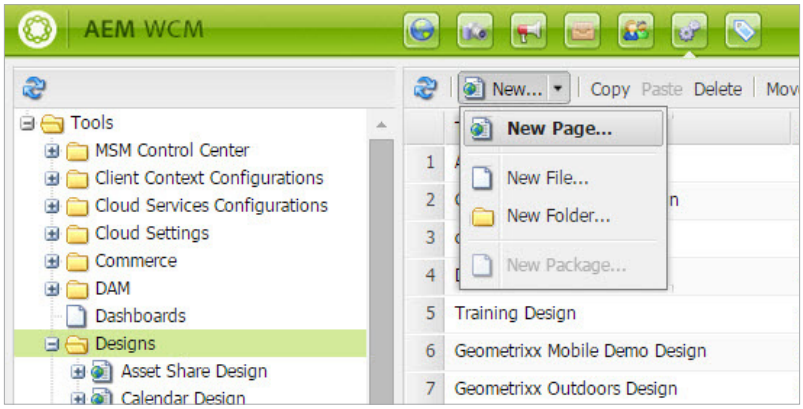
AEM has been developed to maximize compliance with the Web Accessibility Guidelines. Web accessibility means that people with disabilities can perceive, understand, navigate, and interact with the web, and they can contribute to the web. This can include measures such as providing textual alternatives 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 AEM, whereas other aspects must be realized during your project development.

Exercise 7.3
Add a Design to the Page

In this exercise, you will see how to add a design to your website.

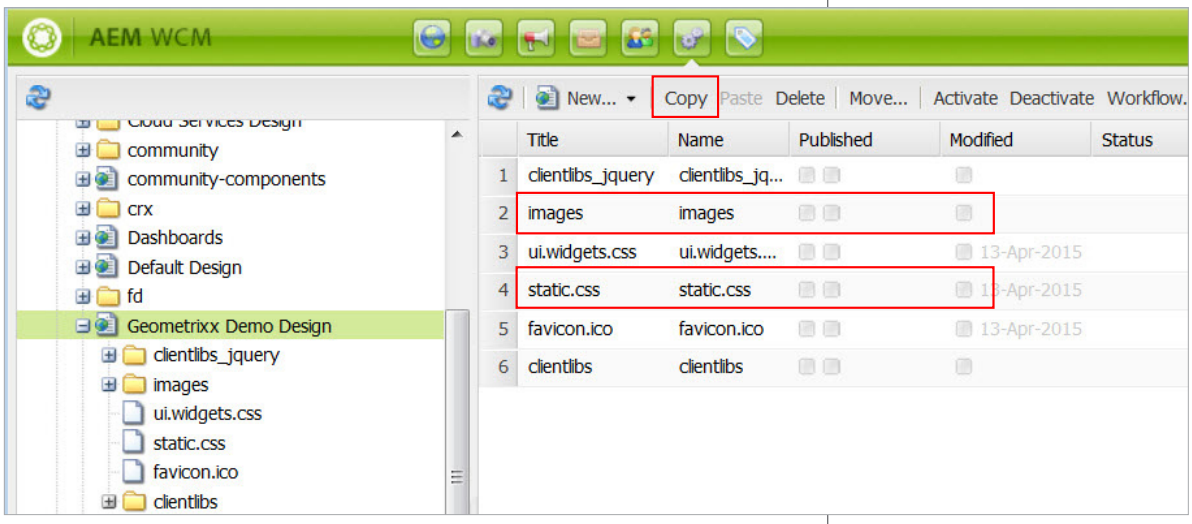
1. Access Tools using the following URL:
   http://localhost:4502/libs/wcm/core/content/misc.html
2. Select the **Designs** folder and click **New** > **New Page**.



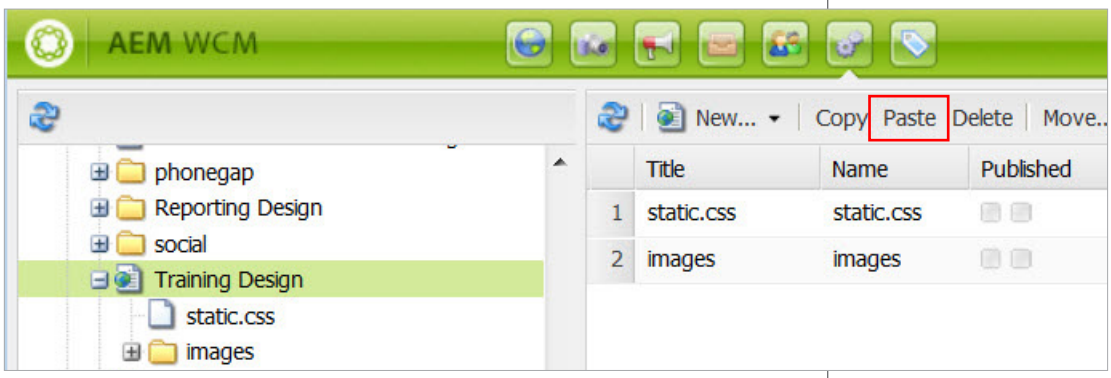
3. Enter the following details, and click **Create**.

Property	Value
Title	Training Design
Name	trainingDesign

4. Select **Geometrixx Demo Design** from the left pane. Select the `images` folder and `static.css` file, and then click **Copy**.



5. Select the newly created **Training Design** page and click **Paste**.

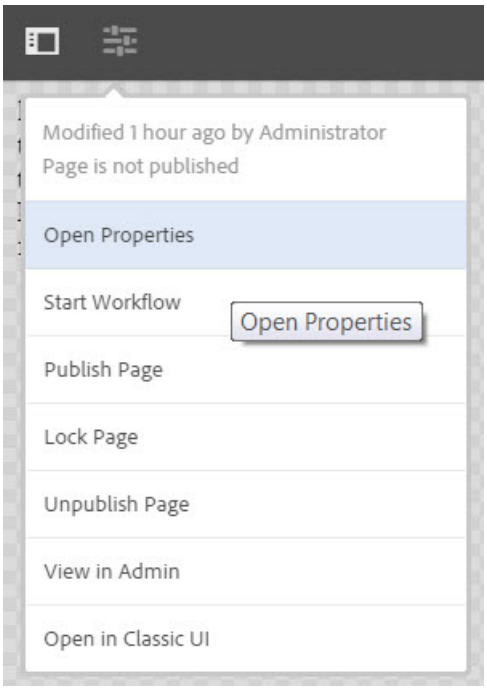


6. Open the root page of your website (Training Site).

NOTE: To copy and paste the files, you can also select the files one at a time, press Ctrl + Alt, click and drag the file to the Training Design folder.

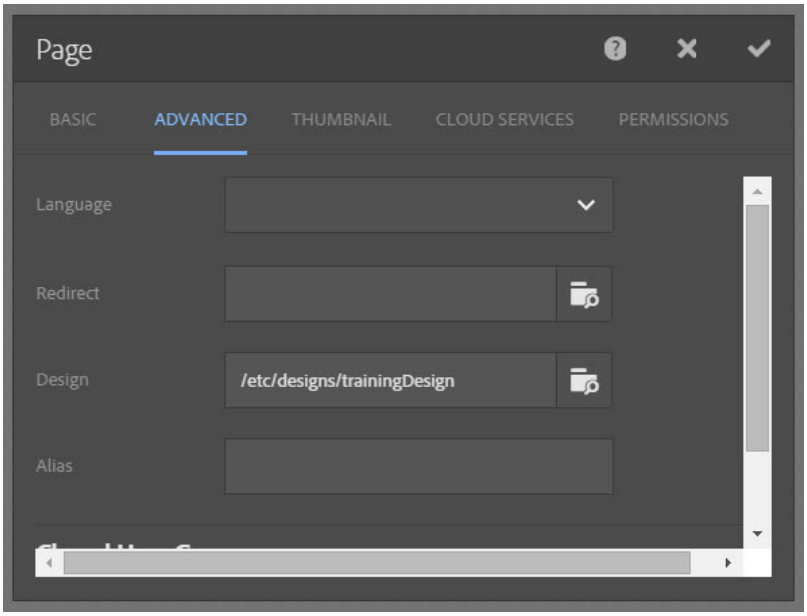
WARNING: If you believe your implementation will have more than one Design(er), which is quite common, it is recommended you create a design structure that will allow multiple Design(er)s to be associated with one project. For example, */etc/designs/trainingDesign*, */etc/designs/trainingDesign/default*, and */etc/designs/trainingDesign/products*.

7. Click the **Page Information** icon on the top bar, and select **Open Properties**.

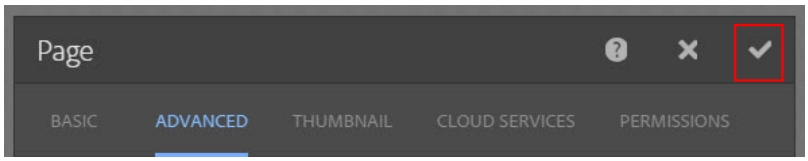


8. On the **Page** dialog box, click the **Advanced** tab. For the **Design** field, browse and select **Training Design**.

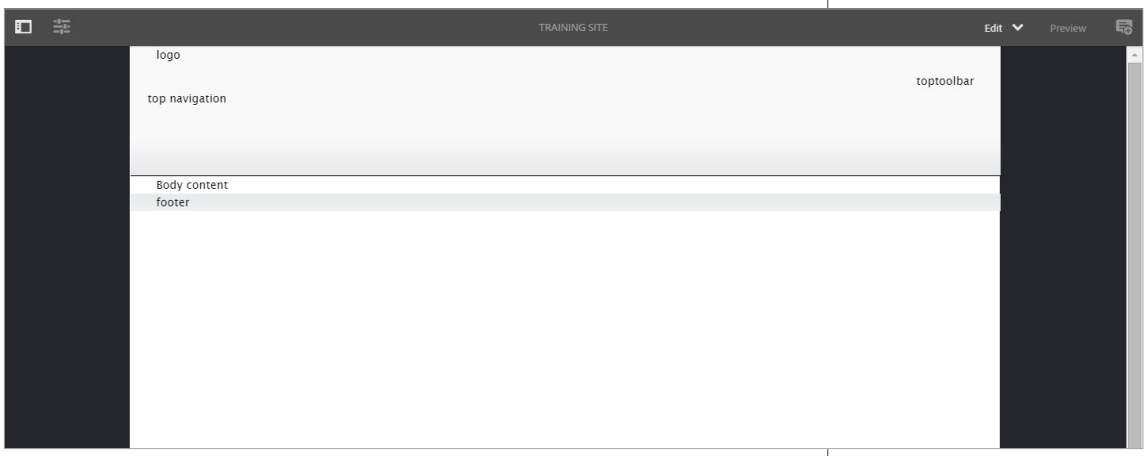
   The path is displayed as `/etc/designs/trainingDesign`.



9. Click the **Done** icon in the upper-right corner.



10. Observe that your website appears with a new look and feel as shown below:



Create Components and Include Them in a Script

You can include a component from within the script using the following tags:

- In Sightly: Using the `data-sly-resource` as shown below.
  The following code includes a component named *topnav* to the script.

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

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

- In JSP:

  ```
  <cq:include path="topnav" resourceType="training/components/topnav"/>
  ```

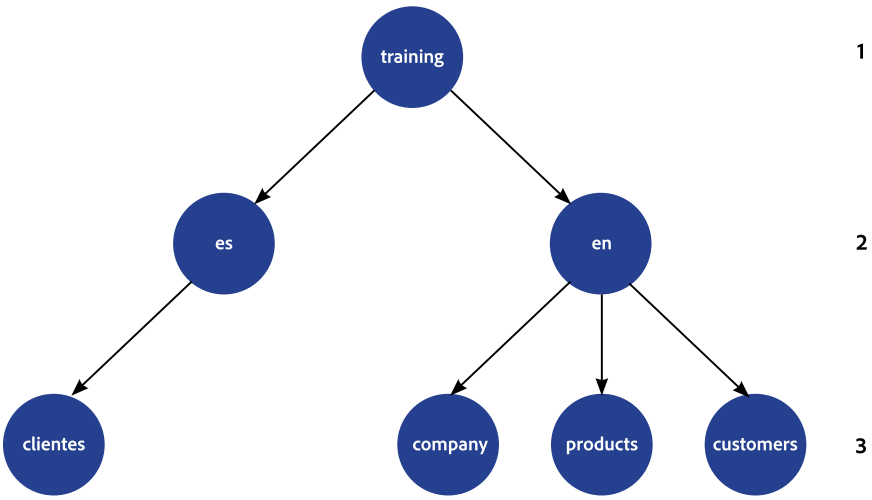
  - path: The path to the resource object to be included in the current request processing. If this path is relative, it is appended to the path of the current resource whose script is including the given resource.
  - resource type: The resource type of the resource to be included. If the resource type is set, the path must be the exact path to a resource object—in this case, adding parameters, selectors, and extensions to the path is not supported. If the resource to be included is specified with the path attribute that cannot be resolved to a resource, the tag may create a synthetic resource object out of the path and this resource type.

Create 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.

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 AEM. Consider the following image, which represents a simple website structure:

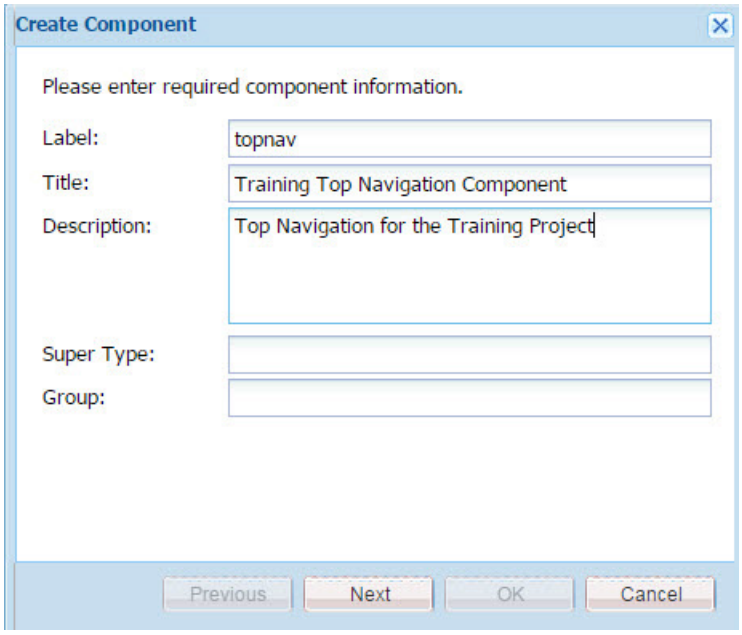


Exercise 7.4
Create a Top Navigation Component and Include it in a Script

In this exercise, you will create the top navigation component. As you saw earlier, you need to list the child nodes available under the root node.

1. Go to CRXDE Lite, and locate **training** > **components**.
2. Right-click the components node, and click **Create** > **Create Component**.
3. Provide the following details:

Property	Value
Label	topnav
Title	Training Top Navigation Component
Description	Top Navigation for the Training Project



4. Click **Next** until you see a screen with the OK button. Click **OK**.
5. Select the `topnav.jsp` file you created. Right-click and click **Rename**. Change the extension from `.jsp` to `.html`. The name of the file becomes `topnav.html`.
6. Open `topnav.html` by double-clicking the file. Insert the following code:

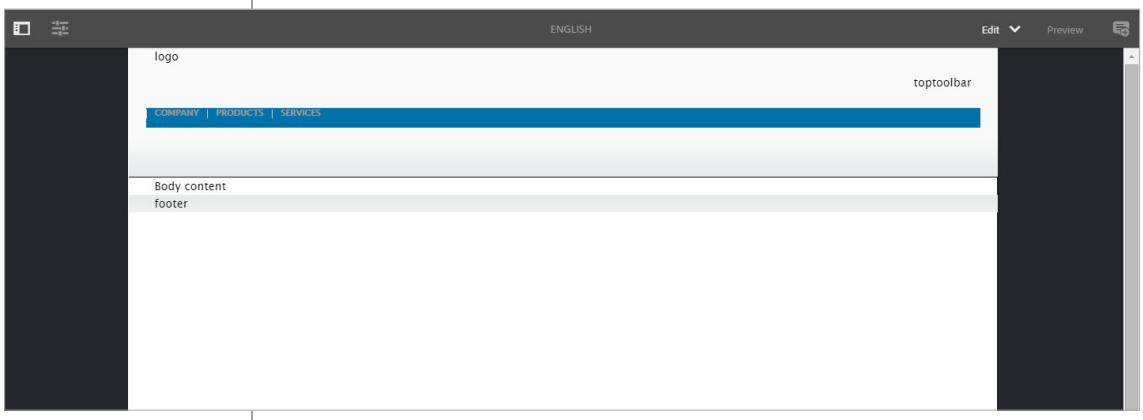
```
<ul class="topnav" data-sly-list="${currentPage.listChildren}">
     <li><a href="${item.path}.html">${item.title}</a></li>
</ul>
```

The `data-sly-list` tag creates an array of the child nodes. The array is then iterated one by one.

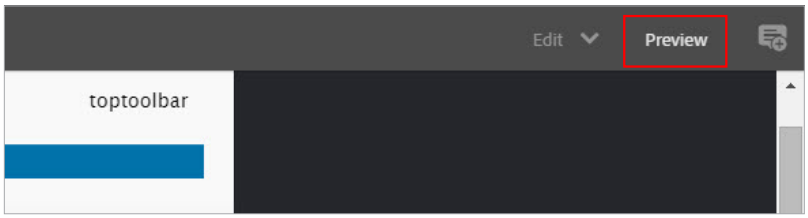
7. Go to `page-content`. Open `header.html` and update the code as follows:

```
<div class="body_header header container_16">
      <div class="header_logo grid_8">logo</div>
      <div class="header_topnav grid_8 search_area">
          <div>toptoolbar</div>
     </div>
    <div class="content_topnav toptoolbar toolbar" data-sly-
    resource="${'topnav' @ resourceType='training/components/topnav'}"></div>
</div>
```

8. Click **Save All**.
9. Open the `/training-site/english` page you created. (http://localhost:4502/editor.html/content/training-site/en.html)
   If you successfully created the top navigation component, it appears as follows. Note that the child pages are displayed as links.



10. Click **Preview** to go to the preview mode.



11. Hover the cursor over the links that were created dynamically. Observe that the hyperlinks are enabled now.
12. Click one of the links and navigate to the respective page.

Example: Including a topnav component using JSP

```
<cq:include path="topnav" resourceType="training/components/topnav"/>
```

Exercise 7.5
Create a Top Navigation Component by Accessing the Root Node

In the previous exercise, you developed a script that displayed the top navigation component. However, if you clicked the links and navigated to the corresponding pages, you would have noticed that the navigation component disappears. In this case, you navigate to a page that does not have any children. The requirement was to display the child pages of the root node (English). Therefore, ideally you need to do the iteration in the root node.

You need to use the `currentPage.getAbsoluteParent()` method to get the root node of a page. You can call this function in a server-side JavaScript file. You can also use Java to accomplish the same, which you will see a little later.

1. Using CRXDE Lite, navigate to the `topnav` component. Right-click the component, and click **Create** > **Create File**.
2. Name the file as `topnav.js` and provide the following code:

```
use(function() {
    return {
        root: currentPage.getAbsoluteParent(2)
    };
});
```

   In the above script, you created a JavaScript function. The function returns the root node using the `currentPage.getAbsoluteParent(2)` method. The argument, 2, ensures that the root node is always `/content/training-site/en`.

3. Change *topnav.html* as follows:

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

4. Click **Save All**.
5. Preview the `/training-site/english` page again.

The `data-sly-use` tag creates a reference to the JavaScript using the `topnav` variable. The `topnav` variable calls the root variable that the JavaScript returns. You then iterate it using the same logic.

Exercise 7.6
Create a Top Navigation Component Using Java

In this exercise, you will develop the top navigation component using Java. You will add an additional logic in Java: hiding a page that is marked as hidden for navigation. You will learn about this option later.

1. Using CRXDE Lite, navigate to the `topnav` component. Right-click the component, and click **Create** > **Create File**.
2. Name the file as `TopNav.java` and provide the following code:

```
package apps.training.components.topnav;
    import java.util.*;
    import java.util.Iterator;
    import com.adobe.cq.sightly.WCMUse;
    import com.day.cq.wcm.api.Page;
    import com.day.cq.wcm.api.PageFilter;
    import info.geometrixx.commons.util.GeoHelper;
public class TopNav extends WCMUse {
        private List<Page> items = new ArrayList<Page>();
        // Initializes the navigation
        public void activate() throws Exception {
            final Page rootPage = getCurrentPage().getAbsoluteParent(2);

if (rootPage != null) {
                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;
        }
    }
```

3. Update the `topnav.html` file as follows:

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

4. Click **Save All**.
5. Refresh the page you created. You should not see any difference. The Java class you created works in the same way the JavaScript worked.

Add a Log Message From the Script

Now that you have created your first non-page-rendering component and encountered the occasional problem, it is time to talk about logging. 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. AEM 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.

Exercise 7.7
Add a Log Message Using the JavaScript File of a Script

1. Open `topnav.html` that you updated in the previous exercise. Modify it to use `topnav.js`. Note that the code is updated with the logic to exclude hidden pages.

```
<ul class="topnav" data-sly-use.topnav="topnav.js" data-sly-list="${topnav.root.listChildren}">
    <li data-sly-test="${!item.isHideInNav}">
        <a href="${item.path}.html">${item.title}</a></li>
</ul>
```

2. Open `topnav.js` that you created before.
3. Update the code as follows. You add a log entry using the `log.error()` method. The method adds a string with the name of the page.

```
use(function() {
    log.error("Hello world: I am from" +currentPage.getName());
        return {
            root: currentPage.getAbsoluteParent(2)
        };
});
```

NOTE: If you are using a JSP file for the script, use the log.info() method. For example, `log.info("child page not found", child.getTitle());`

4. Refresh one of the pages you created.
5. Go to the installation folder of AEM.
6. Navigate to the `crx-quickstart/logs` folder.
7. Open the `error.log` file.
8. Search for '*Hello World*', the string that you entered as an error log. Observe that the string is displayed with the page name in the `error.log` file.

```
'DRIDDSVC\x01<<<\x01\x00\x00\x00\x00\x00\x00\x00>>>ibs/granite/typek...nents/repl/clie' for HttpChannelOverHttp@903628{r=0,c=false,a=IDLE,uri=-}
27.03.2015 11:27:53.833 *WARN* [qtp944052238-210] org.eclipse.jetty.http.HttpParser badMessage: 400 Illegal character 0x1 in state=METHOD in
'DRIDDSVC\x01<<<\x01\x00\x00\x00\x00\x00\x00\x00>>>urity/userinfo.js...\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' for
HttpChannelOverHttp@2f0356f4{r=0,c=false,a=IDLE,uri=-}
27.03.2015 11:27:54.258 *WARN* [0:0:0:0:0:0:0:1 [1427435874134] GET /content/training-site/en.html HTTP/1.1] com.day.cq.wcm.webservicesupport.impl.ConfigurationManagerImpl Cloud
service [] not found
27.03.2015 11:27:54.328 *ERROR* [0:0:0:0:0:0:0:1 [1427435874134] GET /content/training-site/en.html HTTP/1.1] apps.training.components.topnav.topnav$html Hello World!!en
27.03.2015 11:27:54.683 *WARN* [0:0:0:0:0:0:0:1 [1427435874518] GET /content/training-site/en.html HTTP/1.1] com.day.cq.wcm.webservicesupport.impl.ConfigurationManagerImpl Cloud
service [] not found
27.03.2015 11:27:54.702 *ERROR* [0:0:0:0:0:0:0:1 [1427435874518] GET /content/training-site/en.html HTTP/1.1] apps.training.components.topnav.topnav$html Hello World!!en
27.03.2015 11:27:54.833 *WARN* [qtp944052238-214] org.eclipse.jetty.http.HttpParser badMessage: 400 Illegal character 0x1 in state=METHOD in
'DRIDDSVC\x01<<<\x01\x00\x00\x00\x00\x00\x00\x00>>>sonalization/touc...\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00' for
HttpChannelOverHttp@3df49a6d{r=0,c=false,a=IDLE,uri=-}
27.03.2015 11:27:55.281 *INFO* [0:0:0:0:0:0:0:1 [1427435874809] GET /libs/wcm/core/content/components.1427435874670.html HTTP/1.1]
com.day.cq.wcm.core.impl.components.ComponentCacheImpl Loaded 1215 components in 468ms
27.03.2015 11:27:55.833 *WARN* [qtp944052238-210] org.eclipse.jetty.http.HttpParser badMessage: 400 Illegal character 0x1 in state=METHOD in
```

AEM Authoring Framework — Dialog Boxes

Overview

A dialog box is a temporary window that prompts the user to provide input that can be used for further action. This chapter helps you to create dialog boxes in AEM for components and global content.

Objective

In the previous chapter, you learned how to create components. In this chapter, you will learn how to create dialog boxes for components. You will specifically learn how to do the following:

- Create dialog boxes for components
- Create design dialog boxes for global content
- Use the Edit_Config property to enhance components

Create Dialog Boxes for Components

So far, you have focused mostly on rendering content (static and dynamic), which is important because one could argue that rendering content comprises 50% of what you do as an AEM developer. However, most components that you create will allow the author to input that content. Typically, the mechanism that authors will use to input content is through a dialog box.

An AEM dialog box 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).

AEM uses two types of dialog boxes:

- One for Classic-UI
- One for touch-optimized UI

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

Touch-Optimized UI

Touch-optimized UI makes use of Granite.js. All the elements that 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`.

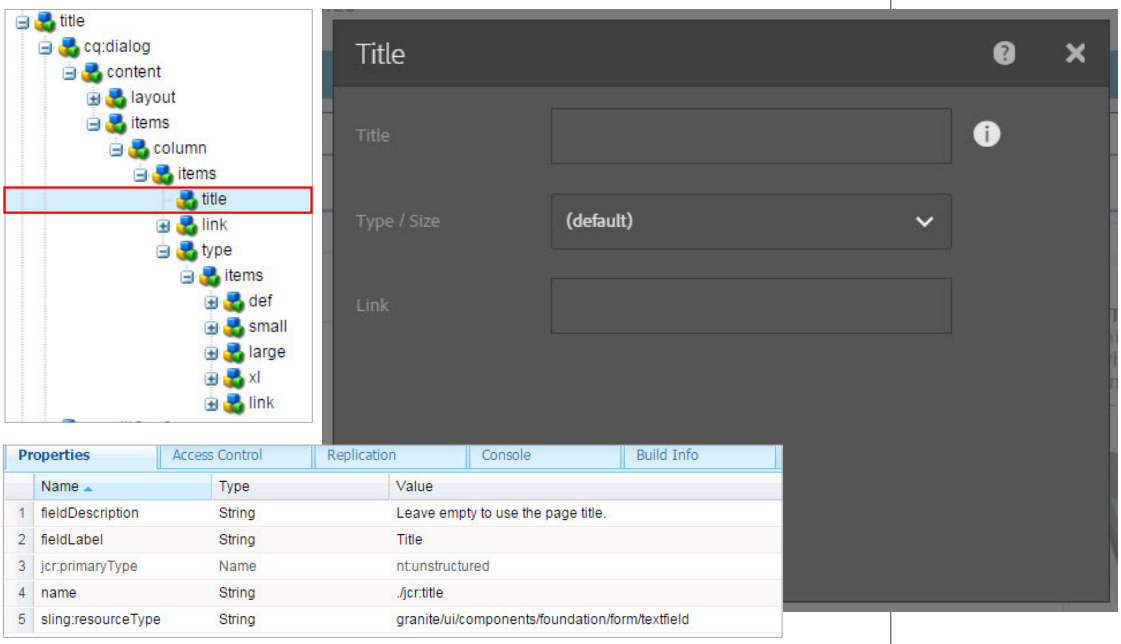
A touch-optimized UI dialog box is defined by using nodes of type `nt:unstructured`. To define the type of control used, you need to set the node's `sling:resourceType` property. For example, to define a text field on a Touch UI dialog, set the `sling:resourceType` property to `granite/ui/components/foundation/form/textfield`.

The following table lists the `sling:resourceType` values that are used to create the components of the dialog box.

sling:resourceType	Description
granite/ui/components/foundation/container	Defines a container for the dialog.
granite/ui/components/foundation/layouts/tabs	Defines a tab that is used in the dialog.
granite/ui/components/foundation/section	Defines a section within a tab.
granite/ui/components/foundation/layouts/fixedcolumns	Defines fixed columns.
granite/ui/components/foundation/form/textfield	Defines a text field that lets authors enter data.
granite/ui/components/foundation/form/textarea	Defines a text area field that lets author more data than a text field.

In the following figure, the node structure on the left produces the dialog box pictured on the right.

- 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).



Now, we will examine the properties on the title widget itself. In the figure above, the title widget has five properties:

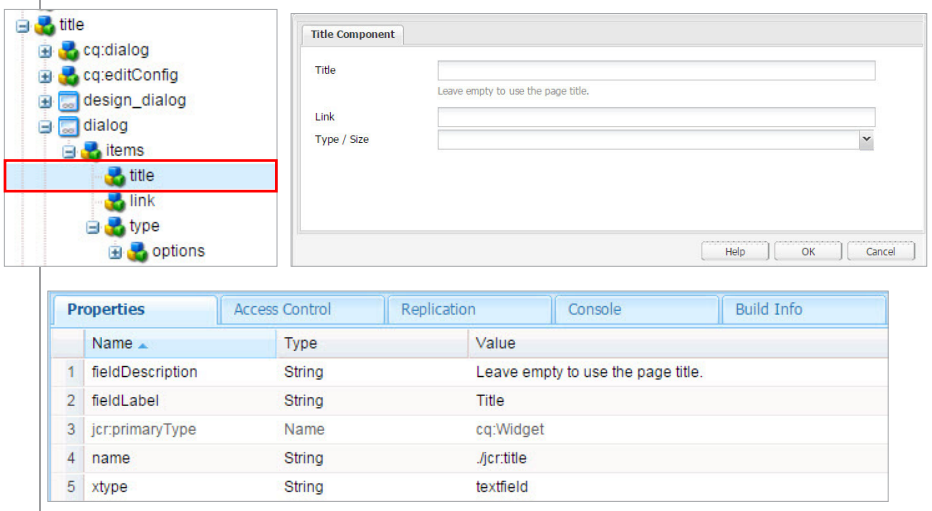
- **jcr:primaryType:** Defines the Node type, in this case, nt:unstructured
- **fieldLabel:** Tells the author what to do, in this case, 'Title'
- **fieldDescription:** Gives the author more information, in this case, 'Leave empty to use the page title'.
- **name:** Tells the JS code that backs this 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

Classic UI

AEM's 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

In the figure below, the node structure on the left produces the dialog box pictured on the right.



The `cq:dialog` node produces the outer gray outline. The `cq:TabPanel` node produces the inner gray outline. The `cq:Panel` node(s) produce the individual tabs—the panel(s) of the dialog box. The input widgets for each panel are placed subordinate to the panel nodes. The 'widget nodes' are of the Node type 'cq:Widget'.

You will notice that the `cq:WidgetCollection` node is a child of the Tab panel and the widget collection node is a child of the panel. Every time you can have more than one of any object, you need a widget collection to hold the objects. So:

- One cq:dialog node - no WidgetCollection
- One cq:TabPanel node - no WidgetCollection

However, you can have more than one panel on a Tab panel, so you need a widget collection to hold the panels. Similarly, you can have more than one widget on a panel so you need a widget collection to hold the widgets.

There are few places in the AEM environment where specific names must be used. The dialog box is one of those places. Rules:

- Dialog box nodes must be of type cq:dialog and must be named 'dialog' ('design_dialog' for design dialog boxes).
- While other nodes may also be named 'items', widget collections are always named 'items'.

If you name the nodes other than the expected names, AEM will not recognize the nodes and will not produce the expected construct.

Now let us examine the properties on the widget itself. In the previous figure, the title widget has five properties:

- **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'.
- **xtype:** Defines the type of input form, in this case, a 'textfield' - single line, alphanumeric

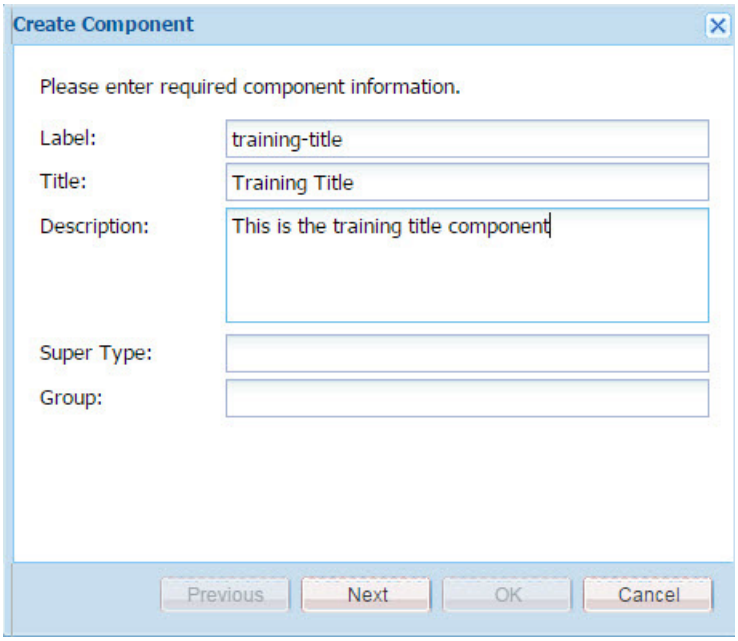
Exercise 8.1
Create a Training Title Component

In this section, you will create a Title component that displays the page title.

1. Go to CRXDE Lite, and locate **training** > **components**.
2. Right-click the components folder and click **Create** > **Create Component**.

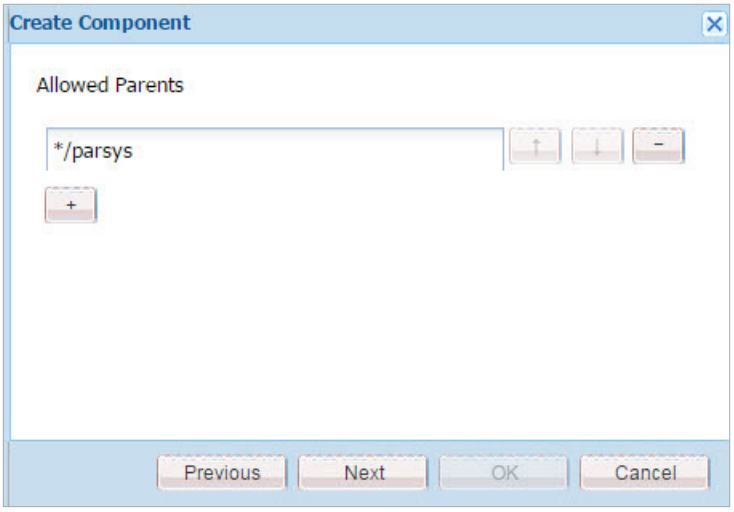
3. Add the following details:

Property	Value
Label	training-title
Title	Training Title
Description	This is the training title component



4. Click **Next**. The Advanced Component Settings screen appears.
5. Click **Next**.
6. Click the + symbol for Allowed Parents. Type the following:

```
*/parsys
```



7. Click **Next**.
8. Click **OK** on the Allowed Children screen. The component is created.

9. Save the changes.
10. Rename the `training-title.jsp` page created to `training-title.html`.
11. Double-click `training-title.html` to open it. Remove the existing code, and add the following:

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

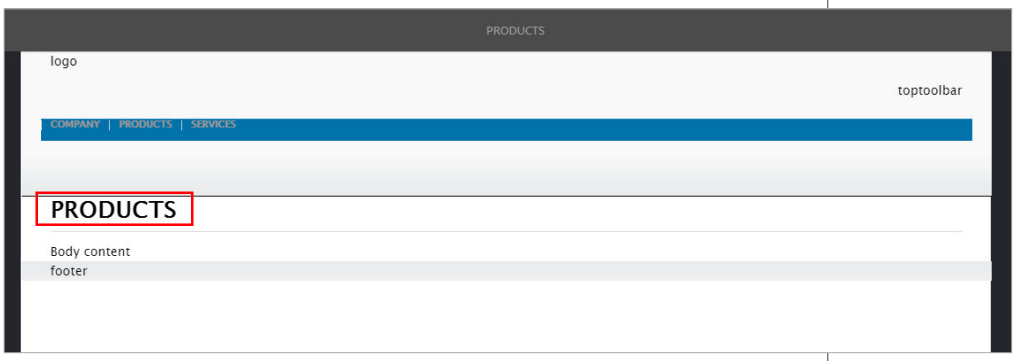
12. Right-click the `training-title` component, select **Create** > **Create File...**
13. Name the file as `training-title.js` file. Add the following code:

```
"use strict";
    use(function () {
        var CONST = {
            PROP_TITLE: "jcr:title",
            PROP_PAGE_TITLE: "pageTitle",
        }
        var title = {};
        title.text = granite.resource.properties[CONST.PROP_TITLE]
           || wcm.currentPage.properties[CONST.PROP_PAGE_TITLE]
           || wcm.currentPage.properties[CONST.PROP_TITLE]
       || wcm.currentPage.name;
        return title;
    });
```

14. Open `body.html` of the page component (page-content) you created and add the following code:

```
<div data-sly-include="header.html"></div>
    <div class="body_content container_16">
        <div class="content_title grid_16" data-sly-resource="${'title' @
    resourceType='training/components/training-title'}"></div>
        <div class="content_title grid_16">Body content</div>
    </div>
<div data-sly-include="footer.html"></div>
```

15. Open one of the pages created. Note that the title appears on the page. For example, the Products page.
http://localhost:4502/editor.html/content/training-site/en/products.html



Exercise 8.2
Create a Dialog Box for a Classic UI

In the previous exercise, you created a component and added it in the template. In the following exercise, you will create a dialog box that works in Classic UI. You can use the dialog box to change the title.

1. Select the training-title node. Right-click the node, and click **Create** > **Create Dialog**. Create a dialog box with the following properties:

Property	Value
Label	dialog
Title	Title Component

2. The newly created dialog box appears as a node. Save the node.
3. Expand the `dialog` node and locate `/dialog/items/items/tab1`. Right-click the `tab1` node. Click **Create** > **Create Node**. Add the following details:

Property	Value
Name	items
Type	cq:WidgetCollection

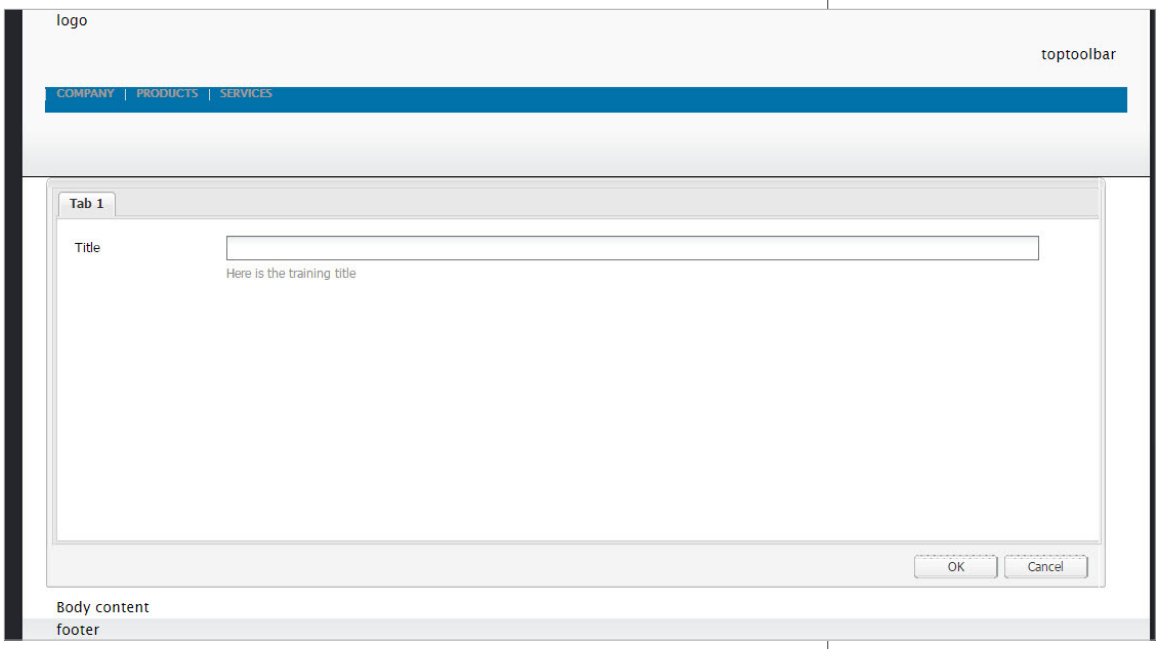
4. Right-click the `items` node you created and create a node with the following details:

Property	Value
Name	title
Type	cq:Widget

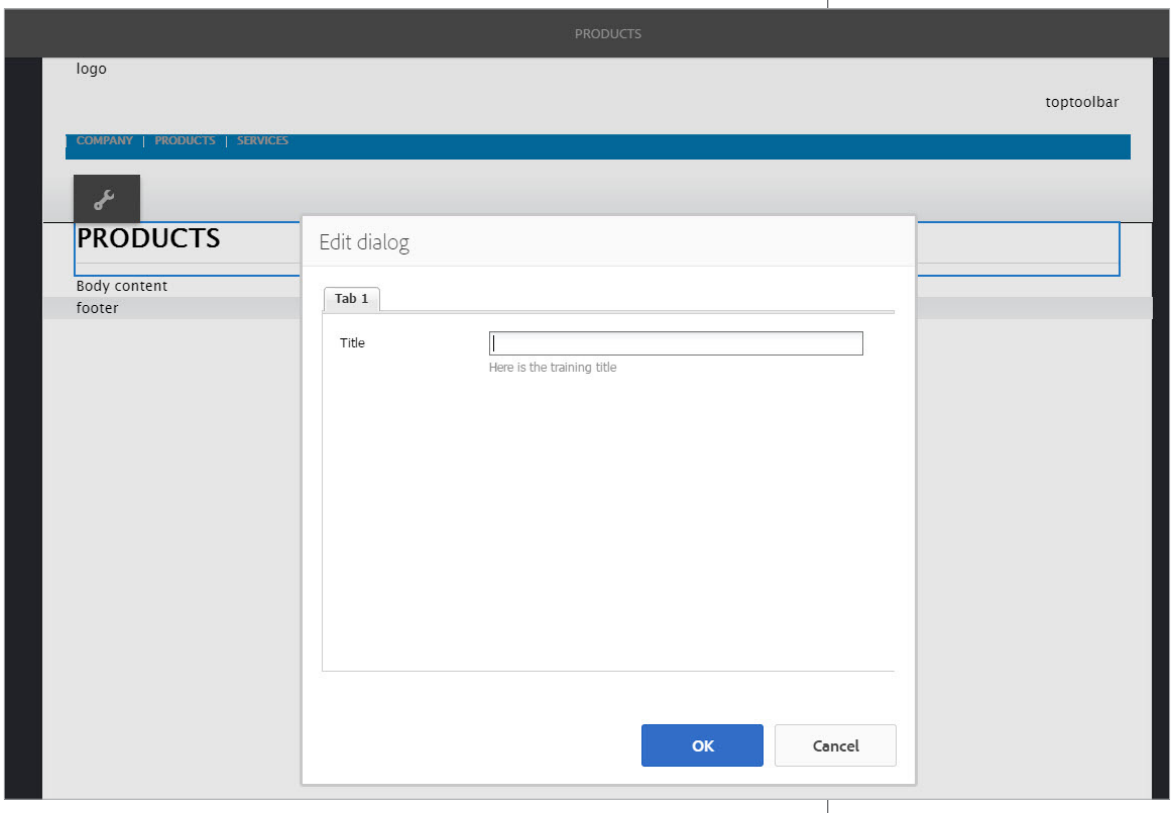
5. Add the following properties to the node:

Name	Type	Value
name	String	./jcr:title
fieldLabel	String	Title
fieldDescription	String	Here is the training title
xtype	String	textfield

6. Open one of the pages you created in Classic UI. Double-click the title you entered. Notice that the dialog box appears.



7. Now, open the page in touch-optimized UI. Double-click the title you entered. Notice that the dialog box appears. However, this dialog box is not optimized for touch-optimized devices. In the next exercise, you will create a dialog box for touch-optimized UI.



The Dialog Conversion Tool

You have just learned how to create a dialog box for the Classic UI. Creating a dialog box for the touch-optimized UI involves the creation many nodes and properties. The Dialog Conversion Tool enables you to easily convert Classic UI dialog boxes (based on ExtJS) to the touch-optimized UI dialog boxes (based on Granite).

NOTE: The result of the conversion may vary based on the complexity of the dialog box. You need to verify the resultant dialog box and make adjustments to its properties accordingly.

The tool extends existing components that have dialog boxes designed for the Classic UI. Extension of the component is achieved by creating a corresponding dialog box for the touch-optimized UI at the same location in the content tree.

The following table shows the difference between the two dialog boxes.

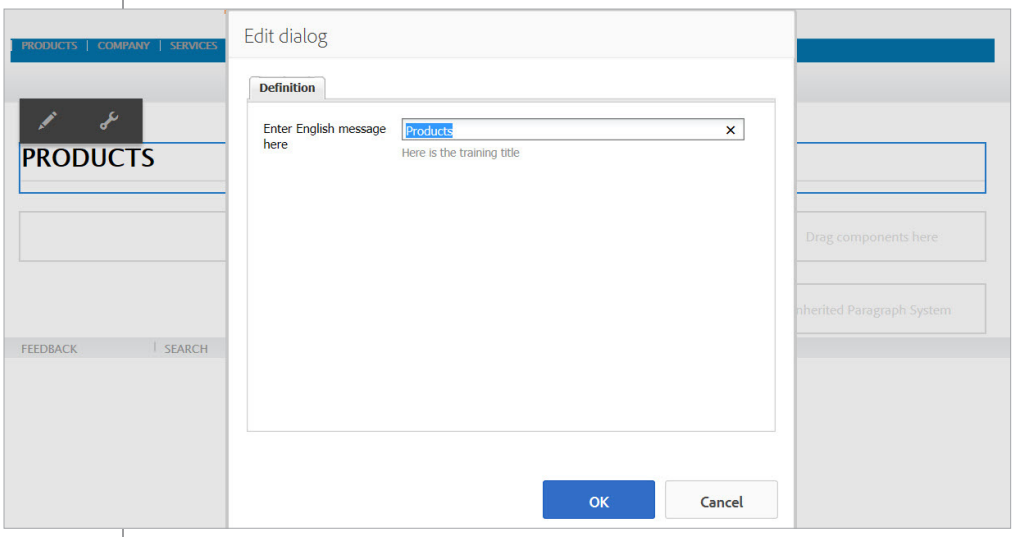
Classic UI	Touch-optimized UI

Node with the following properties:
`name = dialog`
`type = cq:Dialog`
	Node with the following properties:
	`name = cq:dialog`

Exercise 8.3
Convert a Classic UI Dialog Box to a Touch-Optimized UI Dialog Box

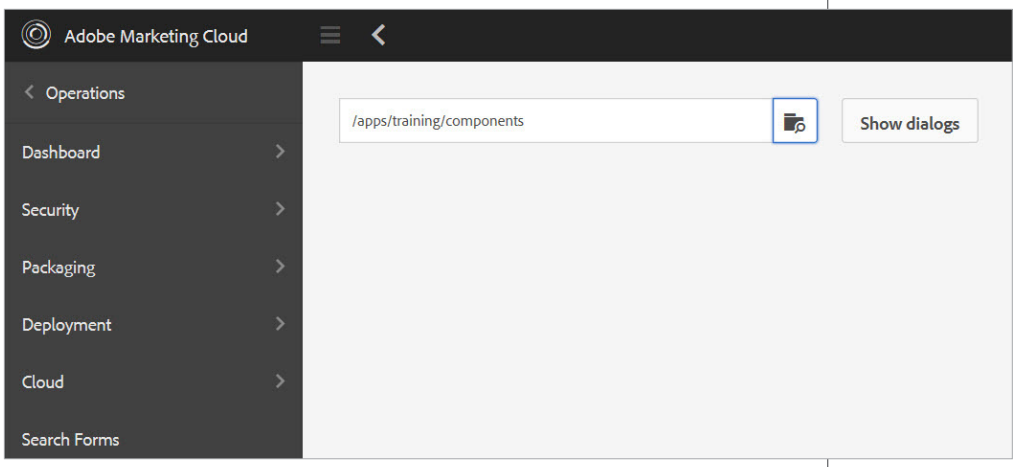
The goal of this exercise is to convert the Classic UI dialog box created in the previous exercise to a touch-optimized UI dialog box, using the Dialog Conversion Tool.

1. Upload and install the package `cq-dialog-conversion-content-1.0.0` provided with the USB contents.
2. Open any one of the training pages created. Double-click the title component and see the dialog box. It is currently in the Classic UI.

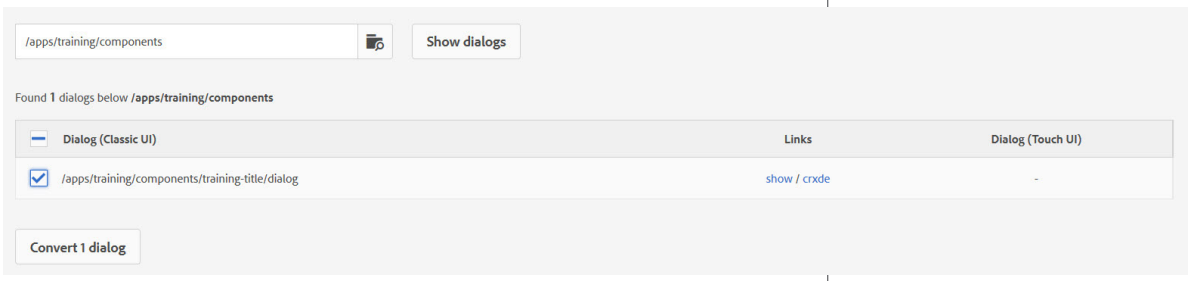


3. In the **Sites** console, navigate to **Tools** > **Operations** > **Dialog Conversion**.

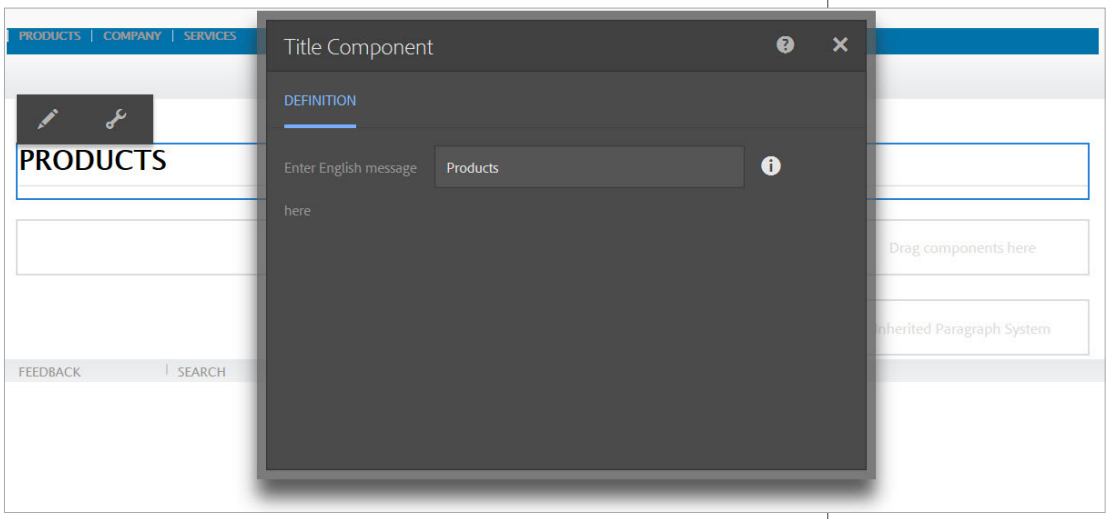
4. Browse and select the components folder (or type `/apps/training/components`). Click **Show dialogs**.



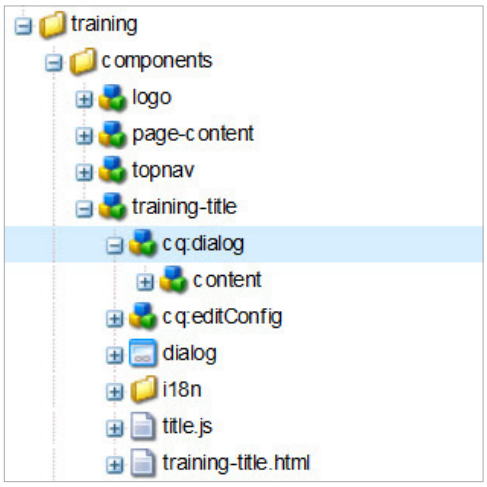
5. A list of existing Classic UI dialog boxes is displayed. Currently, you have just one dialog box (that is, `/apps/training/components/training-title/dialog`). Select the dialog box, and click **Convert 1 dialog**.



6. Open the training page again, and double-click the title component. You will see that the dialog has now upgraded to the touch-optimized UI.



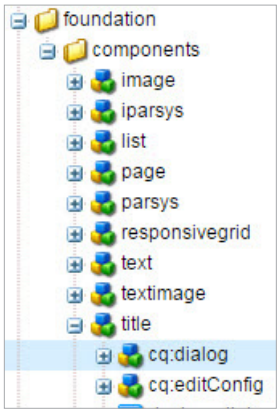
7. Open CRXDE Lite, and navigate to `/apps/training/components/training-title`. Expand the `training-title` component. You will see that the `cq:dialog` node is added. This node was created by the conversion tool. You can then modify the properties of the node to suit your requirements.



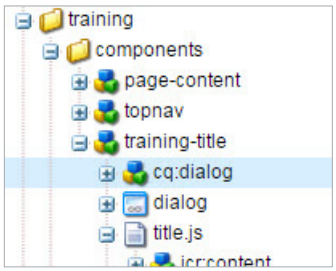
8. For the purposes of this training, delete the node. The next exercise shows how to create a dialog box for the touch-optimized UI without the conversion tool.

Exercise 8.4
Create a Dialog Box for Touch-Optimized UI

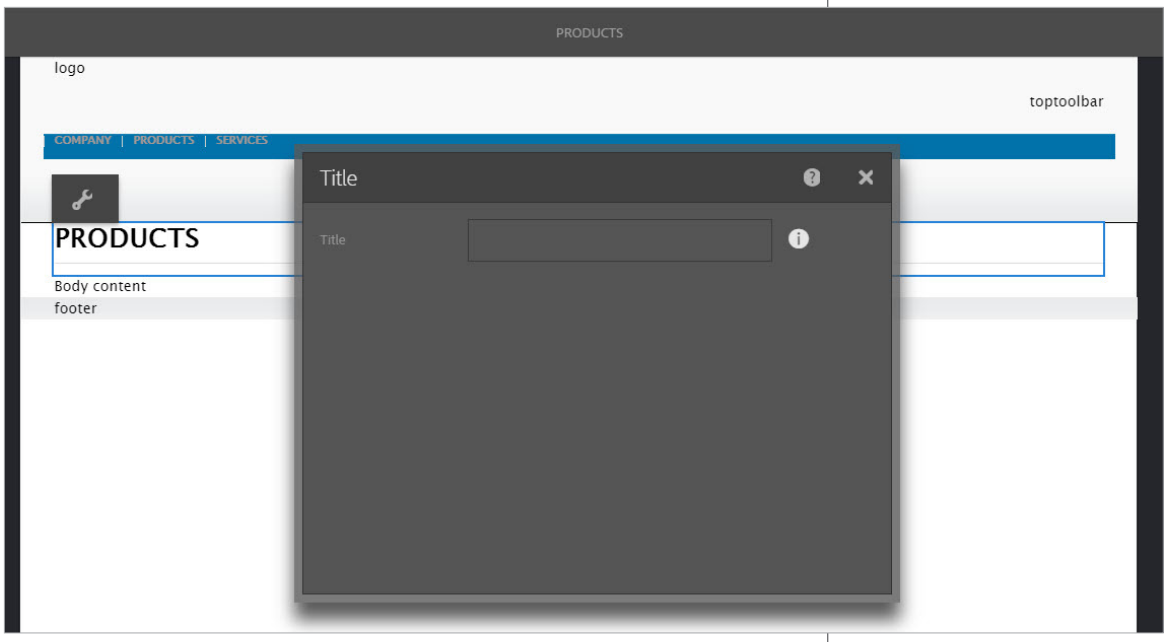
1. Go to the `/libs/wcm/foundation/components/title` folder.
2. Copy the `cq:dialog` node.



3. Paste it in the training-title node.



4. Move to the `/apps/training/components/training-title/cq:dialog/content/items/column/items` node.
5. Click **Save All**.
6. Open the page in the touch-optimized UI.
7. Double-click the title displayed. Note that a dialog box appears.



8. Switch back to Classic UI.
9. Double-click the `Title` component to open the dialog box.
   Observe that the dialog box you saw in the previous exercise appears.

NOTE: Apart from the classic UI dialog box being ExtJS based, and that of the touch-optimized UI being Granite based, check the `cq:dialog` node and the `dialog` node to see the difference in their properties.

Extra Credit - Create a listChildren Component

The following exercise tests your knowledge of component creation and taking input from an author. The goal is to create a listChildren component. Using the Page, PageFilter, and PageManager classes, create a listChildren component. Remember that you can look up these classes in the Java docs. You have all the information you need, from the topnav and title components, to complete this component.

1. Include/import the appropriate Java classes.
2. Get a property whose value will become the parent of the list. The property should be a path.
3. If the property is empty, use a default. (Perhaps a static path or path to the current page.)
4. Use the path that the author entered to get the parent page. (Hint: The PageManager class has a method to get a page object when you know its path.)
5. Set up a page iterator, filtering by valid pages.
6. Iterate over the children.
7. For each child, write a link.
8. Create a dialog box.
9. Place an input widget on the dialog box so the author can enter the list parent.
10. Edit `body.html` to include the component.

Use Design Dialog Boxes for Global Content

As discussed earlier, 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

To demonstrate the use of the Design dialog box, we will be creating a Logo component. The Logo component will use the smartimage widget. This is the first time that we will be dealing with binary content.

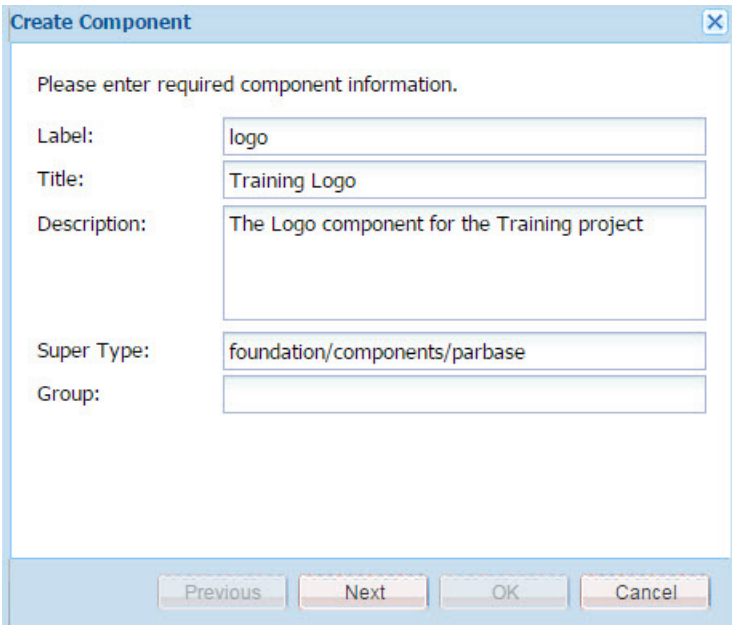
Therefore, it is important to note the use of the resource SuperType foundation/components/parbase. The sling:resourcesSuperType property is a key component because it allows components to inherit attributes from other components, similar to subclasses in object-oriented languages such as Java, C++, and so on. The Parbase component defines tree scripts to render images, titles, and so on so that all components subclassed from this parbase can use this script.

Exercise 8.5
Create a Logo Component

In this exercise, you will create a logo component. You will use a Design dialog box to create the logo component because a logo is a global content. It does not change from page to page. You need to set it only once. Later, if you want to change the logo, you need to change it only once. It should be updated in all the pages that use the template.

1. Log in to CRXDE Lite. Right-click the `components` folder of the `training` node. Click **Create** > **Create Component**.
2. Enter the following details:

Property	Value
Label	logo
Title	Training Logo
Description	The Logo component for the Training project
Super Type	foundation/components/parbase



3. Click **Next** until you reach the last screen, and then click **OK**.
4. Click **Save All**.

5. Open the `logo.jsp` file, enter the following code, and then click **Save All**:

```
<%@include file="/libs/foundation/global.jsp"%><%
    %><%@ page import="com.day.text.Text,
        com.day.cq.wcm.foundation.Image,
        com.day.cq.commons.Doctype" %><%
       String home = Text.getAbsoluteParent(currentPage.getPath(), 2);
       Resource res = currentStyle.getDefiningResource("fileReference");
       if (res == null) {
        res = currentStyle.getDefiningResource("file");    }
       log.error("path is:" + currentStyle.getPath());
       %><a href="<%= home %>.html"><%
       if (res == null) {
        %>Home Page Placeholder<%
       } else {
        Image img = new Image(res);
        img.setItemName(Image.NN_FILE, "file");
        img.setItemName(Image.PN_REFERENCE, "fileReference");
        img.setSelector("img");
        img.setDoctype(Doctype.fromRequest(request));
        img.setAlt("Home Page Placeholder");
        img.draw(out);
    }
    %></a>
```

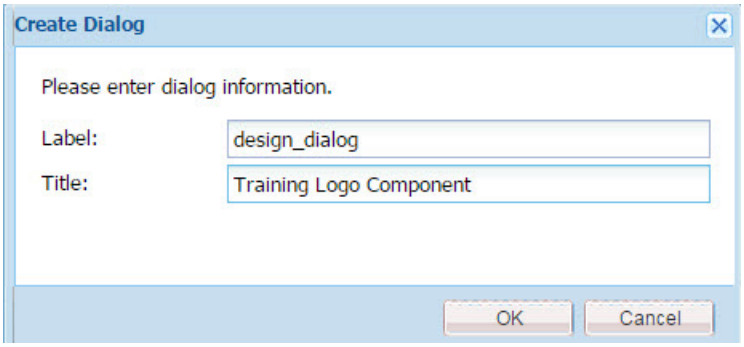
6. Under the `page-content` component, open the `header.html` of `page-content` file and replace the code that you used for the logo component.

```
<div class="body_header header container_16">
    <%@include file="/libs/foundation/global.jsp" %>
        <div class="header_logo grid_8" data-sly-resource="${'logo' @
    resourceType='training/components/logo'}"></div>
        <div class="header_topnav grid_8 search_area">
            <div>toptoolbar</div>
        </div>
        <div class="content_topnav toptoolbar toolbar" data-sly-resource="${'topnav' @
    resourceType='training/components/topnav'}"></div>
    </div>
```

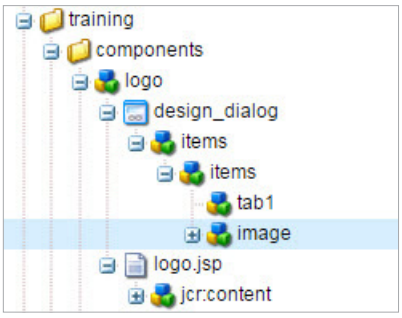
7. Create a new Design dialog box for the logo component. To do so, right-click the `logo` component, and select **Create** > **Create Dialog...**.

8. Add the following properties to the dialog box, and click **OK**.

Property	Value
Label	design_dialog
Title	Training Logo Component



9. To create the smartimage widget that you will use for input and maintenance of the logo image, copy the node `/libs/foundation/components/image/dialog/items/image` and then paste to the logo's Design dialog box so that it is a peer of the `tab1` node. Example: `/apps/training/components/logo/design _ dialog/items/items`



In addition, you may have noticed that this widget was pasted in the location where tabs are typically located. The smartimage xtype (along with few others) is unique in that it provides a better content author experience when 'casted' as a tab.

10. Click **Save All**.

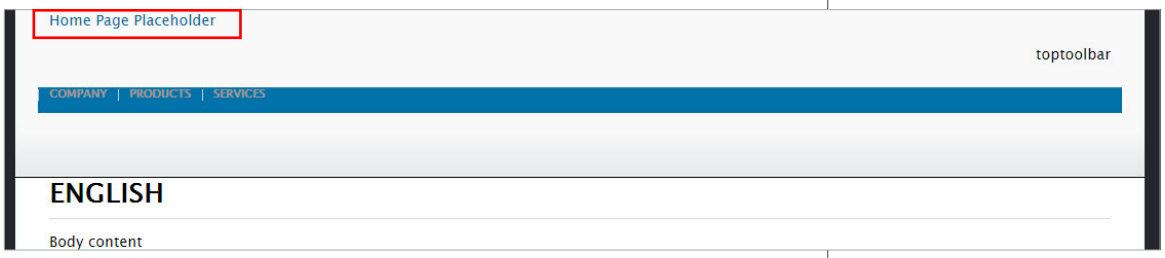
11. Review the properties associated with this smartimage widget.

NOTE: Instead of always reinventing the wheel, it is often more efficient to copy and paste existing Dialog boxes or widgets that meet your needs. However, it is wise to review what you have just copied to better understand the internal workings of AEM.

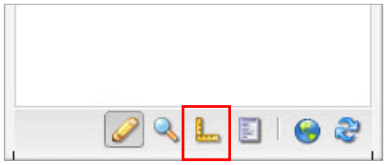
Properties | Access Control | Replication | Console | Build Info

	Name	Type	Value
1	cropParameter	String	./imageCrop
2	ddGroups	String[]	media
3	fileNameParameter	String	./fileName
4	fileReferenceParameter	String	./fileReference
5	jcr:primaryType	Name	cq:Widget
6	mapParameter	String	./imageMap
7	name	String	./file
8	requestSuffix	String	.img.png
9	rotateParameter	String	./imageRotate
10	title	String	Image
11	xtype	String	html5smartimage

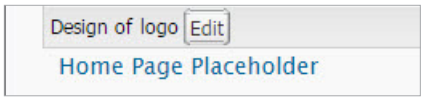
12. Open the training page that you created in the Classic UI. Observe that the placeholder appears in the page.



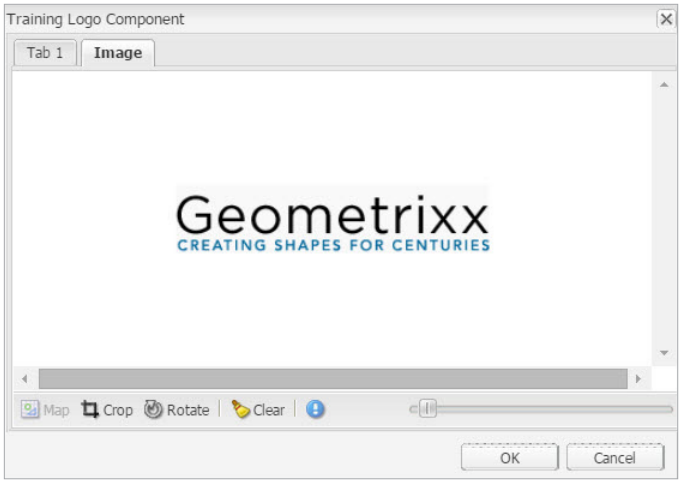
13. Switch to the Design mode by clicking the **Design** icon in the sidekick.



14. In the Design mode, click **Edit** in the Design of logo component.



15. Go to the **Image** tab. Double-click the Image icon, navigate to the exercise folder, and then select `logo.png`.



16. If successful, you should see the custom page logo, similar to the image below:

Extra Credit: Modify Your topnav Component

This extra credit exercise tests your knowledge of creating design elements. The goal is to modify your local topnav component so that the code allows an author with design privileges to set the parent of the navigation by entering a path.

- Make whatever modifications you need to obtain the following result:
  The author/designer can enter a path design element to set the parent of the top navigation list of pages.

Use cq:EditConfig to Enhance the 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:

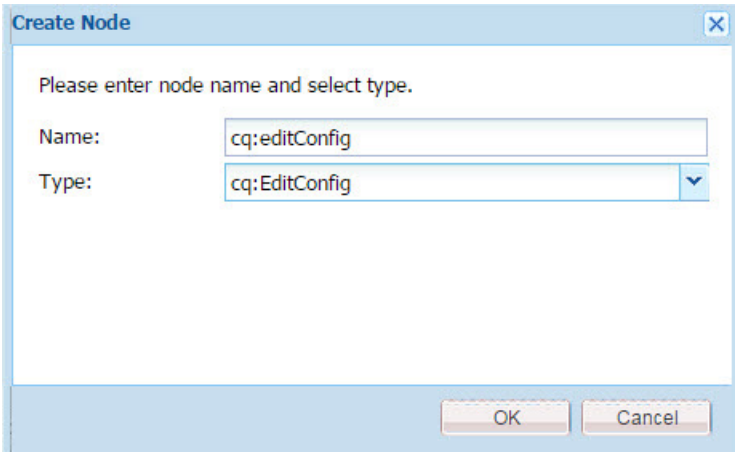
- Node of type cq:editConfig that is a child node to the cq:component node
- Configuration node that is a child of the cq:EditConfig node. This node defines what action you are configuring.
- Asset node that is a child of the cq:dropTargets node. This node defines what types of assets the paragraph will accept. In this example, assets of type image.

Exercise 8.6
Enable In-place Editing in the Title Component

In this exercise, we will enable in-place editing in the Title component. In-place editing allows you to edit the component text without opening the Edit dialog box. For example, you will be able to change the title without opening the Edit dialog box.

1. Locate the `training-title` component.
2. Right-click the node, and click **Create** > **Create Node**.
3. Create a node with the following properties:

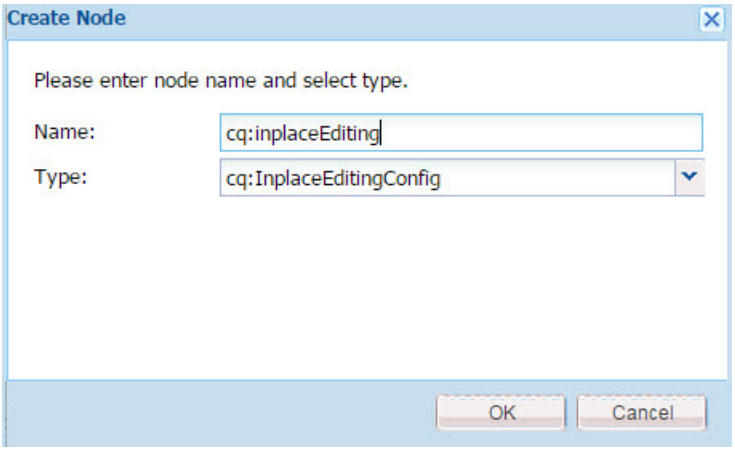
Property	Value
Name	cq:editConfig
Type	cq:EditConfig



4. Right-click the newly created node, and click **Create** > **Create Node**.

5. Create a node with the following properties:

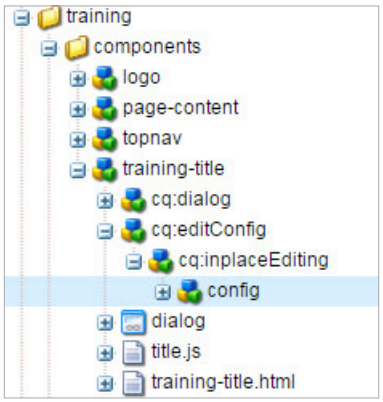
Property	Value
Name	cq:inplaceEditing
Type	cq:InplaceEditingConfig



6. Add the following properties to the node:

Name	Type	Value
active	Boolean	true
editorType	String	title

7. Copy the `/libs/wcm/foundation/components/title/cq:editConfig/cq:inplaceEditing/config` node and paste it under the node you just created.



8. Click **Save All**.
9. Open any one of the pages that you created.
10. Select the title component, and click once more. Notice that the cursor appears in the text box, allowing you to edit the title without opening the dialog box.

Chapter Nine

AEM Authoring Framework — Foundation Components, Internationalization, and Client Libraries

Overview

AEM provides an authoring framework, including foundation components, with which you can create your websites.

Objective

In this chapter, you will use the authoring framework you will build up on your website. You will specifically learn how to do the following:

- Work with the foundation components
- Internationalize the authoring environment
- Include client libraries

Work With the Foundation Components

Adobe provides you with a large number of components that you can use to build your website. They range from a simple Title component to the more complex paragraph system component.

The out-of-the-box components are located at:

- *libs/wcm/foundation/components* (the Sightly components)
- *libs/foundation/components* (the JSP components)

Use out-of-the-box components as much as possible rather than developing components from the beginning. You can also copy the out-of-the box components to your apps folder, and then customize them to your requirement.

Exercise 9.1
Include a Breadcrumb Foundation Component

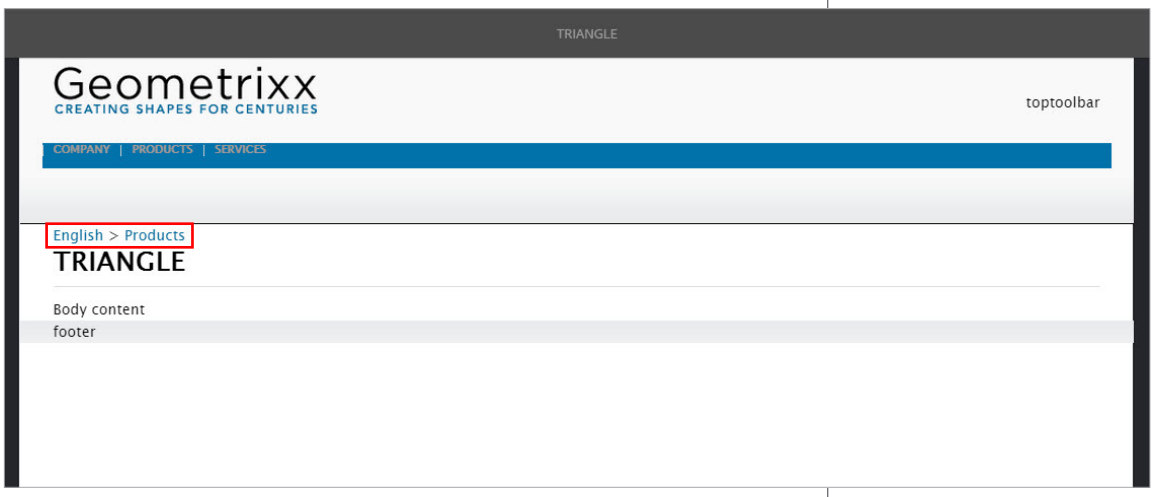
In this exercise, you will add a breadcrumb component to the template.

1. Go to CRXDE Lite, and locate **training** > **components** > **page-content**.
2. Open `body.html` by double-clicking it.
3. Update `body.html` as follows:

```
<div data-sly-include=”header.html”></div>
    <div class=”body_content container_16”>
        <div class=”content_title grid_16” data-sly-resource=”${‘breadcrumb’ @
        resourceType=’foundation/components/breadcrumb’}”></div>
       <div class=”content_title grid_16” data-sly-resource=”${‘title’ @
    resourceType=’training/components/training-title’}”></div>
        <div class=”content_title grid_16”>Body Content</div>
    </div
    <div data-sly-include=”footer.html”></div>
```

4. Click **Save All**.

5. Open the Triangle page you created before. Note that the previous pages in the hierarchy are displayed on the top as a hyperlink.



Example: Including a breadcrumb component using JSP

Use the following code if you are using JSP to create the breadcrumb.

```
<cq:include path="breadcrumb" resourceType="foundation/
components/breadcrumb"/>
```

Extra Credit: Foundation Breadcrumb Component

The following exercise tests your knowledge of creating design elements. The goal is to modify the foundation breadcrumb component so that an author with design privileges can set the path that determines the start of the breadcrumb trail.

1. As a best practice, we never modify anything in `/libs`. Create the foundation `/components` structure in `/apps` and copy the foundation breadcrumb component to the `apps/<your app>/components` folder.
2. Modify the code to allow the author to enter the start of the crumbs as a path instead of a level.
3. Modify the Design dialog box to allow the author to enter a path instead of a level. If you previously entered a number (level) with the same property name (for example, absParent), the breadcrumb component will show an error because it is looking for a string that is a path and encounters a number. To fix this, navigate to the design element (`/etc/designs/trainingDesign/jcr:content/breadcrumb`) and delete the `absParent` property. Also, remember that Geometrixx also uses the foundation breadcrumb component. Therefore, you will have to modify the Geometrixx design in the same fashion to get the Geometrixx pages to render properly.
4. Test your code. When the author enters values into the design dialog box, and saves them, these values are written into the design repository.

Include the Paragraph System

The paragraph system component is the main content area of a webpage. By adding the paragraph system component to the template, you structure and control the areas where content authors can add content.

The use of the AEM paragraph system component is a necessity when you want to have manageable and scalable page content, without requiring excessive coding and template creation. This foundation parsys component can be located at the path below:

- */libs/wcm/foundation/components/parsys* (the Sightly components)
- */libs/foundation/components/parsys* (the JSP components)

It enables content authors to dynamically add, delete, move, copy, and paste 'paragraphs' on a page and use the column control component to structure content in columns. In addition, you can decide what 'content' components are allowed to be used by specific instances of the parsys.

It is Adobe's strong opinion that the foundation `parsys` component should be used as often as possible, thus allowing for simple component configuration, as opposed to creating a large number of templates, which developers then have to maintain.

Exercise 9.2
Include the Paragraph System Component

In this exercise, you will extend the paragraph system component to the script. You will then enable some of the out-of-the-box components to use in your page.

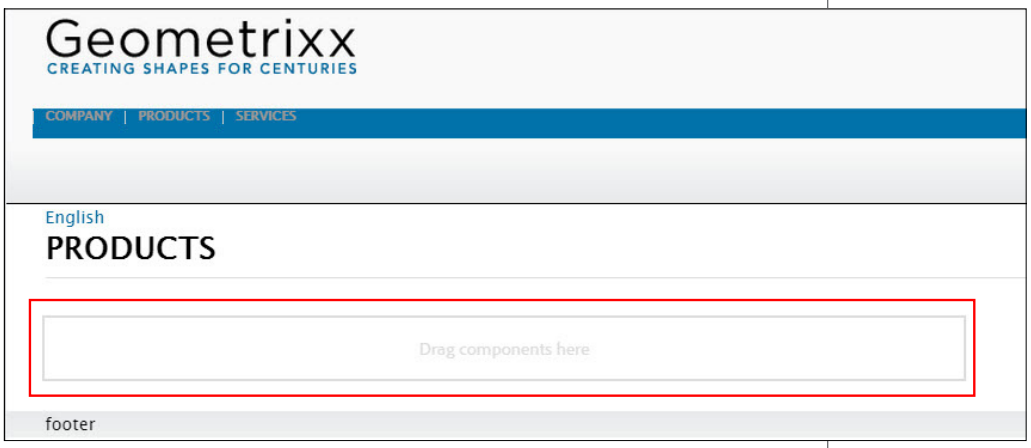
1. From the `page-content` component, open `body.html` by double-clicking it.
2. Select the following code:

```
<div class="content_title grid_16">Body Content</div>
```

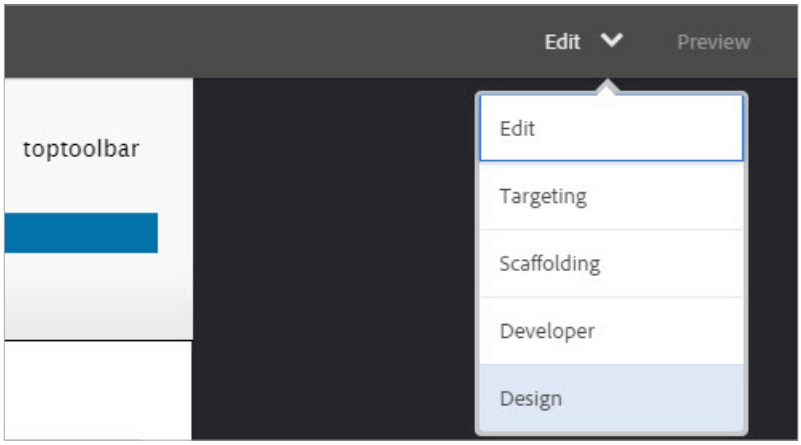
3. Replace it with the following code:

```
<div data-sly-include="header.html"></div>
    <div class="body_content container_16">
        <div class="content_title grid_16" data-sly-resource="${'breadcrumb' @
    resourceType='foundation/components/breadcrumb'}"></div>
       <div class="content_title grid_16" data-sly-resource="${'title' @
    resourceType='training/components/training-title'}"></div>
        <div class="content_main grid_12" data-sly-resource="${'content' @
    resourceType='wcm/foundation/components/parsys'}"></div>
    </div>
    <div data-sly-include="footer.html"></div>
```

4. Open one of the pages you created, for example, Products. It appears as shown below. Notice that the paragraph system component is displayed, where you can drag and drop another out-of-the-box component. However, you would not find any components listed yet. The following steps enable components to be listed in the Components tab.

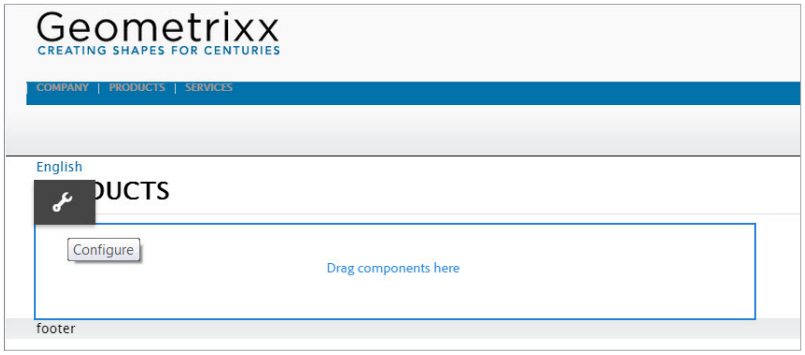


5. Click **Edit** on the upper-right corner of the screen, and select **Design**.



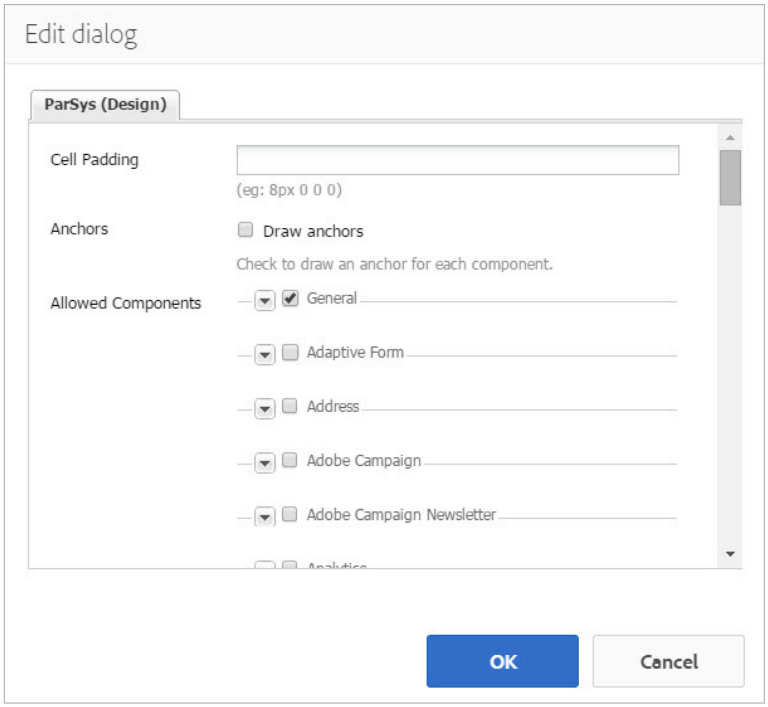
NOTE: You can also click the Parent icon to select the parsys component.

6. Select the new component. Ensure that you select the outer layer of the component; that is, when you get the Configure icon rather than the Parent icon.
7. Click the **Configure** icon.



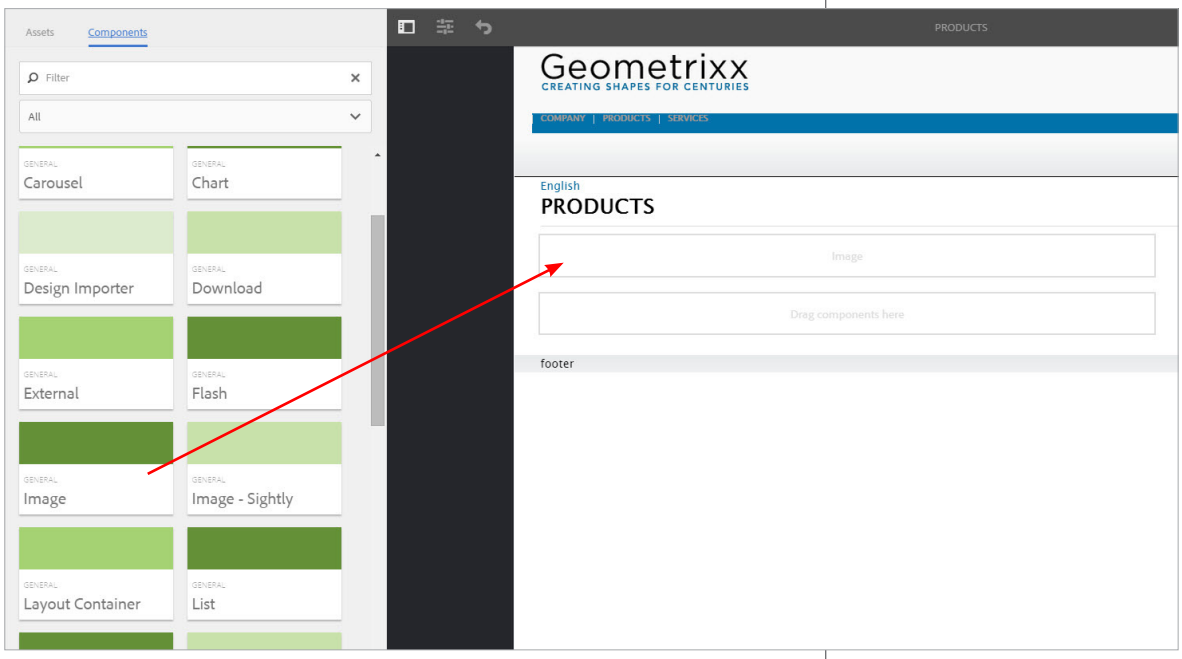
The **Edit** dialog box appears.

8. Select the **General** section to include all the components within General tab, and click **OK**.



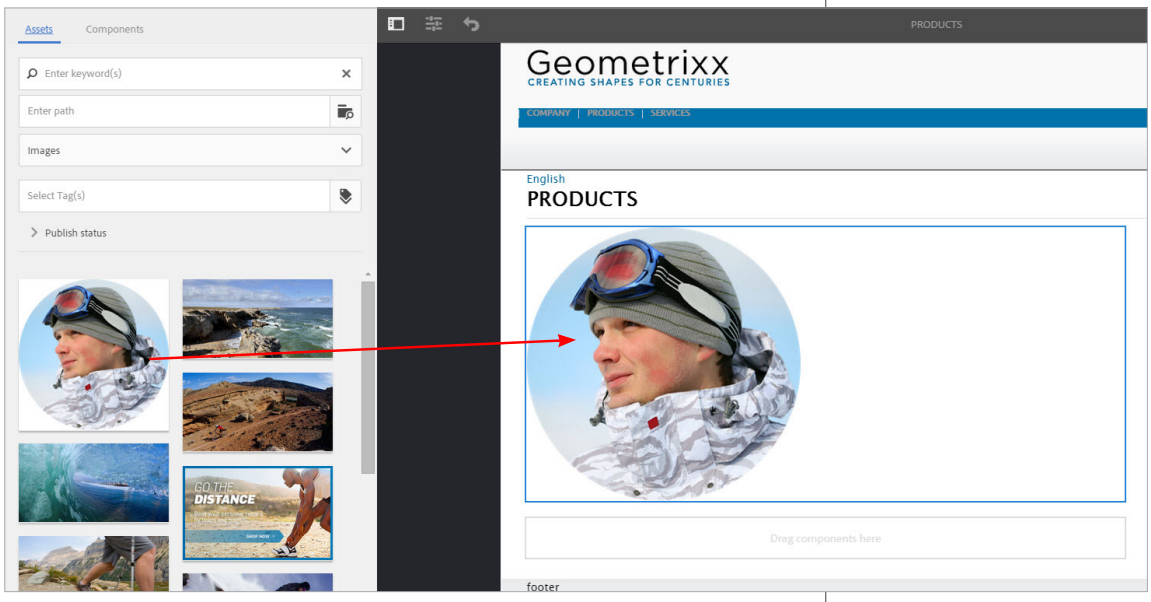
9. Go back to the edit mode by clicking **Edit** on the upper right corner of the screen.
10. Click the toggle button on the left, and select the **Components** tab to view the list of components.

11. From the list of components, drag and drop the **Image** component to the paragraph system component.



12. Select the **Assets** tab.

13. Drag an image from the list of assets, and drop it onto the Image component.



14. Refresh the page to see the added image.

Exercise 9.3
Use the Toolbar Component

One of the requirements in the wireframe is to add the following links:

- Header
  - Contacts
  - Login
- Footer
  - Feedback
  - Search

You will use the Toolbar component to create these links in your template. The Toolbar component allows you to list the child nodes present under the node named Toolbar. You can also set a property named `cq:toolbars` and assign a value to filter the nodes based on your requirements.

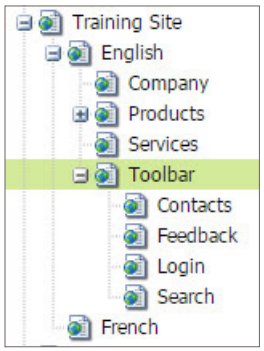
1. Log in to CRXDE Lite. Locate the `/apps/training/components` node.
2. Go to the `page-content` folder, and open `footer.html`.
3. Replace the placeholder for toolbar as shown below:

```
<div class="body_footer footer container_16">
        <div class="header_toptool grid_6 toptoolbar toolbar cq-element-toptoolbar" data-sly-
    resource="${'bottomToolbar' @ resourceType='foundation/components/toolbar'}"></div>
</div>
```

4. Open the `header.html` page. Replace the code for the top toolbar placeholder as shown below:

```
<div class="body_header header container_16">
    <div class="header_logo grid_8" data-sly-resource="${'logo' @ resourceType='training/
    components/logo'}"></div>
           <div class="header_topnav grid_8 search_area">
               <div class="header_toptool toptoolbar toolbar cq-element-toptoolbar" data-sly-
    resource="${'topToolbar' @ resourceType='foundation/components/toolbar'}"></div>
           </div>
    <div class="content_topnav toptoolbar toolbar" data-sly-resource="${'topnav' @
    resourceType='training/components/topnav'}"></div>
</div>
```

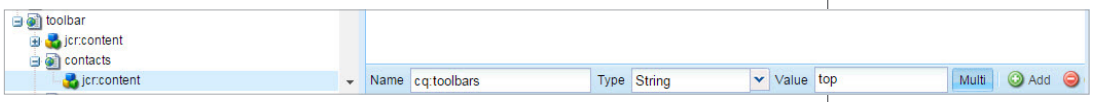
5. Open the site admin, and create a Toolbar page on the English page. Ensure that the page name is toolbar.
6. Under Toolbar, create the following pages:
   a. Contacts
   b. Feedback
   c. Login
   d. Search



7. In CRXDE Lite, locate `/content/training-site/en/toolbar/contacts`.

8. Select the `jcr:content` node and add the following multi properties:

Property	Value
Name	cq:toolbars
Type	String
Value	top



9. Ensure that **Multi** is selected, and then click **Add**. In the **Edit** dialog box, click **OK**.

10. Select the `jcr:content` node of `login` and add the following properties:

Property	Value
Name	cq:toolbars
Type	String
Value	top

11. Ensure that **Multi** is selected, and then click **Add**. In the **Edit** dialog box, click **OK**.

12. Select the `jcr:content` node of `feedback` and add the following properties:

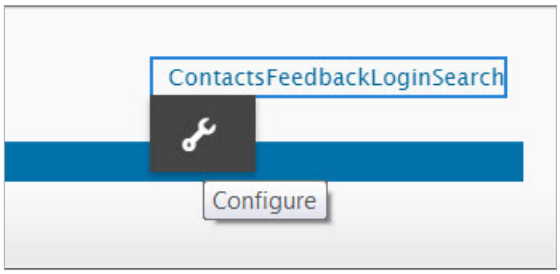
Property	Value
Name	cq:toolbars
Type	String
Value	bottom

13. Ensure that **Multi** is selected, and then click **Add**. In the **Edit** dialog box, click **OK**.

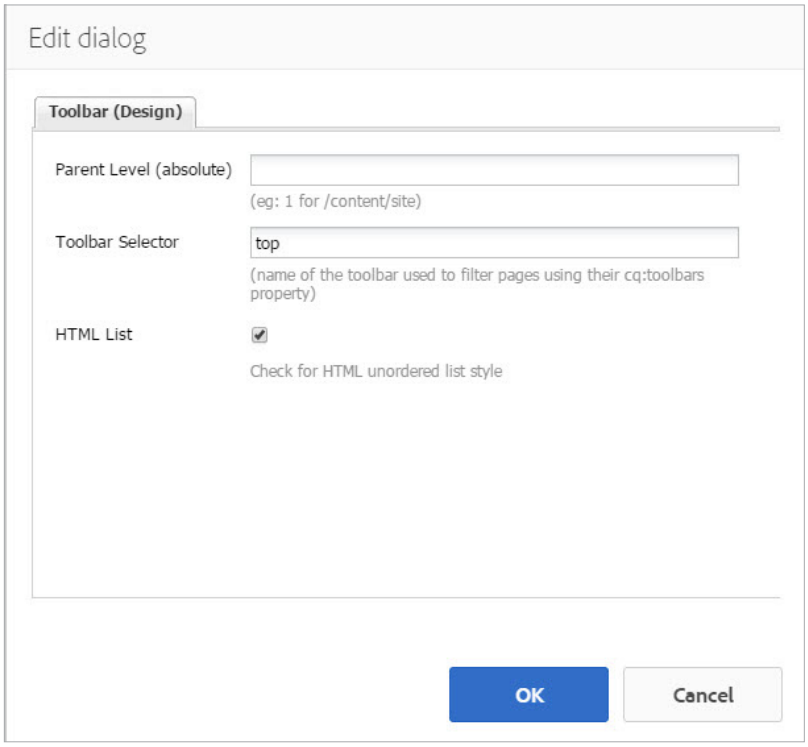
14. Select the `jcr:content` node of `search` and add the following properties:

Property	Value
Name	cq:toolbars
Type	String
Value	bottom

15. Ensure that **Multi** is selected, and then click **Add**. In the **Edit** dialog box, click **OK**.
16. Save your changes.
17. Open the website page you created. Go to the design mode.
18. Select the top toolbar, and click the **Configure** icon.



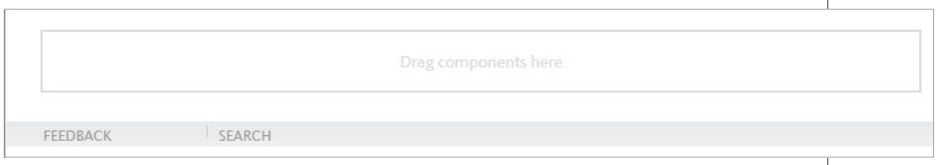
19. In the **Edit Component** dialog box, for the **Toolbar Selector**, type `top`. Select the **HTML List** check box. Click **OK**, and refresh the page.



Observe that only the pages with the *top* property appear on the top.

20. Select the bottom toolbar, and click the **Configure** icon.

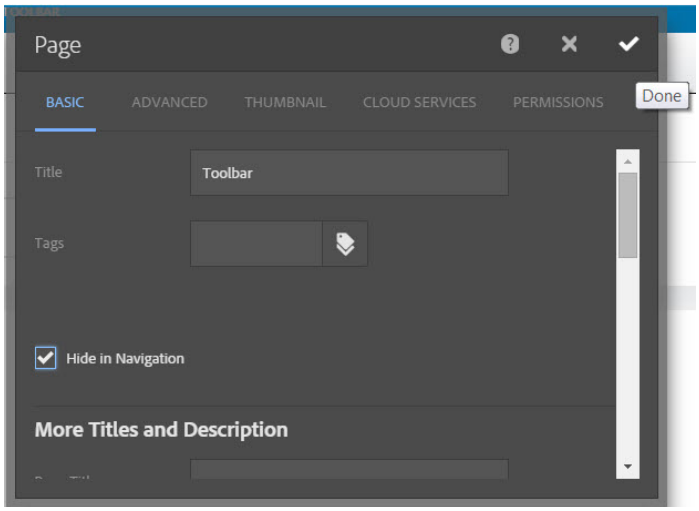
21. In the **Toolbar selector**, type `bottom`. Select the **HTML List** check box. Click **OK**, and refresh the page.



Observe that only the pages with the `bottom` property appear on the bottom. Also, notice that the top navigation bar now displays a Toolbar link that is not needed.

22. To remove the Toolbar link from the top navigation bar, open the Toolbar webpage. Click the **Page Information** icon located in the upper left corner of the page, and select **Open Properties**.

23. In the **Page** dialog box, select the **Hide in Navigation** check box, and click the **Done** icon.



Observe that the refreshed page does not contain the Toolbar link.

Exercise 9.4
Include the iParsys Component

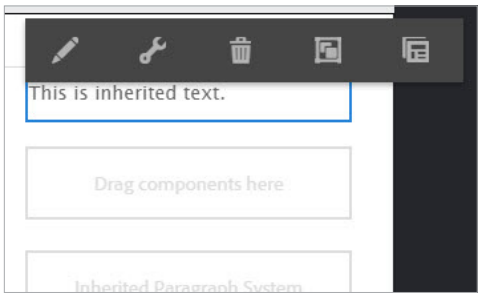
The iParsys (Inherited paragraph system) component is a paragraph system component that allows you to inherit the paragraphs from the parent page. This is used to create sidebars where content in the parent and child page would be the same.

1. From the `page-content` component, open `body.html`.
2. Add the following code:

```
<div data-sly-include="header.html"></div>
       <div class="body_content container_16">
           <div class="content_title grid_16" data-sly-resource="${'breadcrumb' @
    resourceType='foundation/components/breadcrumb'}"></div>
           <div class="content_title grid_16" data-sly-resource="${'title' @
    resourceType='training/components/training-title'}"></div>
           <div class="content_main grid_12" data-sly-resource="${'content' @
    resourceType='wcm/foundation/components/parsys'}"></div>
           <div class="grid_4 right_container">
               <div data-sly-resource="${'sidebar' @ resourceType='wcm/foundation/components/
    iparsys'}"></div>
           </div>
       </div>
    <div data-sly-include="footer.html"></div>
```

3. Open the root page, that is, Training Site.
4. Go to the Design mode. Select the Inherited Paragraph System component, and click **Configure**.
5. Ensure that the **General** tab is selected to enable all components under it. If no components are enabled, then you cannot drag and drop components into the Inherited Paragraph System.
6. Go to the Edit mode, drag and drop the `Text` component to iParsys.

7. Add some text.



8. Open the English page of the Training site. You can see that the text you added appears on the child page.

Internationalize the Authoring Interface

The AEM Authoring interface ships in seven languages, allowing authors to enter and manage content in their language of choice. When you create your own components and dialog boxes, as we learned earlier, you can provide those dialog boxes in multiple languages as well.

Internationalization of the authoring interface allows you to provide dynamic messaging based on the authors' language preference. Internationalization message bundles are stored in the repository under nodes (nodeType sling:Folder) named **i18n**.

The children nodes (nodeType `sling:Folder + mixin mix:language`) of the *i18n* node represent languages and are named using the ISO code of the languages that are supported, for example, en, de, and fr. Below these language nodes are the message bundle nodes (nodeType sling:MessageEntry), which will contain a simple key message pair.

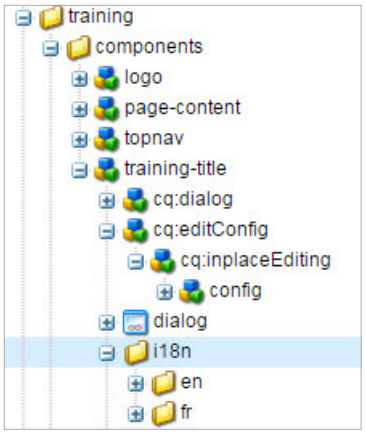
The location of the i18n node within the repository determines the scope of the message bundles. If located in a project directory, for example, /apps/training, it should contain only messages related to the current project. However, if located in a component hierarchy, it should contain only component-specific translations. Globally used messages should be place in /apps/i18n.

Exercise 9.5
Internationalize the Title Component's GUI

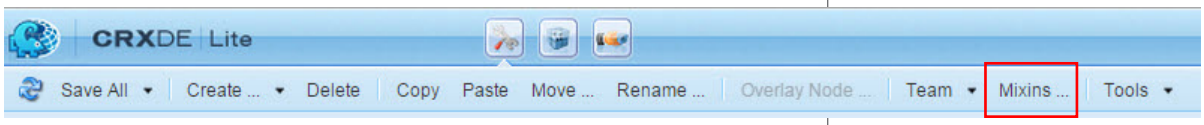
In this exercise, you will internationalize the GUI instruction provided in the Title component. You will have noticed the text that comes in the Edit dialog box, '*Enter the title here*'.

1. Using CRXDE Lite, locate /`apps/training`.
2. Go to the `components` folder, and select `training-title`. Right-click the component, and click **Create** > **Create Folder**.
3. Create a folder named `i18n`.

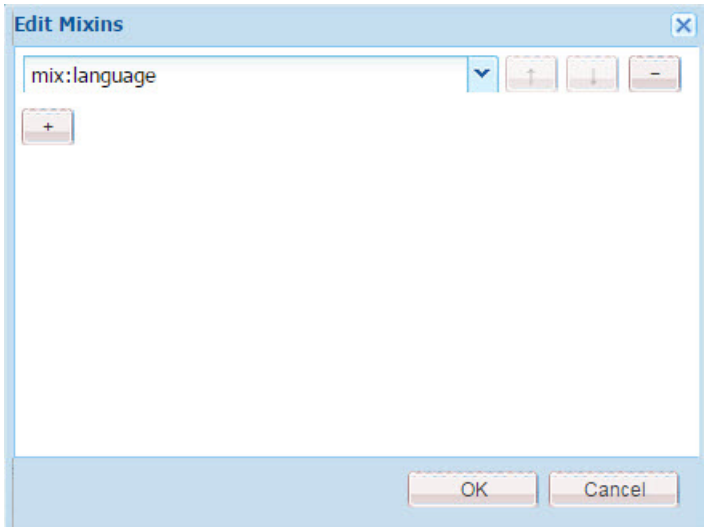
4. Create two folders underneath `en` and `fr`, and click **Save All**.



5. Select the `en` folder and click the **Mixins** button displayed on the top bar.



6. Click the + symbol in the **Edit Mixins** dialog box and select `mix:language`. Click **OK**.



7. Add the following property to the `en` node and click **Save All**.

Name	Type	Value
jcr:language	String	en

8. Repeat steps 5–7 for the `fr` node. Ensure that you set the `jcr:language` property to `fr`.

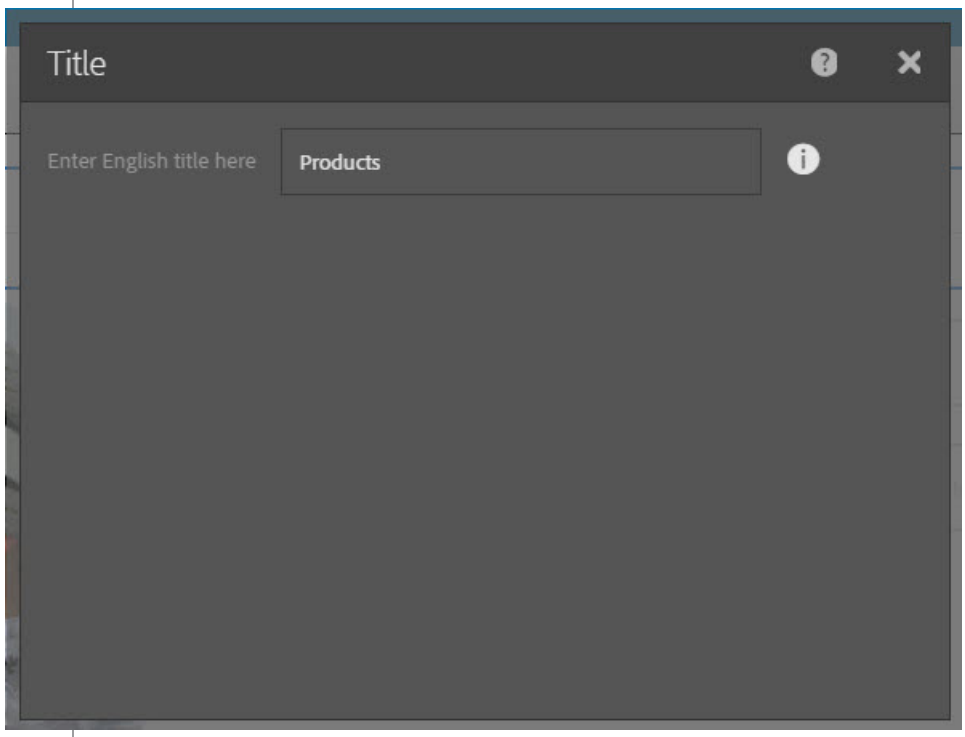
9. Select the `en` node. Right-click and create a node with the following details:

Property	Value
Name	title
Type	sling:MessageEntry

10. Add the following properties to the `title` node:

Name	Type	Value
sling:key	String	i18n-title
sling:message	String	Enter English title here

11. Repeat steps 9 and 10 to the `fr` node. Ensure that the value of the `sling:message` property is '`Enter French title here`'.
12. Click **Save All**.
13. To internationalize the touch-optimized UI, go to the `cq:dialog` node. Navigate to `/training-title/cq:dialog/content/items/columns/items/title`.
14. Change the value of the **fieldLabel** property to `i18n-title`.
15. To internationalize the Classic UI, go to the `dialog` node. Navigate to `/training-title/dialog/items/items/tab1/items/title`.
16. Change the value of the **fieldLabel** property to `i18n-title`, and click **Save All**.
17. Open one of the pages that you created, for example the Product page.
18. Double-click the `Title` component. Note that the label for the text box is changed.

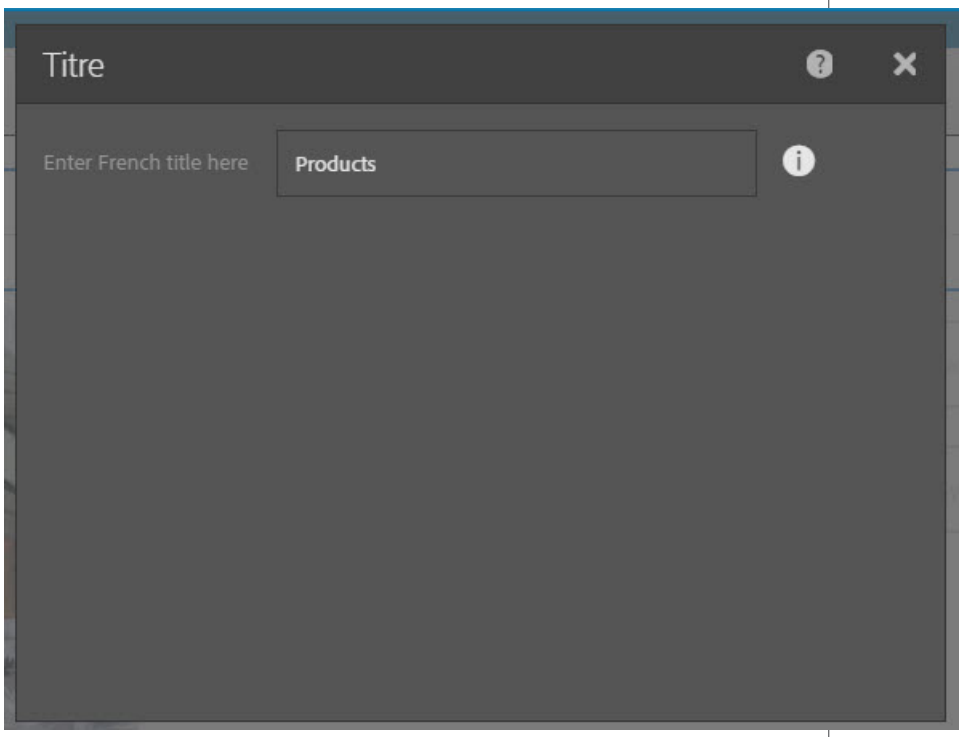


NOTE: Translations are maintained in an AEM dictionary, and can be seen in http://localhost:4502/libs/cq/i18n/translator.html

19. To view the label in French, in the Sites console (touch-optimized UI), click the **User Settings** icon located in the upper right corner of the screen, and select **User Preferences**.

20. Select the language as **French**, and click **OK**.

21. Now open one of the training site pages and double-click the title. You can see the change in the label as shown below:



Add Client Libraries

In today's modern web development, comprehensive JavaScript libraries, with HTML and CSS, are responsible for some exciting web experiences.

Managing these client-side assets can become quite cumbersome, especially because AEM allows authors to select components and templates at their own convenience. Developers cannot really plan when and where client-side assets will be used.

Another challenge is that many components and templates require client-side assets.

Client or HTML Libraries

AEM has introduced a very interesting concept: client libraries, also known as 'clientlibs'. Client libraries are 'folders' (nodes of node-type cq:ClientLibraryFolder) that contain the client-side assets, CSS and JS files, and required resources, for example, images or fonts. There can be unlimited client library folders. The folder content can be loaded individually and at any given time.

AEM has a tool called Dumplibs that lists all the defined client libraries and its properties. You can access it from the following link:
http://localhost:4502/libs/granite/ui/content/dumplibs.html

Client Library Conventions

Create client libraries under /etc/clientlibs or within the component folder. A client library 'folder' is created as a node with the Node type cq:ClientLibraryFolder, with the following properties:

- jcr:primaryType: cq:ClientLibraryFolder
- categories: An array of names to identify the client library
- dependencies: An array of categories (dependent client libraries)
- embed: An array of client libraries that will be included

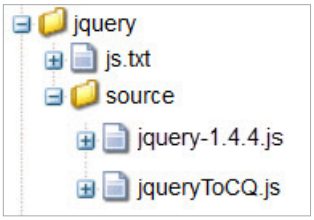
Create sub-folders for the CSS and JS files, for example, /scripts or /styles. Create a css.txt and/or a js.txt file and add the .css and .js files that will be minified and loaded by templates or components. If the assets are in a sub-folder, the .txt file must start with '#base=sub-folder'. Resources, like images or fonts, which are used by JS or CSS files are stored in the client library folder.

Client libraries are loaded with the <cq:includeClientLib> tag. This tag includes a client library, which can be a JS, a CSS, or a Theme library. For multiple inclusions of different types, for example, JS and CSS, this tag needs to be used multiple times in the JSP.

Client libraries can be loaded programmatically with the com.day.cq.widget. HtmlLibraryManager service interface.

Examples of Client Libraries

An example to define jQuery:



Properties of the jQuery folder:

jcr:primaryType = cq:ClientLibraryFolder
categories = cq.jquery (dot-notated names are allowed)

The js.txt file contains:

```
#base=source
jquery-1.4.4.js
jqueryToCQ.js
```

The file starts with defining the folder where the JS files can be found and then the JS files are listed that will be loaded.

This client library can now be used as a 'dependent' one or it can be 'embedded' in another client library.

- Dependencies: The libraries of categories listed in 'dependencies' have to be already included; otherwise, the current library will not be included.
- Embed: The libraries of categories listed in 'embed' will be included in the HTML page as well. If the libraries have already been included, they are ignored.

Include Client Libraries

AEM provides a custom JSP tag, which makes it easy to include client libraries or parts of them: `<cq:includeClientLib>`.

In Sightly, use `data-sly-call` tag to include client libraries. The purpose of the `<cq:includeClientLib>` tag is to include JS and CSS assets in the HTML page.

The parameters are:

Categories: A list of comma-separated client library categories. This will include all JavaScript and CSS libraries for the given categories. The theme name is extracted from the request.

js: A list of comma-separated client library categories. This will include all JavaScript libraries from the listed categories.

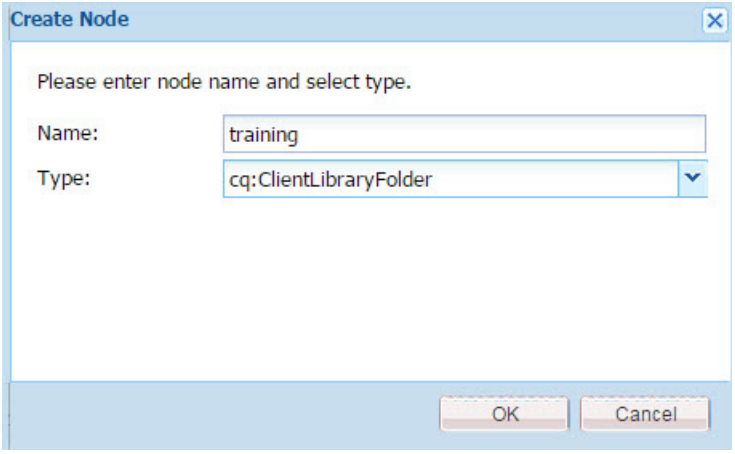
css: A list of comma-separated client library categories. This will include all CSS libraries from the listed categories.

theme: A list of comma-separated client library categories. This will include all theme-related libraries (CSS and JS) for the listed categories. The theme name is extracted from the request.

themed: A flag that indicates if only themed or non-themed libraries should be included. If omitted, both sets are included. This only applies to pure JS or CSS includes (not for categories or theme includes).

Exercise 9.6
Include a JavaScript Function From Client Libraries

1. In CRXDE, create a new client library node, `training`, in `/etc/clientlibs`.



2. Add the following property to the node. Use the **Multi** option in CRXDE Lite to add a String array property.

Name	Type	Value
categories	String[]	training.edit

3. Under `training`, create a folder named `scripts`.
4. Inside `scripts`, create a JavaScript file named `trainingEdit.js`.

5. Add the following code to `trainingEdit.js`.

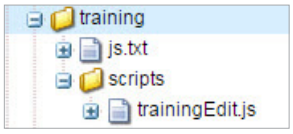
```
/*Training JavaScript*/
alert("This is from ClientLib");
```

6. In the `training` folder you created in step 1, create a file, `js.txt`.

7. Add the following in `js.txt`:

```
#base=scripts
trainingEdit.js
```

8. The following must be the directory structure:



9. The client libraries are now ready to be included in an HTML page. Add the client library to the website. Add the following code at the top of the `header.html` file:

```
<meta data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html" data-sly-
    call="${clientlib.all @ categories='training.edit'}" data-sly-unwrap>
</meta>
```

10. Refresh any of the pages you created. The alert message appears in the screen as shown below:

11. Remove the following code that you added in `header.html` after you test this. It is provided for the demo purpose. Addition of this script makes navigation in the site difficult.

```
<meta data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html" data-sly-
    call="${clientlib.all @ categories='training.edit'}" data-sly-unwrap>
</meta>
```

Example: Adding a Client Library in JSP

```
<%@include file="/libs/foundation/global.jsp" %>
<html>
    <head>
        <title> Training </title>
        <cq:include script="/libs/wcm/core/components/init/init.jsp"/>
        <cq:includeClientLib categories="training"/>
        <c:if test="<%= WCMMode.fromRequest(slingRequest) == WCMMode.EDIT%>">
            <cq:includeClientLib categories="training.edit"/>
        </c:if>
    <head>
    <body>
        <p id="publishData">Publish-mode Data</p>
        <p id="authorData">Publish-mode Data</p>
    </body>
</html>
```

Mobile Websites

Overview

Users may want to view your websites in handheld devices. Responsive Design for Mobile pages allows you to create an optimal viewing experience across various mobile devices.

Objective

In this chapter, you will learn how to do the following:

- Use Responsive Design for Web pages
- Create mobile components
- Create mobile websites using MSM
- Work with emulators

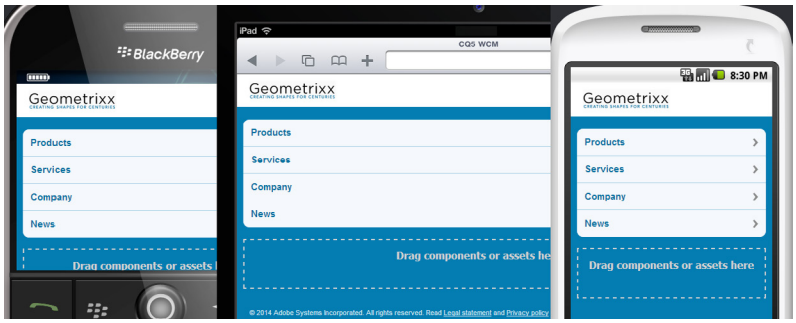
Responsive Design

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.

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.

Exercise 10.1
Preview the Site With Various Devices

In this exercise, you will use styles to get your site to adapt to various screen sizes. Following are the styles you will use:

- responsive-1200px.css: Styles for all media that are 1,200 pixels wide or more
- responsive-980px-1199px.css: Styles for media that are between 980 pixels and 1,199 pixels wide
- responsive-768px-979px.css: Styles for media that are between 768 pixels and 979 pixels wide
- responsive-767px-max.css: Styles for all media that are less than 768 pixels wide
- responsive-480px.css: Styles for all media that are less than 481 pixels wide

You will then add the Device Group to the page so that the page can be previewed in specific devices.

1. Include the client library folder to your template. Use the client library files created for Geometrixx Media. Open `header.html` and include the following code at the top of the page:

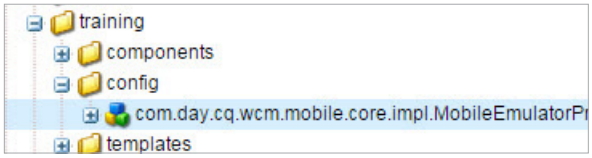
```
<meta data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html" data-sly-
    call="${clientlib.all @ categories='apps.geometrixx-media'}" data-sly-unwrap>
</meta>
```

2. Include the device list in the sidekick. Open `header.html` and include the following code immediately after the code you added in step 1:

```
<head>
    <div data-sly-include="/libs/wcm/mobile/components/simulator/simulator.jsp">   </div>
</head>
```

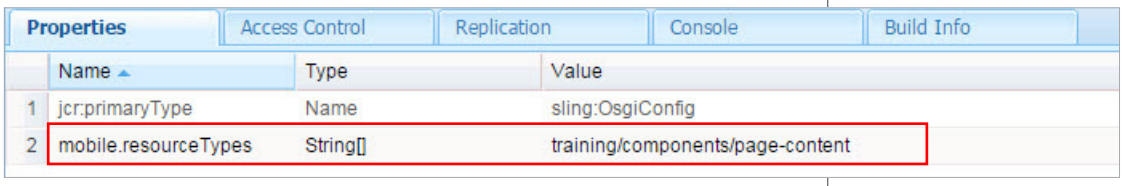
3. To enable the device simulator to support your pages, register your page components with the `MobileEmulatorProvider` factory service and define the `mobile.resourceTypes` property. Create a folder named `config` in the `training` folder. In the `config` folder, create a node with the following details:

Name	Type
com.day.cq.wcm.mobile.core.impl. MobileEmulatorProvider-trainingconfigconfig	sling:OsgiConfig



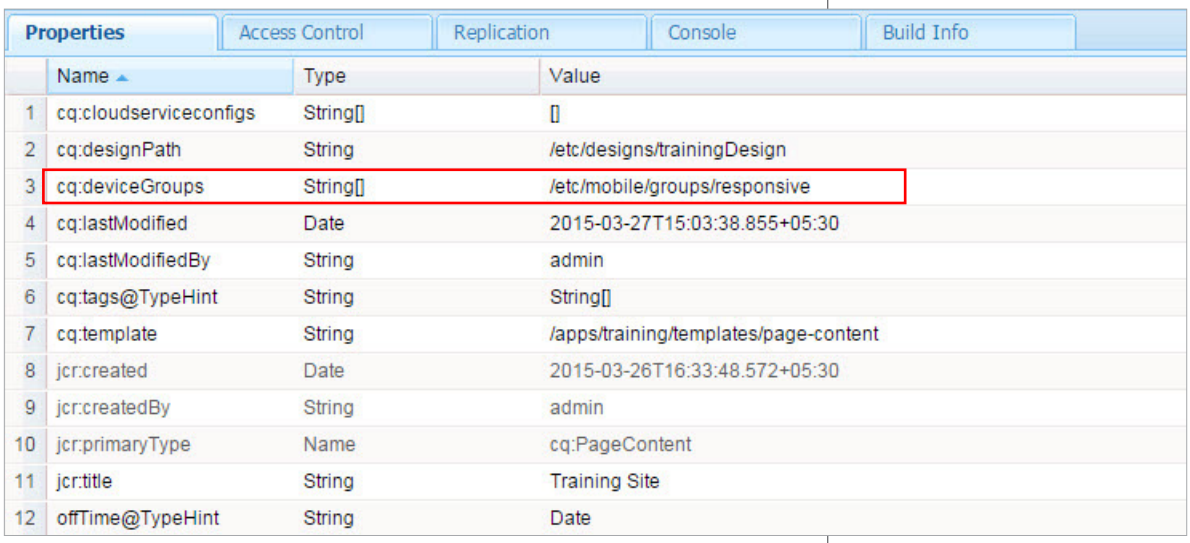
4. Add the following property to the node, using Multi:

Name	Type	Value
mobile.resourceTypes	String[]	training/components/page-content

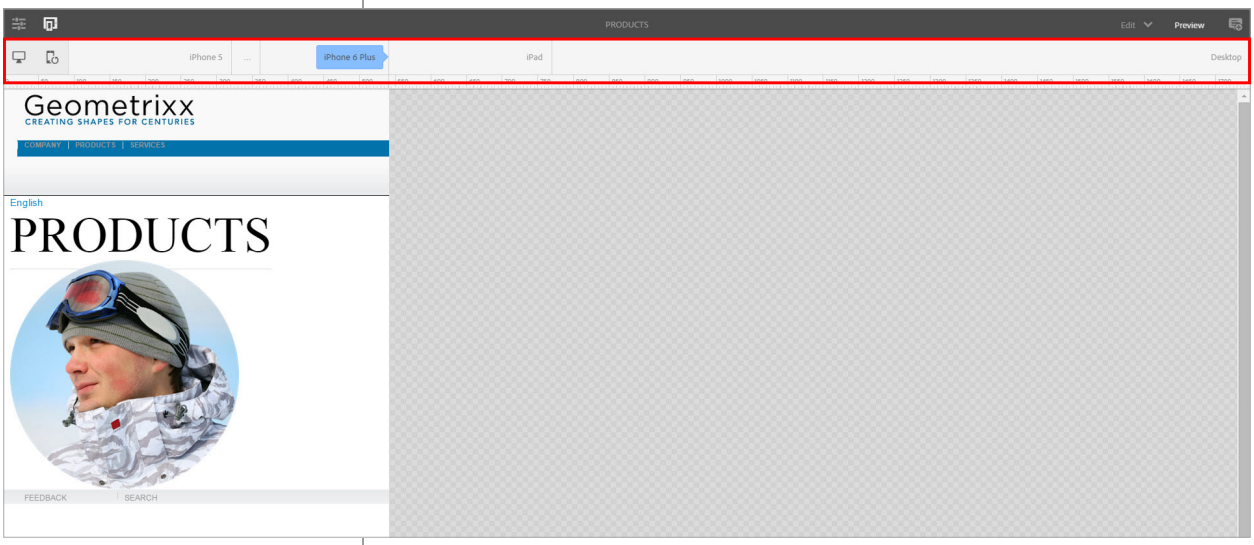


5. Specify the device groups that appear in the devices list. Go to the `/content/training-site/jcr:content` node. Add the following property:

Name	Type	Value
cq:deviceGroups	String[]	/etc/mobile/groups/responsive



6. Open one of the pages; for example, Products page.
7. Click **Preview** to go to the Preview mode, and click the  **(Emulator)** icon located in the upper-left corner of the screen.
8. 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.

9. The above exercise is created for testing purpose. Now, exclude the CSS from your template by removing the following code from the `header.html` file. (This CSS can cause issues to your current CSS.)

```
<meta data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html" data-sly-
    call="${clientlib.all @ categories='apps.geometrixx-media'}" data-sly-unwrap></meta>
<head>
    <div data-sly-include="/libs/wcm/mobile/components/simulator/simulator.jsp"></div>
</head>
```

Mobile Components

Creating mobile components is similar to creating regular components. The only difference between a regular component and a mobile component is that the component needs to be aware if the emulator that is used to display the page is able to handle the kind of content that it needs to display. For that, you have to use classes that are useful every time you develop mobile components.

- com.day.cq.wcm.mobile.core.MobileUtil
- com.day.cq.wcm.mobile.api.device.capability.DeviceCapability

The class MobileUtil provides you with the following methods:

static DeviceGroup	getDefaultDeviceGroup(Page page) Finds the device groups currently assigned to the given page or its closest ancestor
static String	getDeviceGroupSelector(SlingHttpServletRequest request) Returns the last selector found in the request URL
static Boolean	hasCapability(SlingHttpServletRequest request, DeviceCapability capability) Retrieves the DeviceGroup from the current request and checks whether the group offers the given capability
static Boolean	isDeviceGroup(Page page) Checks whether the given Resource represents a WCM Device Group
static Boolean	isDeviceGroup(Resource resource) Checks whether the given Resource represents a WCM Mobile Device Group
static Boolean	isMobileRequest(SlingHttpServletRequest r) True if the request's user agent is a mobile device
static Boolean	isMobileResource(Resource r) True if the given Resource is to be handled as a mobile resource
static Boolean	isNoMatch(String[] selectors)

The most important method for you is the *hasCapability* method, which indicates whether you should render a piece of content depending on the capability of the emulator to handle the kind of content.

The interface DeviceCapability on the other hand has the following fields:

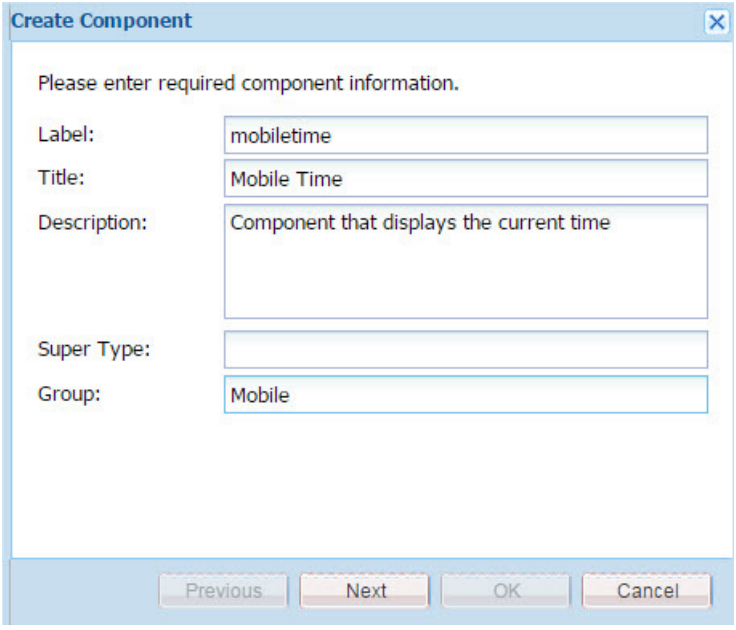
Field Summary	
staticDeviceCapability	CAPABILITY_CSS
staticDeviceCapability	CAPABILITY_DEVICEROTATION
staticDeviceCapability	CAPABILITY_IMAGES
staticDeviceCapability	CAPABILITY_JAVASCRIPT

It is easy then to use both elements to create a component that will render the appropriate content based on the device capabilities of the mobile device (and the emulator) used to display the page.

Exercise 10.2
Create a Mobile Time Component

1. Right-click `/apps/training/components`. Click **Create** > **Create Component**.
2. Enter the properties of the node as below:

Property	Value
Label	mobiletime
Title	Mobile Time
Description	Component that displays the current time
Group	Mobile



3. Click **Next**, and then click **Next** once more.
4. In the **Allowed Parents** section, click the **+** symbol and type `*/*parsys`.
5. Click **Next**, and then click **OK.** Your component is now created.
6. Open the `mobiletime.jsp` file and insert the following code:

```
<%@ page import="com.day.cq.wcm.mobile.api.device.capability.DeviceCapability,
com.day.cq.wcm.mobile.core.MobileUtil" %> <%
%><%@include file="/libs/foundation/global.jsp"%>
    <%
    // only show the times if the device supports javascript
    if (MobileUtil.hasCapability(slingRequest, DeviceCapability.CAPABILITY_
    JAVASCRIPT)) {
    %>
        <script type="text/javascript">
           function startTime() {
           var today=new Date();
           var h=today.getHours();
           var m=today.getMinutes();
           var s=today.getSeconds();
           // add a zero in front of numbers<10     m=checkTime(m);
           s=checkTime(s);
           document.getElementById('timing').innerHTML=h+":"+m+":"+s;      t=setTime
    out('startTime()',500);
           }
        function checkTime(i) {
               if (i<10) {
               i="0" + i;
           }
        return i;
          }
        </script>
        <div id="timing" style="color:yellow; font-style:bold"></div>
        <script type="text/javascript">
        startTime();
          </script>
          <% } else { %>
       <p>Your device does not support javascript</p>
    <% } %>
```

7. Save your changes.
8. Select your component, and create an empty dialog box for it. You need this so that it appears in the list of components available to the paragraph system.
9. Open the English page of Geometrixx Mobile Demo Site in Classic UI. Go into Design mode, and add your mobiletime component to the design. Return to Edit mode.
10. From the sidekick, drag the Mobile Time component into the paragraph system of the **Geometrixx Mobile Demo Site** > **English** page. See the result in the devices that support JavaScript.

Devices that do not support JavaScript display a message indicating the same.

If the display does not refresh automatically after adding this component and you do not immediately see the effect, manually refresh the page.

To correct this, you can add the following configuration on the `cq:editConfig` node under the component:

```
cq:editConfig
          cq:listeners (type cq:EditlistenersConfig)
                property afterInsert = value REFRESH_PARENT
```

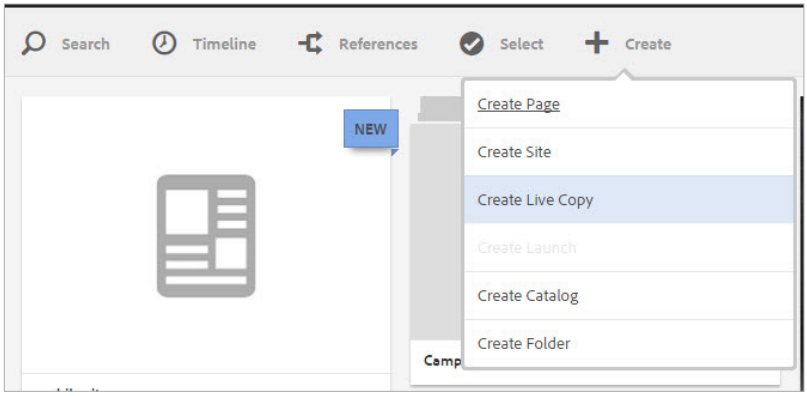
Creation of a Mobile Website Using MSM

The Geometrixx Outdoors Mobile Site has a setup that enables the creation of a special live copy from the Geometrixx Outdoors Site (the standard, non-mobile one). When the live copy is created and the standard site is rolled out, the content of the mobile site is synced with the standard site, and the rendering components are transformed into mobile ones (the sling:resourceType properties are rewritten) to best suit the rendering on mobile devices.

Exercise 10.3
Create a Mobile Website

In this exercise, you will create a mobile website. You can use Multi Site Manager (MSM) to create a mobile live copy from a standard site. The standard site is automatically transformed into a mobile site. The mobile site has all the features of standard mobile sites (for example, edition within an emulator) and can be managed in sync with the standard site.

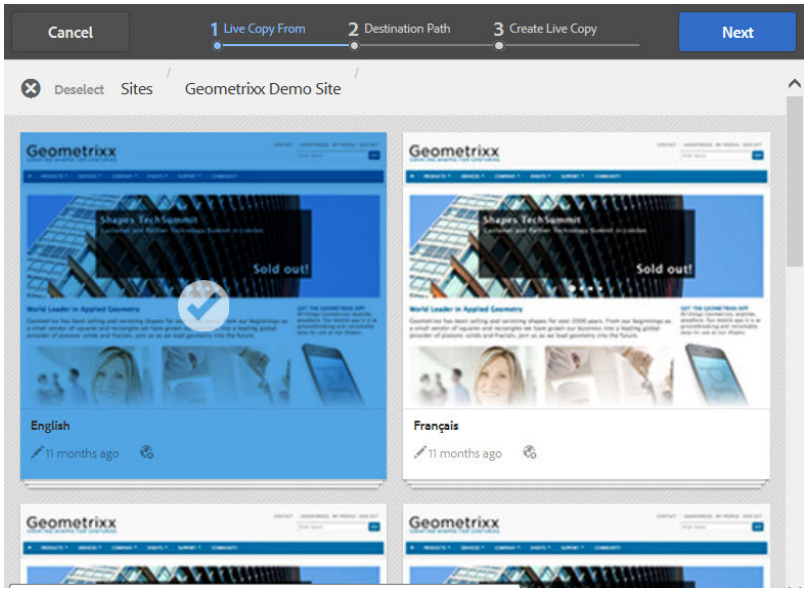
1. Log in to the touch-optimized UI. From the rail, select **Sites**.
   http://localhost:4502/
2. Click **Create**, and then click **Create Page**.
3. Select the template you already created and click **Next**.
4. Provide a name (*mobile-site*) and title (*Mobile Site*), and click **Create**. In the confirmation dialog box, click **Done**.
5. Go back to **Sites**, click **Create**, and click **Create Live Copy**.



6. Click the Geometrixx Demo Site.

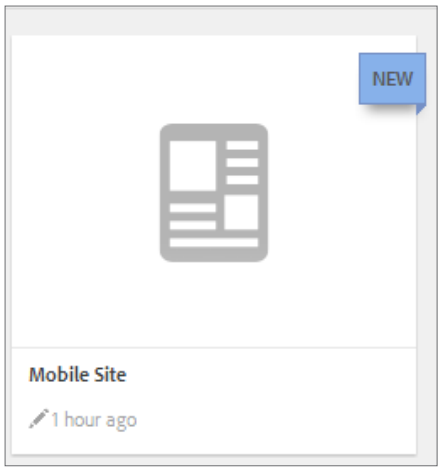
7. Click **Select** in the upper-left corner, to enter into selection mode.

8. Select **English** and click **Next**.

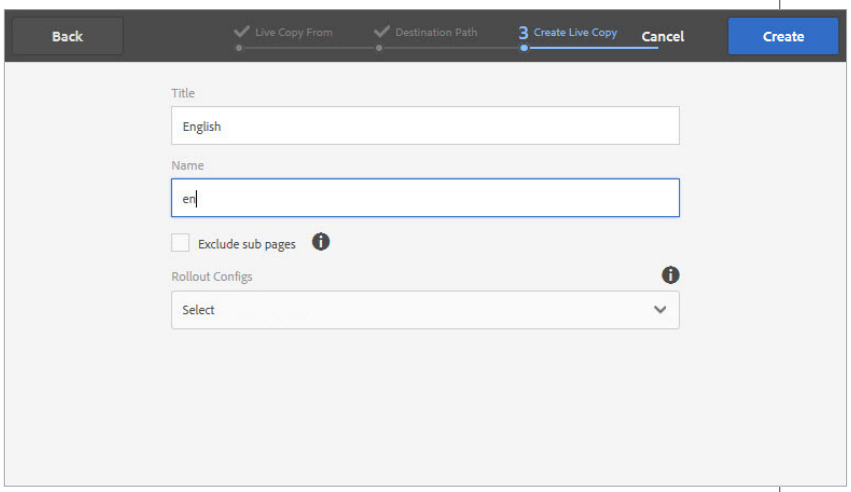


9. From the upper-left corner, click **Sites**.

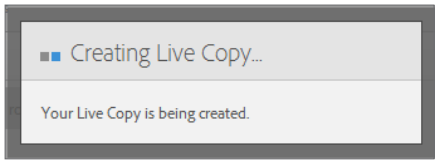
10. Select **Mobile Site** that you created, and click **Next**.



11. Enter the title as `English`.
12. Enter the name as `en`.
13. From the drop-down list, select the following configurations:
    a. Standard rollout config
    b. Geometrixx Mobile

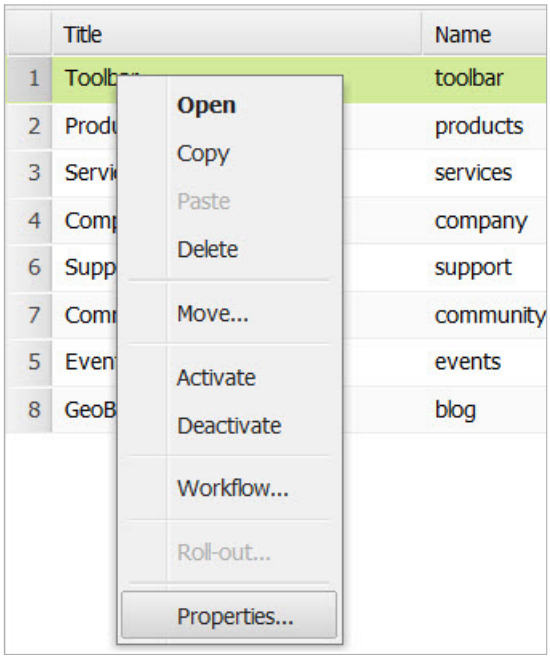


14. Click **Create** to create the live copy.

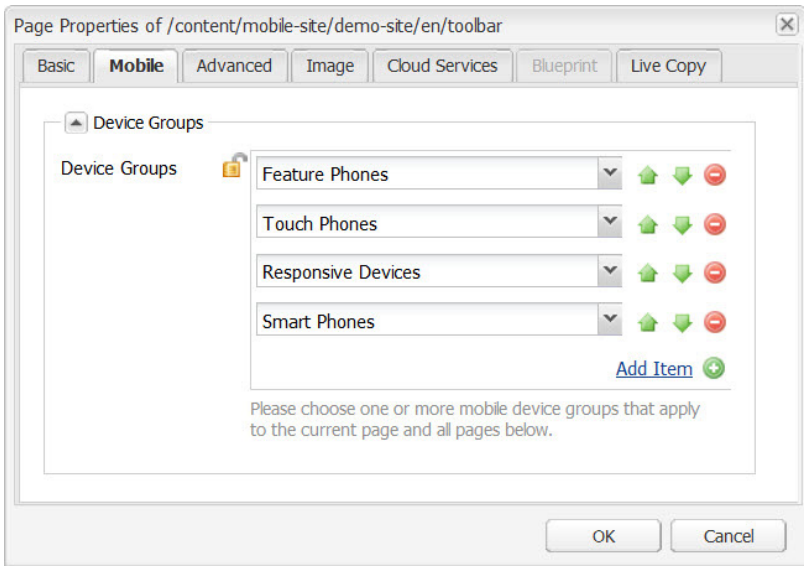


15. Wait for the live copy to be created.

16. After the live copy is created, go to the website console in the Classic UI.
17. Select **Mobile Site**.
18. In the left panel, select the English node. Its contents are listed on the right. Right-click on any of the pages, and select **Properties**.



19. Go to the **Mobile** tab and click the lock symbol to cancel inheritance. Then, add one or more mobile devices that apply to the current page and all the pages below. Click **OK**.



Mobile Emulators

AEM provides several device emulators that allow you to see how your web content will appear on those mobile devices. The default emulators can be found in the following location in the repository: */libs/wcm/mobile/components/emulators*

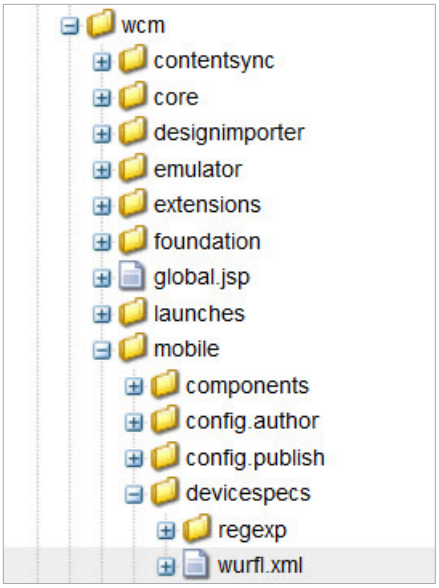
Those emulators are grouped, based on sets of capabilities, for example, 'supports images' or 'supports rotation'.

WURFL

WURFL stands for Wireless Universal Resource FiLe. AEM uses WURFL to determine which rendition of the page will be rendered to the mobile device. WURFL is an xml database that stores the different web browsing capabilities of a mobile device based on its User Agent. For AEM, WURFL is used to match the User Agent of the mobile device that is browsing the AEM page to the AEM rendering engine that displays the page. You can find more information about WURFL at:

http://wurfl.sourceforge.net/

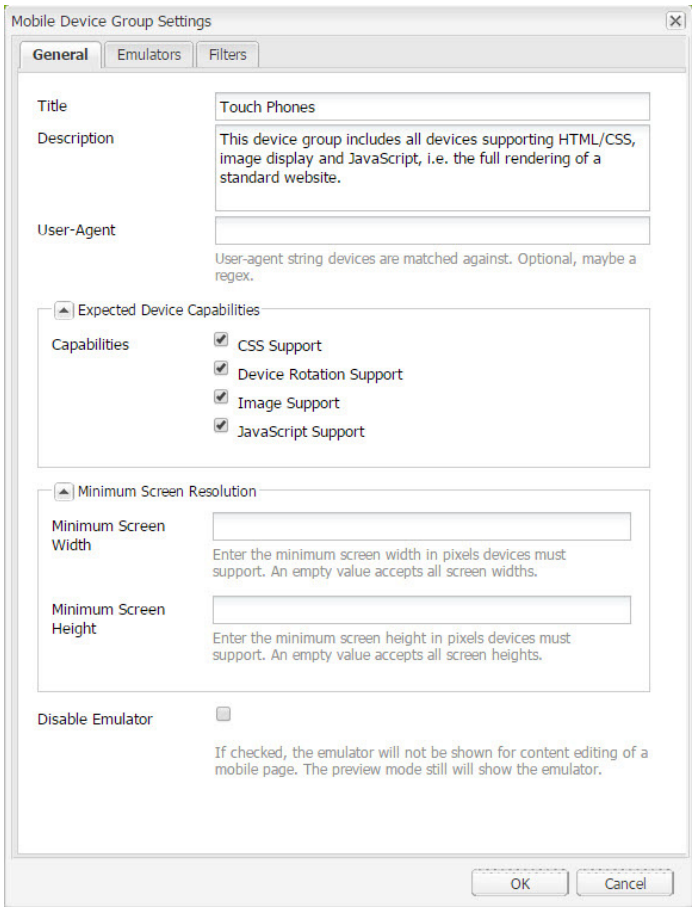
The AEM wurfl.xml file can be located at this URI: */libs/wcm/mobile/devicespecs/*



Each page in the mobile website allows you to specify the group of mobile devices that will be rendering the page. When the mobile page-rendering component inherits from the foundation mobile page component, */libs/wcm/mobile/components/page*, the emulator functionality is automatically integrated in the page.

Emulator Groups

An emulator group can be configured to specify the features that can be expected from it as well as the minimum screen resolution that can be expected from devices of such a group.



AEM enables authors to view a page in an emulator that simulates the environment in which an end user will view the page, for example, on a mobile device or in an email client.

Emulator Framework

The AEM emulator framework:

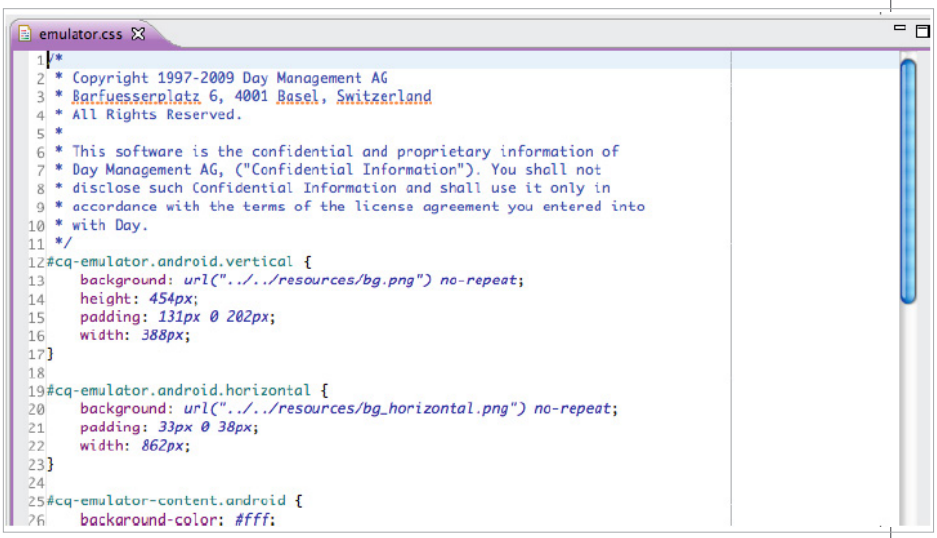
- provides content authoring within a simulated UI, for example, a mobile device or an email client (used to author newsletters)
- adapts the page content according to the simulated UI
- allows the creation of custom emulators

The emulator works by wrapping the HTML body content into emulator DIVs. See the following example HTML code:

```
<body>
    <div id="wrapper" class="page mobilecontentpage ">
        <div class="topnav mobiletopnav">
        ...
        </div>
        ...
    </div>
    </body>
```

When you create a mobile emulator, it will inherit many of its characteristics from the foundation emulator component's init.html.jsp. You can find the foundation emulator at: */libs/wcm/mobile/components/emulators/base*

The emulator appearance is controlled by a CSS client library. As an example, refer to: */libs/wcm/mobile/emulators/android/css/source/emulator.css*



If needed, define a JS client library, for example, to define a specific plugin.

- Name = js
- Node Type = cq:ClientLibrary

As an example, you can refer to the node: */libs/wcm/mobile/components/emulators/base/js*

If the emulator supports specific functionalities defined by plugins (like touch scrolling), create a configuration node below emulator:name = cq:emulatorConfig, Node type = nt:unstructured, and add the property that defines the plugin.

For example, to support rotation:

- Name = canRotate
- Type = Boolean
- Value = true

To support touch scrolling:

- Name = touchScrolling
- Type = Boolean
- Value = true

More functionalities can be added by defining your own plugins.

Complex Components Using JSP

Overview

This chapter explores turning end-user requirements into a development plan and then implementation.

Objective

In this chapter, you will learn how to:

- create and work with complex components.

Working With Complex Components

The components that have been created so far are not real-life components. The following sections explore turning end-user requirements into a development plan and then implementation. The requirements define a 'complex' component that manages textual and binary (i.e., image) content, and a Dialog box and Design dialog box. The requirements include allowing the component to be used by the parsys component, creating a dialog box with multiple tabs, enabling the functionality offered by the Content Finder (i.e., drag and drop), and other configurations.

When will I need to create a 'complex' component? Because the creation of 'complex' components is a common occurrence in AEM, it would benefit you to observe a mock requirements analysis. This exercise provides just that. First, you will observe the needs of a user. Then you will observe how an AEM developer may translate those needs.

Requirements: A component that can be dropped into a paragraph system and displays an image, rich text, and the path of a page in the system

The image:

1. must be editable by a content author
2. can be dragged and dropped from the Content Finder

The rich text:

1. must be editable by a content author
2. must allow tables to be created
3. must have a default value of 'This is some text'

The path of a page:

1. must be the same for every instance of this component, yet editable by a 'super' author
2. must live under the website structure '/content/training-site'
3. Widget should have property regexText with an error message if a regular expression fails

Solution Engineer's Translation: A paragraph system component that allows for the writing and displaying of three properties (two paragraph properties, one design/style property), and has a dialog box and Design dialog box.

The image:

1. is a Dialog box widget, most likely an xtype of smartimage
2. must be configured in the component's cq:editConfig to allow for dragging and dropping of images from the Content Finder

The rich text:

1. is a Dialog box widget, most likely an xtype of richtext
2. Widget should enable all the features of the rich text editing plugin table
3. Widget should populate property defaultValue with 'This is some text'

The path of a page:

1. is a Design dialog box widget, most likely an xtype of pathcompletion
2. Widget should have property regex with a regular expression validating the user's input (for example, '/^\\/content\\/training-site\\/(.)*$/')
3. Widget should have property regexText with an error message if a regular expression fails.

Although the order of sequence can differ, an Adobe Solution Engineer would most likely:

1. Create a component, and ensure that any paragraph system component can be this component's 'parent'.
2. Edit the default JSP so it displays the content/properties in an appropriate manner, even without any written content.
3. Create a dialog box for the component.
4. Create a Design dialog box for the component.
5. Add this component to the list of allowed components for a paragraph system component.
6. Test this component by adding it to a page to observe its output without any written content.
7. Add a rich text widget to the dialog box.
8. Add an image widget to the dialog box.
9. Add a path completion widget to the Design dialog box.
10. Perform necessary component configurations.

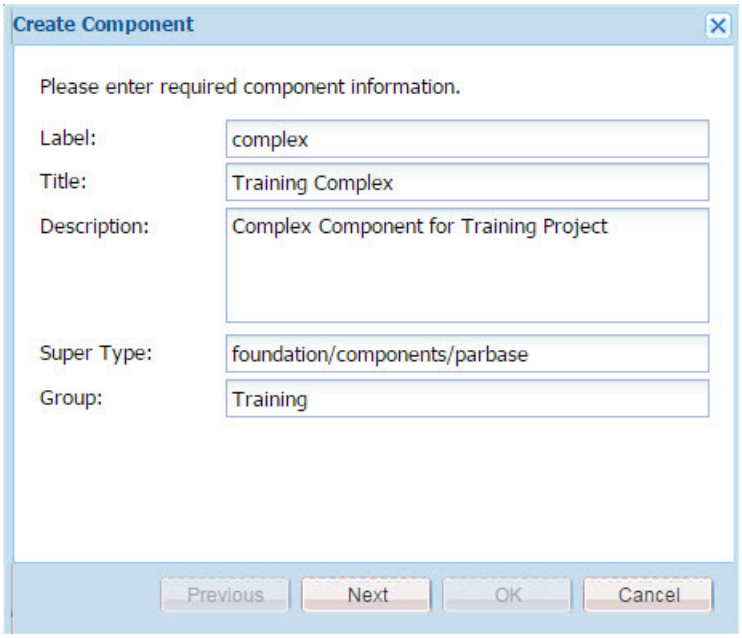
11. Test this component by writing content using the newly created dialog box and Design dialog box.

Exercise 11.1
Create a Complex Component

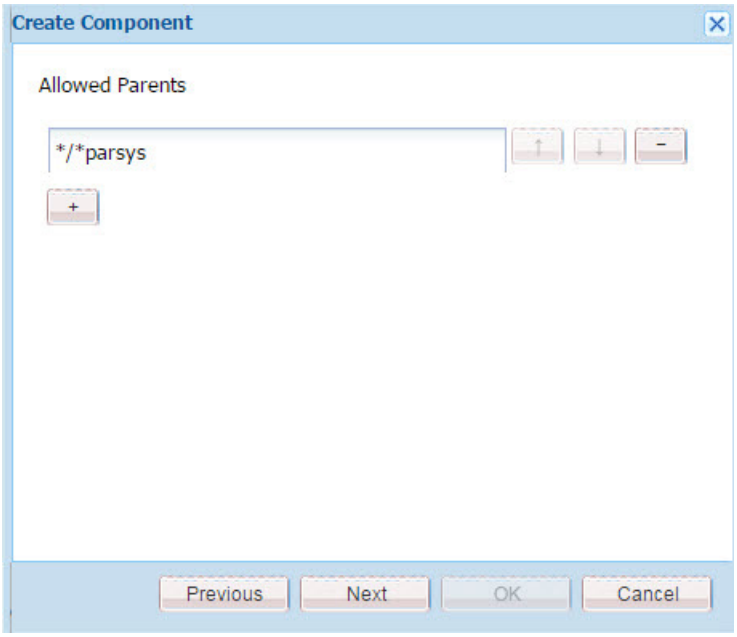
1. In CRXDE Lite, navigate to `/apps/training/component`, right-click the `component` node, and click **Create** > **Create Component...**.

2. Enter the following details for the component:

Property	Value
Label	complex
Title	Training Complex
Description	Complex Component for Training Project
Super Type	foundation/components/parbase
Group	Training



3. Click **Next**, and click **Next** once more. In the Allowed Parents section, enter `*/*parsys`.



4. Click **Next**, and then click **OK**.

5. Open the file `complex.jsp`, enter the following code, and then click **Save**.

```
<%@include file="/libs/foundation/global.jsp"%>
<%@ page import="com.day.cq.wcm.foundation.Image" %>
<%
// getting the image from the Dialog/resource
// notice the use of the second parameter "image" -- this signifies that the
// image will live on a resource (called image) below the requested resource
Image image = new Image(resource, "image");
// setting the selector so that the "parbase" can work its magic image.setSelector(".
    img");
// getting the rich text from the Dialog
String text = properties.get("text", "TEXT NA");
// getting the path from the Design Dialog
String path = currentStyle.get("path", "PATH NA");
%>
<h2><%= path %></h2>
<%= text %><br />
<%
image.draw(out);
%>
```

6. Right-click the `complex` component, and select **Create** > **Create Dialog...**. Enter the name as `dialog`, and click **OK**.

NOTE: For this exercise, do not concern yourself greatly with how the content is displayed because this can easily be altered by using code changes and/or CSS.

7. Locate the `tab1` node of the `dialog` node, and create an `items` node (node type `cq:WidgetCollection`) under it.
8. Create a `text` node (node type `cq:Widget`) under the newly created `items` node.
9. Assign the following properties to the newly created text node:
   a. The property that will define where content is stored

Name	Type	Value
name	String	./text

   b. The property that will define the widget type

Name	Type	Value
xtype	String	richtext

   c. The property that hides the label of the widget

Name	Type	Value
hideLabel	Boolean	true

   d. The property that will define the default value of the widget

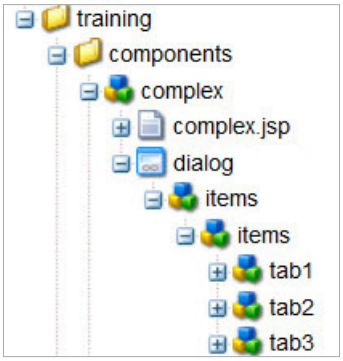
Name	Type	Value
defaultValue	String	This is some text.

10. Create an `rtePlugins` node (node type nt:unstructured) under the newly created `text` node.
11. Create a `table` node (node type nt:unstructured) under the newly created `rtePlugins` node.
12. Assign the following property to the `table` node:

Name	Type	Value
features	String	*

13. Copy the nodes `tab2` and `tab3` under the node `/libs/foundation/components/textimage/dialog/items`, and then paste it within the complex component's `dialog` node so that they are a peer of `tab1` (for example, `/apps/training/components/complex/dialog/items/items`).
    a. tab2 = the smartimage tab
    b. tab3 = advanced image properties

NOTE: Once again, it is often more efficient to copy and paste existing Dialogs/widgets that meet your needs. That being said, it is wise to review what you copy to better understand the internal workings of AEM. If you examine closely enough, you will see that the widget is actually storing image-related content at a level deeper than the current resource (for example, ./image/file, ./image/fileReference, and so on). This ties nicely with your previously written code (Image image = new Image(resource, "image");).



14. Create a Design dialog box.
15. Create an `items` node (node type is cq:WidgetCollection) under the `tab1` node of your component's Design dialog box.
16. Create a `path` node (nodeType cq:Widget) under the newly created `items` node.
17. Assign the following properties to the newly created path node:
    a. The property that will define where content is stored

Name	Type	Value
name	String	./path

    b. The property that will define the widget type

Name	Type	Value
xtype	String	pathfield

    c. The property that will define the label applied to the widget

Name	Type	Value
fieldLabel	String	Enter a Path

    d. The property that will define the root path to appear

Name	Type	Value
rootPath	String	/content/training-site

    e. The property that will define the regular expression used to evaluate user input

Name	Type	Value
regex	String	/^\\/content\\/training-site\\/(.)*$/

f. The property that will define the error message if a user's input fails the regular expression

Name	Type	Value
regexText	String	Please insert a page that "lives" under /content/training-site

18. Copy the node `/libs/foundation/components/textimage/cq:editConfig`, and then paste to the root node of your complex component (for example, `/apps/training/components/complex`). This enables drag-and-drop capabilities from the Content Finder. To be able to drag and drop assets from the Content Finder to a component on a page, there must be a drop targets configuration node called `cq:dropTargets` (of type nt:unstructured) below the edit configuration node (cq:editConfig) of a component.

19. Navigate to `/apps/training/components/complex/cq:editConfig/cq:dropTargets/image`.

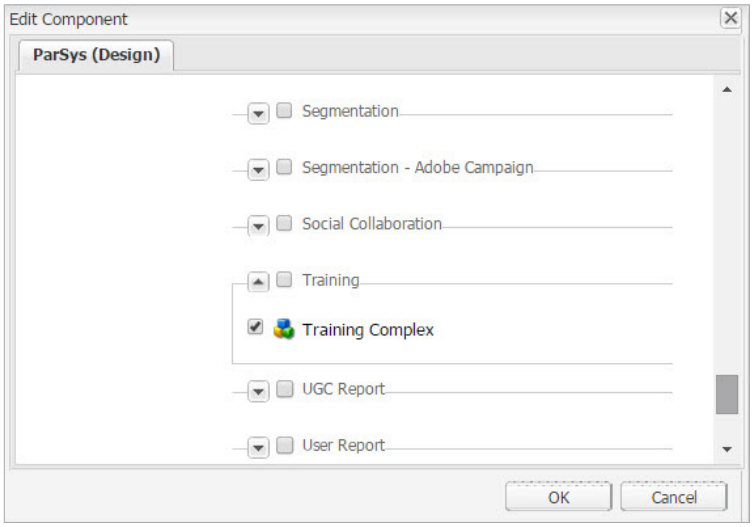
20. Validate that the image node has the following properties:
    a. accept (Type = String): The media types to be accepted (for example, image/.*)
    b. groups (Type = String): The groups in the Content Finder that assets can be accepted from (for example, media)
    c. propertyName (Type = String): The property the reference should be stored in (for example, ./image/fileReference)

21. Navigate to the `parameters` node that is a child to the `image` node. Change the value of the `sling:resourceType` property to have the value: `training/components/complex`. Remember that the `sling:resourceType` property should reference the location to find a rendering script.

22. Save your changes.

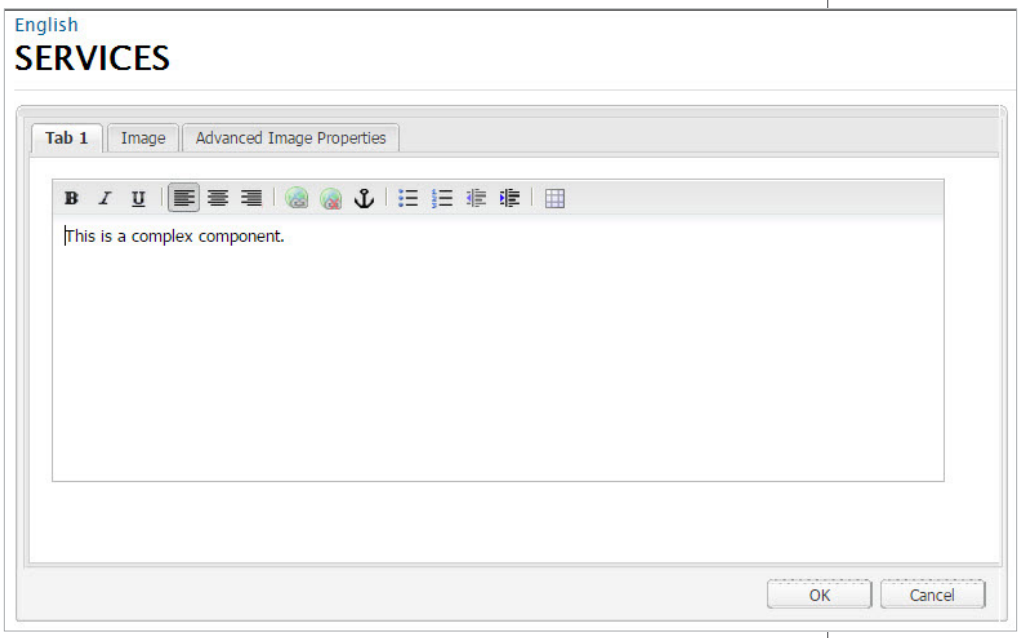
23. Open the Services page of the training site in Classic UI, and go to the design mode.

24. Click **Edit** next to the parsys component (shown as Design of content). In the **Training** group, select the **Training Complex** check box, and click **OK**.

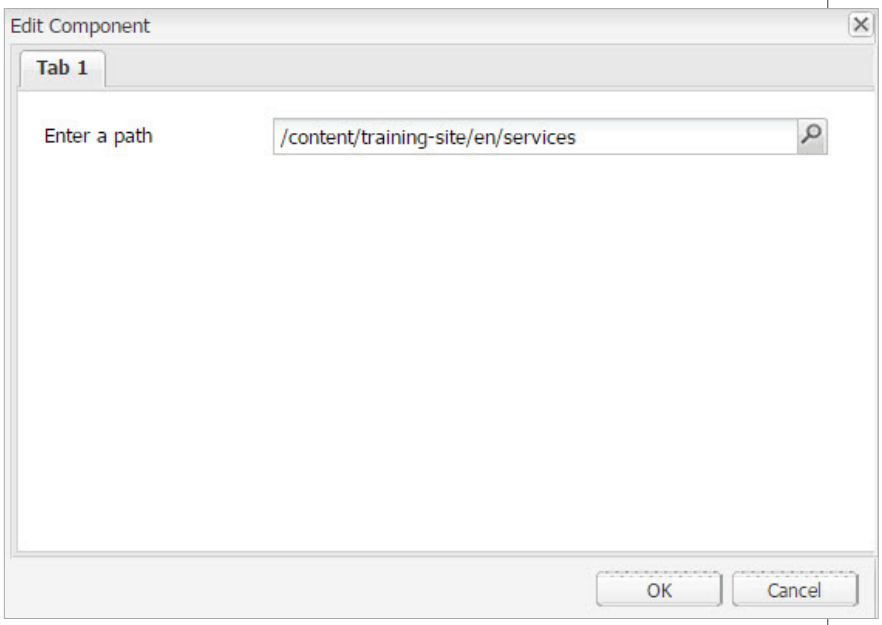


NOTE: Notice how your complex component is listed under the Training group. This is because you declared the group as Training during the creation of your component.

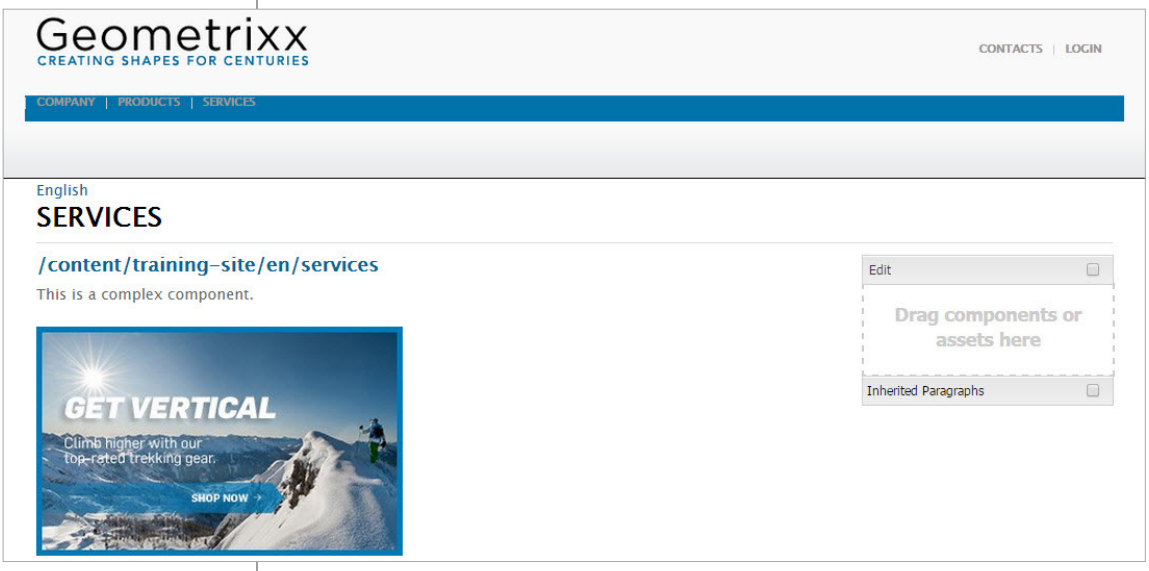
25. Go back to the edit mode. From the components list, drag and drop the **Training Complex** component onto the page.
26. From the Assets list on the right, drag and drop an asset onto the complex component.
27. Select the text area, and click once more to enable in-line editing. Type some text there, and then click outside the component to save and exit.
28. Double-click the complex component. You can see the following dialog box:



29. Go to the design mode, and click **Edit** next to **Design of complex**.
30. In the **Edit Component** dialog box, browse and select the path. Click **OK**.



31. Go to the preview mode, and view the result.



Congratulations! You successfully created a complex component that contains a dialog box, Design dialog box, default values, custom configuration, and validation of user input, in addition to enabling drag-and-drop functionality from the Content Finder.

End-User Search

Searching for content in AEM is similar to 'traditional' searches you created in the past.

1. An HTML form is needed to collect the user's input (search string).
2. After the form has been submitted, you need to capture the search string.
3. You need to prepare a query statement based on the search string.
4. A query object needs to be created that will connect to the content repository and implement the query statement.
5. You need to collect and parse the query results, if any.
6. Output related to the query results should be displayed appropriately.

When querying a JCR using the JCR API, some basic functionality (API calls) needs to be implemented:

- javax.jcr.Session: JCR session, can be reached for example by Node. getSession()
- javax.jcr.Workspace: JCR workspace, can be reached for example by Session. getWorkspace()
- javax.jcr.query.QueryManager: QueryManager is used to create a Query object and can be reached using Workspace.getQueryManager()
- createQuery(String statement, String language): Creates the Query object for the provided statement in the provided language (for example, SQL)

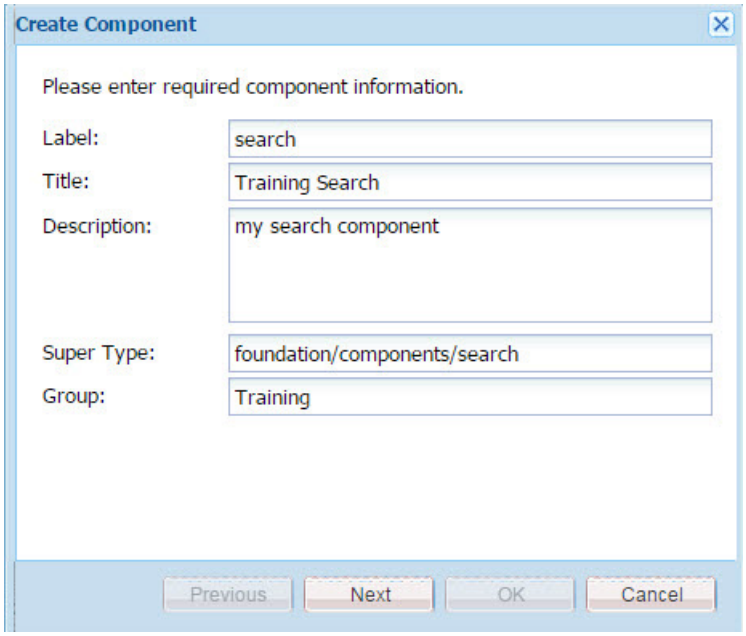
- javax.jcr.Query
- execute(): Executes this Query and returns a QueryResult object
- javax.jcr.query.QueryResult
- getRows(): Returns an iterator over the Rows of the query result table

For more detailed information, review the Java docs for the JCR and CRX provided in the documentation.

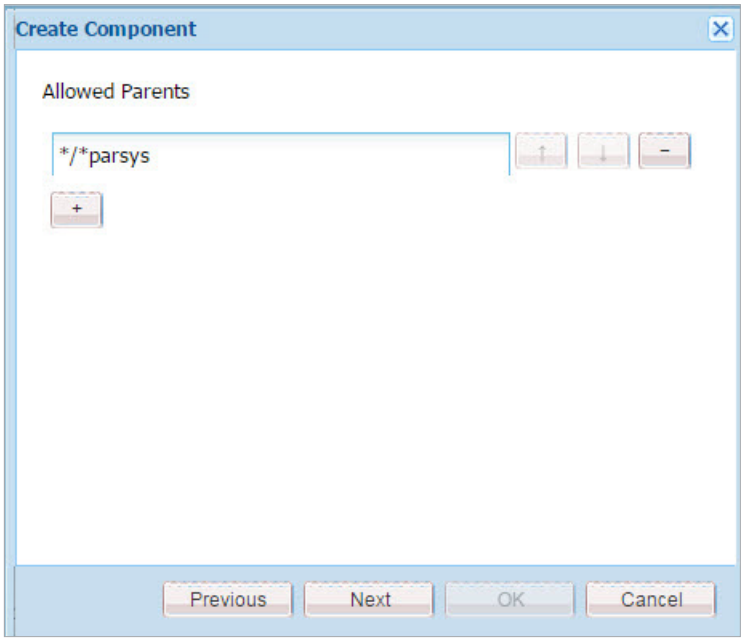
Exercise 11.2
Create a Search Component

The following instructions explain how to create a component that will allow visitors to search the content of the website/repository. This exercise will demonstrate the differences among the multiple Search APIs. The search component can be placed in the parsys of any page, and has the ability to search the content of the website based on a query string provided in the request.

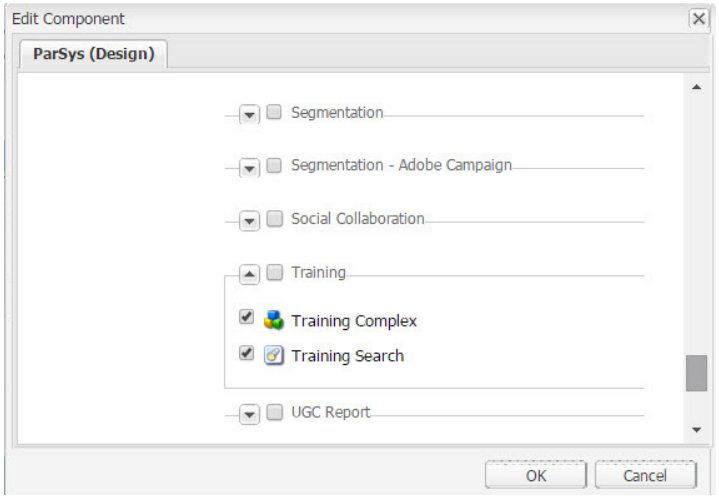
1. Create a new `search` component. Right-click `components`, and select **Create** > **Create Component...**.



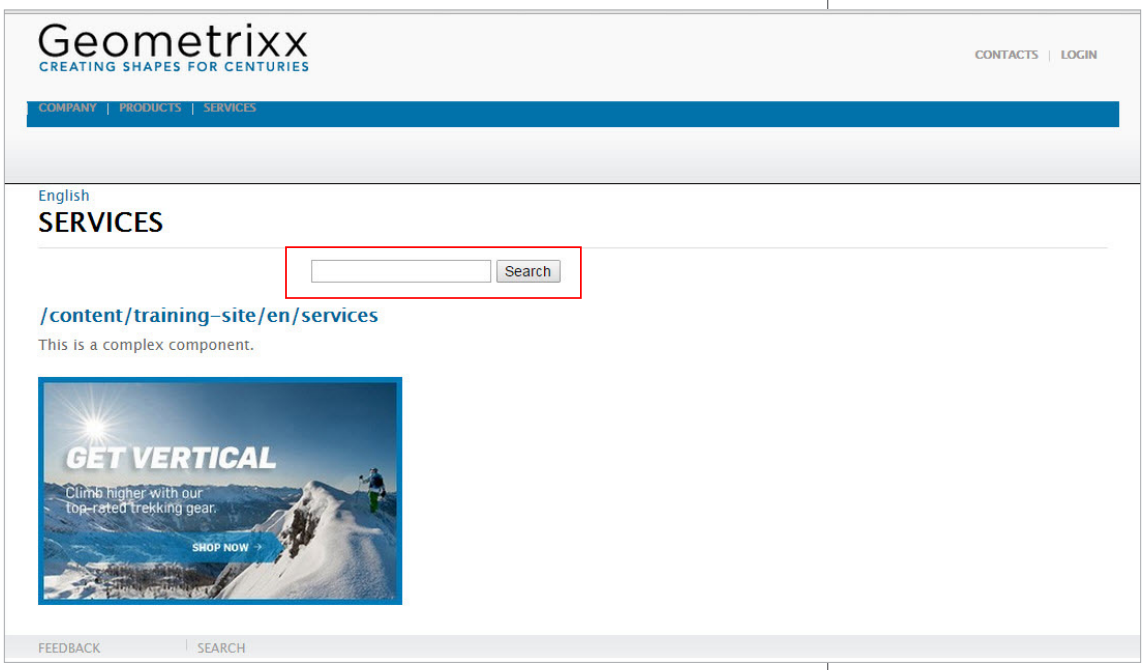
2. Click **Next**, and click **Next** once more. In the **Allowed Parents** section, type `*/*parsys`.



3. Click **Next**, and then click **OK**.
4. Expand the `search` component. Replace `search.jsp` with the `search.jsp` from the USB drive.
5. Update `search.jsp` to ensure that the search path matches the path to your training site.
6. Save your changes.
7. Open the **Services** page of your training site in design mode.
8. Add your search component to the paragraph system component. Notice how the component will be found in the 'Training' group. This was defined during the creation of the component.



9. Go to the edit mode, and test your script by adding the Training Search component to the paragraph system component of a training page. If successful, you should see the default search component, similar to the image below.



10. Search for a word you know exists on a separate page in your Training website structure. You may need to perform this search in page preview mode to ensure a clean request.
11. Now replace the contents of `search.jsp` with the contents of `search.cq5wcm.jsp`.
12. Examine the code to see the differences between the JCR search API and the WCM search APIs.
13. Try the search again.
14. Now replace the contents of `search.jsp` with the contents of `searchenhanced.cq5wcm.jsp`.
15. Examine the code to see a good example of using Expression Language (JSTL) with AEM. You should also note the extensive use of properties set in the dialog box and the use of facets. The code also builds a search term tag cloud called 'Search Trends'.
16. Try the search again.

Congratulations! You successfully created a search component that queries the content in your Training website structure. You can further enhance this component by adding widgets to the dialog box to output default messages written by a content author if the search was successful or unsuccessful.

Using jQuery With Ajax and Apache Sling

Sling lets you drive your content repository from within a webpage by making an Ajax request with jQuery. jQuery is a cross-browser JavaScript library designed to simplify the client-side script of HTML. jQuery works like JavaScript where it is used to help with interaction and effects with your development code. jQuery makes it easy to handle DOM objects, events, effects, and Ajax, automatically takes care of JavaScript leaks, and has countless third-party plugins.

jQuery is not a language but it is a well-written JavaScript code. The most common jQuery effects are drop-down menus, drag-and-drop elements, animations, and form validation. Developers have also connected this with other coding languages like JSP, ESP, and ASP.

The jQuery library has a full suite of Ajax capabilities. The functions and methods provided allow you to load data from the server without a browser page refresh.

Exercise 11.3
Create a Component to Display Dynamic Grid of User Accounts

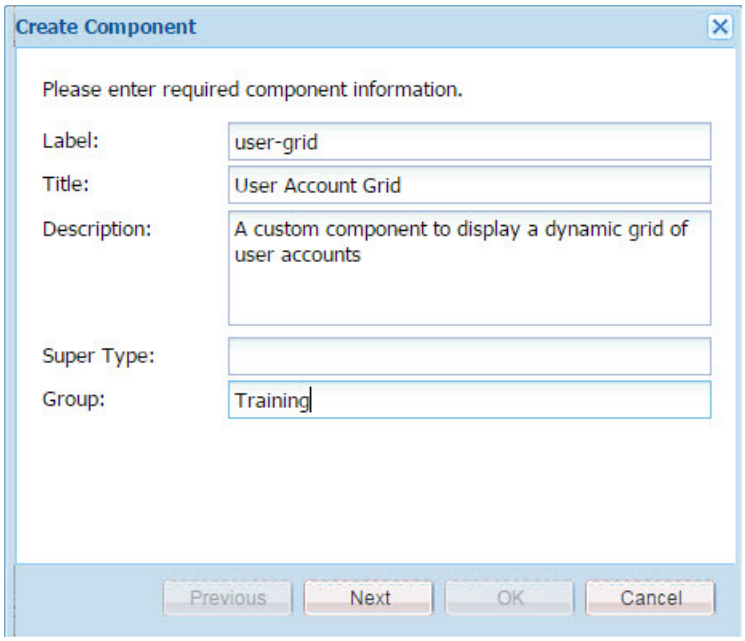
NOTE: The default Sling Servlet in AEM can be used to retrieve content in JSON format for anything; however, this use case does not use the default Sling Servlet. We will provide our own request handler so we can understand the complete workflow involved.

Create a custom component that will display a dynamic grid of user accounts. This component will make use of the jQuery plugin that uses Ajax to retrieve data from AEM. The jQuery Flexigrid plugin is the target for this exercise. The plugin will execute a callback to a JSP in AEM to retrieve JSON-formatted data.

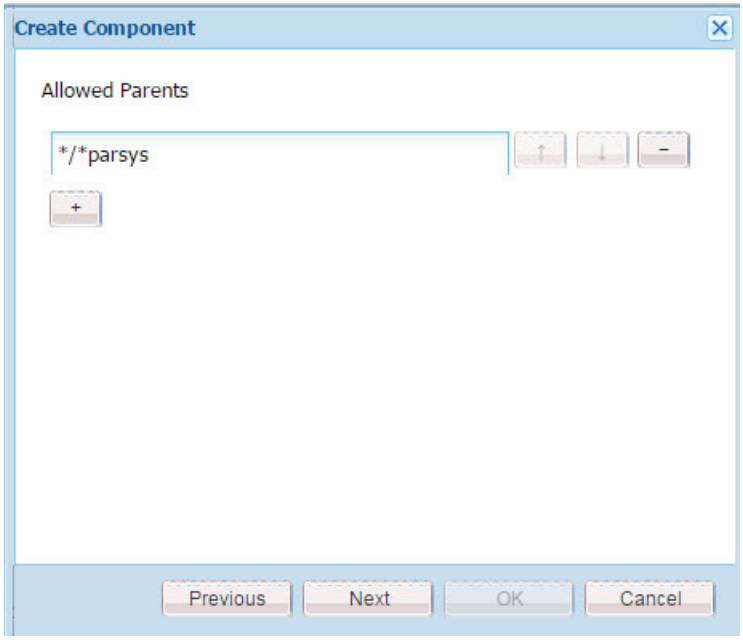
Create the Component

1. Right-click the `components` folder, and select **Create** > **Create Component...**.
2. Add the following properties to the component:

Property	Value
Name	user-grid
Title	User Account Grid
Description	A custom component to display a dynamic grid of user accounts
Group	Training



3. Click **Next**, and click **Next** once more. Enter `*/*parsys` for **Allowed Parents**.



4. Click **Next**, and then click **OK**.

5. Expand the newly created component, and add the following code to the `user-grid.jsp` file.

```
<%@include file="/libs/foundation/global.jsp"%><%
%>
<script type="text/javascript">
   $CQ(function ($) {
    $('.user-table').flexigrid();
   });
</script>
<table class="user-table">
    <thead>
       <tr>
        <th width="100">Col 1</th>
        <th width="100">Col 2</th>
        <th width="100">Col 3 is a long header name</th>
        <th width="300">Col 4</th>
       </tr>
    </thead>
    <tbody>
       <tr>
        <td>This is data 1 with overflowing content</td>
        <td>This is data 2</td>
        <td>This is data 3</td>
        <td>This is data 4</td>
       </tr>
       <tr>
        <td>This is data 1</td>
        <td>This is data 2</td>
        <td>This is data 3</td>
        <td>This is data 4</td>
       </tr>
       <tr>
        <td>This is data 1</td>
        <td>This is data 2</td>
        <td>This is data 3</td>
        <td>This is data 4</td>
       </tr>
       <tr>
        <td>This is data 1</td>
        <td>This is data 2</td>
        <td>This is data 3</td>
        <td>This is data 4</td>
       </tr>
       <tr>
        <td>This is data 1</td>
        <td>This is data 2</td>
```

```
<td>This is data 3</td>
        <td>This is data 4</td>
       </tr>
       <tr>
        <td>This is data 1</td>
        <td>This is data 2</td>
        <td>This is data 3</td>
        <td>This is data 4</td>
       </tr>
    </tbody>
</table>
```

6. Create an empty dialog box. Right-click the `user-grid` component, and select **Create** > **Create Dialog**. Retain the default values, and click **OK**.

7. Right-click the `user-grid` component, and click **Create...** > **Create Node...**

Property	Value
Name	clientlibs
Type	cq:ClientLibraryFolder

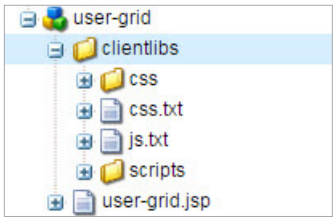
8. Add the following properties to the `clientlibs` node:

Name	Type	Value
dependencies	String[]	cq.jquery
categories	String[]	componentlab

9. Install the `user-grid-libs-1.0.zip` from the respective folder in USB. This package installs the assets you need to develop the User Grid component in a `custom` folder of the Training project.

10. Refresh the training project folder.

11. Copy the content of the `custom` folder to the `clientlibs` folder. The directory structure looks as shown below:



12. Examine flexigrid.js and flexigrid.css files so you understand what they do. (They are also provided in the Asset folder in USB.)

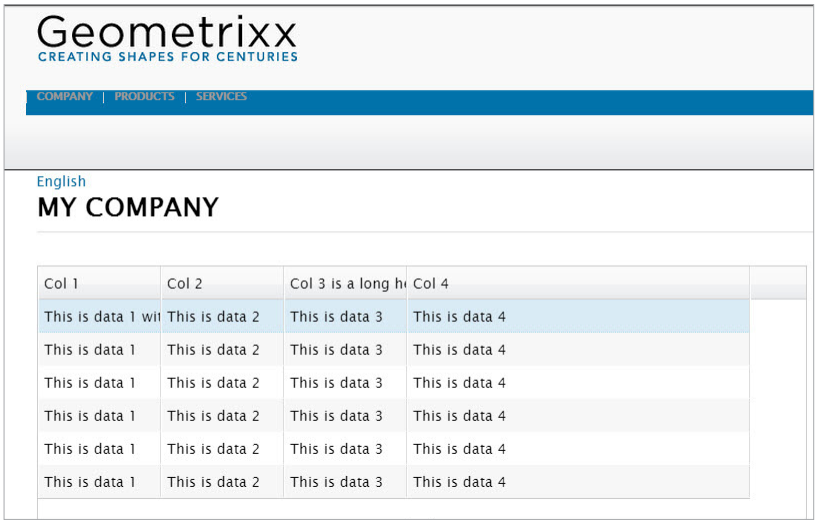
NOTE: Components will not show up in the design unless it has a classic dialog box attached to it.

13. Make the following change to the beginning of `user-grid.jsp` to pick up the client library:

```
<%@include file="/libs/foundation/global.jsp"%><%
%>
<!-- pick up the client libraries -->
<cq:includeClientLib categories="componentlab" />
<script type="text/javascript">
    $CQ(function ($) {
    $('.user-table').flexigrid();
    });
</script>
```

14. Open the Company page of the training site in design mode. Click the **Configure** icon for the paragraph component, and add the **User Account Grid** component.

15. Go back to the edit mode, drag and drop the **User Account Grid** component to the page. Refresh your page. It should look like the following:



Make the Component Interactive

1. Modify `user-grid.jsp` by adding the following code:

```
<%
%><%@include file="/libs/foundation/global.jsp"%><%
%><%@page session="false" %>
<!-- Pick up the client libraries -->
<cq:includeClientLib categories="componentlab" />
<script type="text/javascript">
/* Grab the JCR path to the content entry that calls this component with Sling, you
    cannot call a script, you must call the jcr content node that resolves to the
    representation (script). */
var baseURL = "<%= currentNode.getPath() %>";
$CQ(function () {}
```

2. Modify the `$CQ(function())` to match the following:

```
CQ(function () {
    $CQ('.user-table').flexigrid({
    url: baseURL + '.json', //This will trigger a POST request back to CQ
    dataType: 'json', //The expected response will be JSON formatted data
    colModel : [ {
        display : 'User ID', name : 'id', width : 215, sortable : true, align
    : 'left', hide: false
    }, {
    display : 'First Name', name : 'givenName', width : 100, sortable : true,
    align: 'left', hide: false
    }, {
    display : 'Last Name', name : 'familyName', width : 100, sortable : true,
    align: 'left', hide: false
    },{
    display : 'Email', name : 'email', width : 215, sortable : true, align :
    'left', hide: false
    }],
    buttons : [
        {name: 'Add', bclass: 'add', onpress : userAdd},
        {name: 'Edit', bclass: 'edit', onpress : userEdit},
        {name: 'Delete', bclass: 'delete', onpress : userDelete},
        {separator: true}
        ],
    searchitems : [
        {display: 'User ID', name : 'user_id',isdefault: true},
        {display: 'First Name', name : 'givenName'},
        {display: 'Last Name', name : 'familyName'}
        ],
    sortname: "id",
    sortorder: "asc",
    usepager: true,
    title: "User Account Grid",
    useRp: true,
    rp: 15,
    showTableToggleBtn: false,
    singleSelect: true,
    width: 700,
    height: 200
    });
});
```

NOTE: In the given code, many advanced jQuery flexigrid plugin options have been provided. The goal of this lab is not to cover flexigrid in detail. You can find additional information about Flexigrid for jQuery at http://flexigrid.info.

3. Now add the following JavaScript functions just before the closing `</script>` tag.

```
function userAdd() {
    alert("Add button clicked");
}

function userEdit() {
    alert("Edit button clicked.");
}

function userDelete() {
    alert("Delete button clicked.");
}
```

4. Delete the hardcoded text between the <table class="user-table"> and </table> tags.

Create the callback JSP File

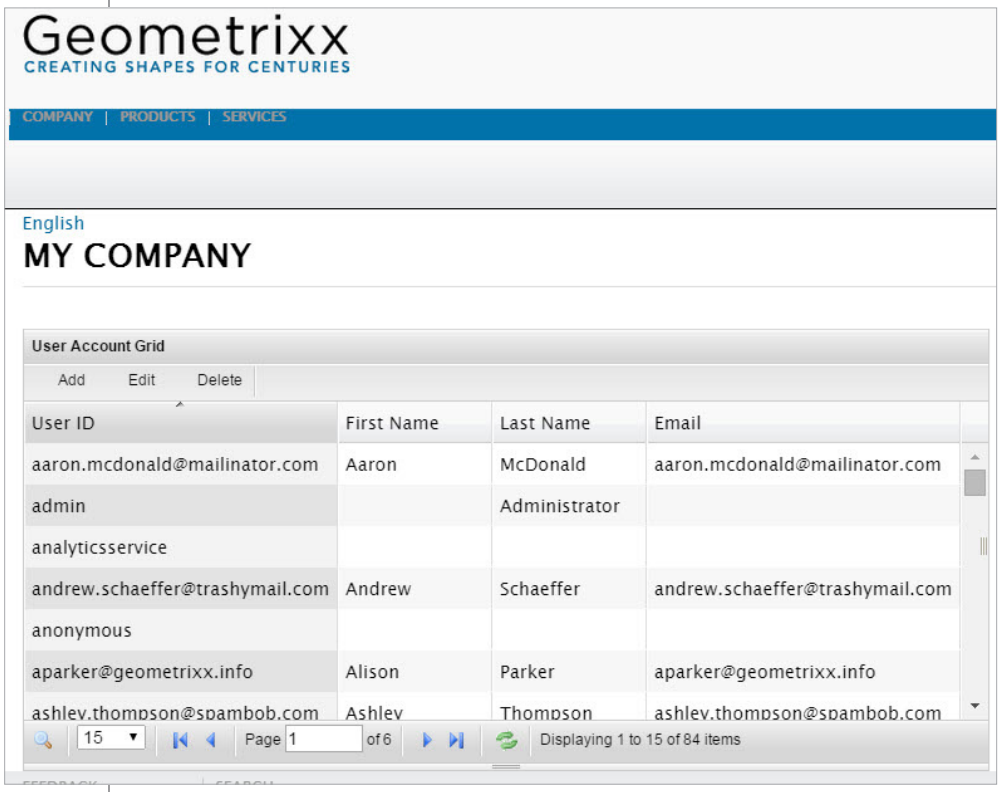
1. Create the `user-grid.json.POST.jsp` callback script. Right-click the `user-grid` component node and click **Create** > **Create File...**.
2. Type the following code into your new user-grid.json.POST.jsp script:

```
<%@page session="false" %>
<%@include file="/libs/foundation/global.jsp"%>
<%@ page import="org.apache.sling.jcr.api.SlingRepository" %>
<%@ page import="com.day.cq.security.UserManager" %>
<%@ page import="com.day.cq.security.UserManagerFactory" %>
<%@ page import="com.day.cq.security.User" %>
<%@ page import="com.day.cq.security.Authorizable" %>
<%@ page import="com.day.cq.security.profile.Profile" %>
<%@ page import="java.util.Iterator" %>
<%@ page import="java.util.List" %>
<%@ page import="java.util.ArrayList" %>
<%@ page import="com.day.cq.commons.TidyJSONWriter" %>

<%

//Local variables
final SlingRepository repos = sling.getService(SlingRepository.class);
final UserManagerFactory umFactory = sling.
getService(UserManagerFactory.class);

Session session = null;
Iterator<User> userIterator = null;
Iterator<Authorizable> authorizableIterator = null;
```

```
try
{
    // Ensure that the currently logged on user has admin privileges.
    session = repos.loginAdministrative(null);

    final UserManager um = umFactory.createUserManager(session);
        final TidyJSONWriter writer = new TidyJSONWriter(response.getWriter());
userIterator = um.getUsers();
List<User> users = new ArrayList<User>();
User tmpUser;

// copy iterator into a List for additional manipulations.
while(userIterator.hasNext())
{
    tmpUser = userIterator.next();
    users.add(tmpUser);
}

//Begin writing JSON response
writer.setTidy("true".equals(request.getParameter("tidy")));
writer.object();
writer.key("page").value(1);
writer.key("total").value(users.size());
writer.key("rows").array();

for(int i=0; i < users.size(); i++)
{

    User aUser = users.get(i);
    Profile aProfile = aUser.getProfile();
    writer.object();
    writer.key("id").value(aUser.getID());
    writer.key("cell").array();
    writer.value(aUser.getID());
    writer.value(aProfile.getGivenName());
    writer.value(aProfile.getFamilyName());
    writer.value(aProfile.getPrimaryMail());
    writer.endArray();
    writer.endObject();
   }
    writer.endArray();
    writer.endObject();
    session.logout();
}
catch (Exception e)
```

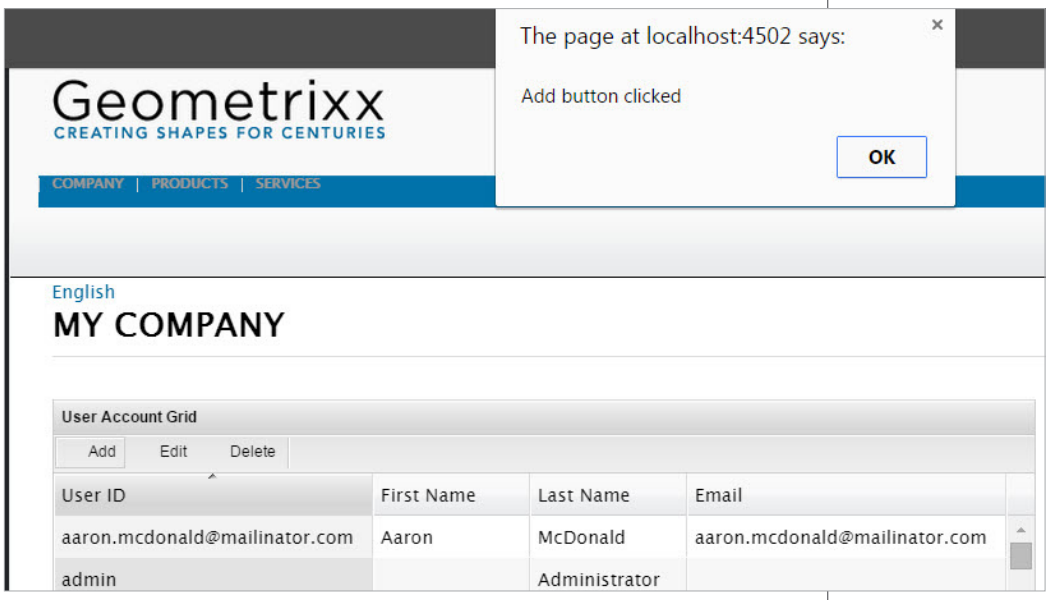
```
{
    System.out.println("myajaxsample Exception Occured: " + e.getMessage());
}
finally
{
    session.logout();
    session = null;
}
%>
```

3. Test your component by refreshing the page that contains the component in the paragraph system. Notice the interactive grid now appearing on the page.



4. Click the action buttons (Add, Edit, and Delete) to see the appropriate message box appearing.

5. If needed, install the complete package from USB > `9-3 user-grid-component/crx-package/user-grid-step2`.



NOTE: Not all the grid features are functional. The search and pagination functions have not yet been implemented. This is left as an extra credit exercise.

OSGi Bundles and Workflow

Overview

OSGi bundles are essential when you need to add new functionalities to your application. OSGi defines an architecture for developing and deploying modular applications and libraries.

Workflows are used to represent processes. The steps within workflows can be defined within AEM and then applied to the appropriate pages.

Objective

In this chapter, you will:

- learn how to create OSGi bundles.
- understand the basics of the Workflow Console.
- implement a Workflow step.

Creating OSGi Bundles

Whenever you need to add new functionality to your application in the form of new Java classes, you can create an OSGi bundle with the Java class inside. This will allow you to create and use custom Java classes in your JSP scripts or Sightly scripts, allowing for more traditional Java development and library reuse. Note that you can also use Java classes directly in Sightly scripts.

What Exactly Is an OSGi Bundle?

As discussed previously, OSGi defines an architecture for developing and deploying modular applications and libraries (it is also known as the Dynamic Module System for Java). OSGi containers allow you to break your application into individual modules (JAR files with additional meta information and called bundles in OSGi terminology) and manage the cross-dependencies between them with:

- services implemented within the container
- a contract between the container and your application

These services and contracts provide an architecture, which enables individual elements to dynamically discover each other for collaboration.

An OSGi framework then offers you dynamic loading/unloading, configuration, and control of these bundles without requiring restarts. This architecture allows you to extend Sling with application-specific modules. Sling, and therefore AEM, uses the Apache Felix implementation of OSGi and is based on the OSGi Service Platform Release 4 Version 4.2 Specifications. They are both collections of OSGi bundles running within an OSGi framework. This enables you to perform the following actions on any of the packages within your installation:

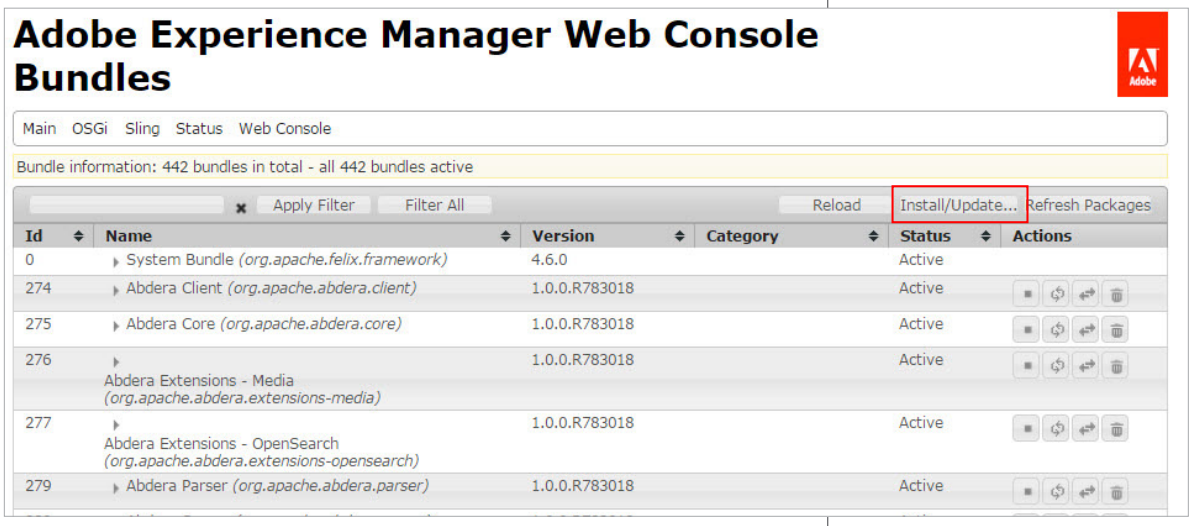
- Install
- Start
- Stop
- Update
- Uninstall
- See the current status
- Access more detailed information (for example, symbolic name, version, and location) about the specific bundles

Exercise 12.1
Consume an OSGi Bundle

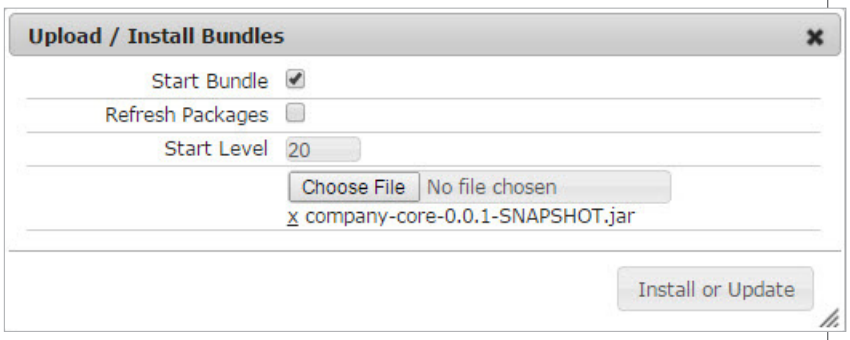
Creating a bundle is beyond the scope of the course. So, you will use the bundle that is provided in the exercise folder. Following is the Java class that is used for creating the bundle:

```
package org.training.test;
public class HelloWorld {
public String getMessage() {
    return "Hello World !!!";
   }
}
```

1. Navigate to the system console.
   http://localhost:4502/system/console
2. In the upper-right corner, click **Install/Update**.



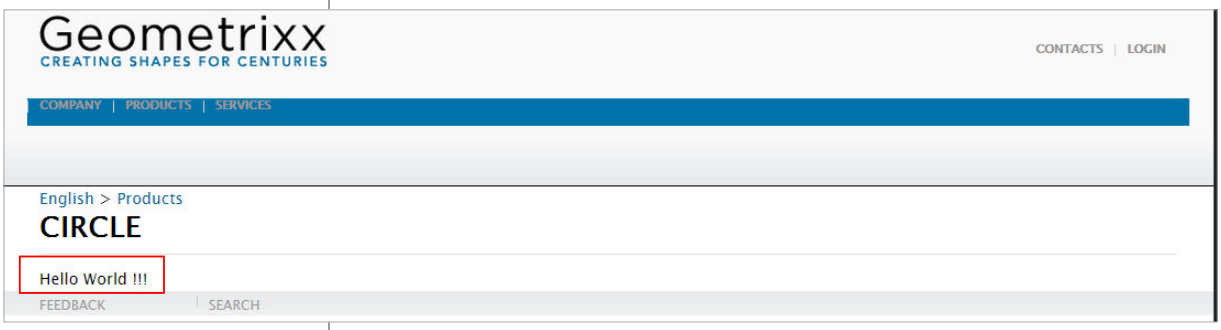
3. Select the **Start Bundle** check box, and click **Choose File**. Select the bundle from the exercise folder, and click **Install or Update**.



4. In the browser, search for Company-Core. (Press Ctrl+F or command-F.) Note that the bundle is installed.
5. Navigate to the **Components** tab (**OSGi** > **Components**). In the browser, search for the `HelloWorld` component. Note that the component is installed.
6. In CRXDE Lite, open the default script of the Title component you developed. (`components/training-title/training-title.html`)
7. Add the following code:

```
<h1 data-sly-use.title="title.js">${title.text}</h1>
<div data-sly-use.bundle="com.adobe.training.core.test.HelloWorld">${bundle.getMessage}</div>
```

8. Refresh any of the pages you created. Observe that the message from the bundle is displayed.



Basics of the Workflow Console

AEM encompasses several applications that are designed to interact and complement each other. In particular, the Workflow engine can be used closely with several other applications.

For example, within AEM, AEM Sites is key. This enables you to generate and publish pages to your website. This functionality is often subject to organizational processes, including steps such as approval and sign-off by various participants. These processes can be represented as workflows, which in turn can be defined within AEM and then applied to the appropriate content pages.

Overview of the Main Workflow Objects

Model

A Model comprises WorkflowNodes and WorkflowTransitions. The transitions connect the nodes and define the flow. The Model always has a start node and an end node. Workflow models are versioned. Running workflow instances keep the initial workflow model version that is set when the workflow is started.

Steps

There are different types of workflow steps:

- Participant (User/Group)
- Process (Script, Java method call)
- Container (Sub Workflow)
- OR Split/Join
- AND Split/Join

All the steps share the following common properties—AutoAdvance and Timeout alerts (scriptable).

Transition

A transition defines the link between two consecutive steps. It is possible to apply rules to the transition.

WorkItem

The WorkItem is the 'there is a task identifier' and is put into the respective inbox. A workflow instance can have one or many WorkItems at the same time, depending on the workflow model.

The WorkItem references the workflow instance. In the repository, the WorkItem is stored below the workflow instance.

Payload

Payload references the resource that has to be advanced through a workflow. The payload implementation references a resource in the repository (by a path or an UUID), a resource by a URL, or by a serialized Java object. Referencing a resource in the repository is flexible and, with Sling, productive. For example, the referenced node could be rendered as a form.

Life Cycle

A life cycle is created when starting a new workflow (by choosing the respective workflow model and defining the payload), and ends when the end node is processed. The following actions are possible on a workflow instance:

- Terminate
- Suspend
- Resume
- Restart

Completed and terminated instances are archived.

Inbox

Each logged in user has a unique workflow inbox in which the assigned WorkItems are accessible. The WorkItems are assigned to a specific user or to a group the user belongs to.

Workflow Console

The Workflow console is the centralized location for workflow management in AEM. It can be accessed using the Workflows button on the Welcome page or the Workflows button on the toolbar on any AEM console (for example, Web sites, Tools, Tagging).

Within the console, there are four tabs:

Models

Lists the workflow models currently available. Here you can create, edit, or delete a workflow model.

Instances

Shows you details of workflow instances that are currently active. These instances are also version dependent.

Archive

Enables you to access details of workflow instances, which have terminated for whatever reason

Launcher

Allows you to define a workflow to be launched if a specific node has been updated

Starting a Workflow

There are four methods of manually starting a workflow:

- Workflow Console
- Site Admin (Websites tab)
- Context Menu
- Workflow item on the toolbar
- Sidekick

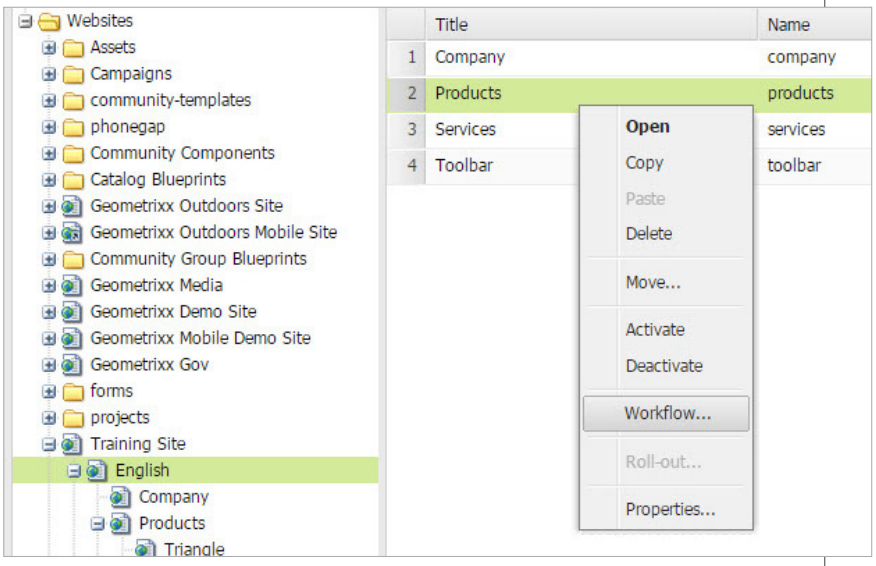
NOTE: The payload is assigned to the current version of the workflow. If the main copy of the workflow is updated later, the changes will have no impact on the currently running instance.

AEM is built within an OSGi application framework. OSGi is a dynamic module system for Java that provides a framework within which small, reusable, standardized components can be composed into an application and deployed. The Apache Sling framework is designed to expose a JCR content repository through an HTTP-based REST API. AEM's native functionality and the functionality of any website built with AEM are delivered through this framework.

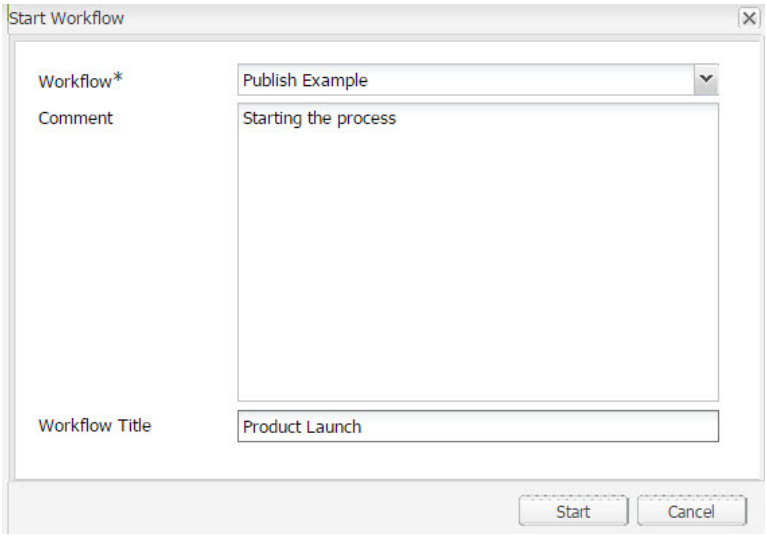
Exercise 12.2
Explore Basic Workflow

You will be starting the workflow from the Site Admin console.

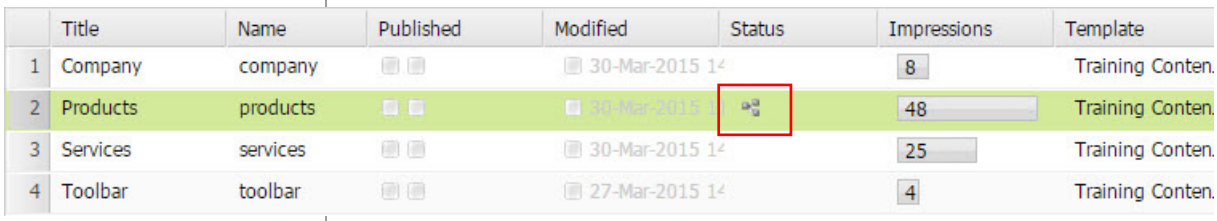
1. Open the Site Admin, or click the following link: http://localhost:4502/siteadmin#/content/training-site/en
2. Right-click one of your pages, and click **Workflow**.



3. Select the **Publish Example** workflow, fill in the desired optional information, and click **Start.**

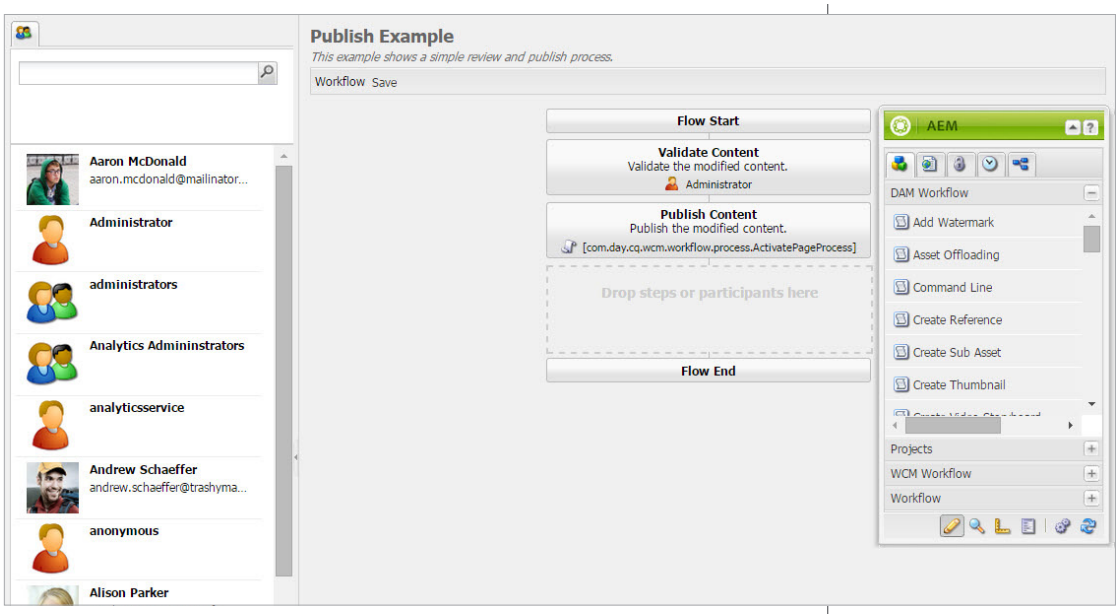


You will notice that the workflow icon shows up in the Site Admin console next to the page you chose.

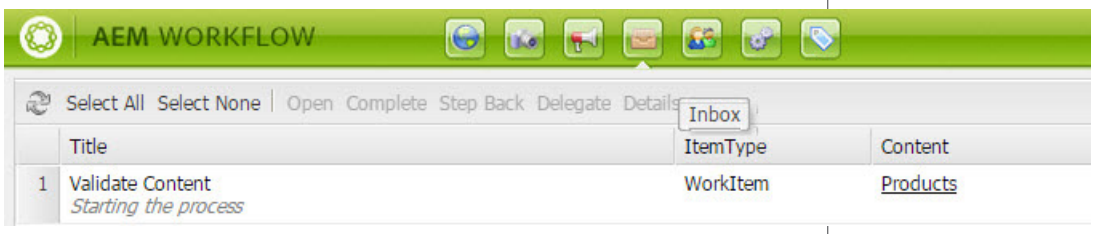


4. Go to Tools by clicking this link: http://localhost:4502/miscadmin#/
5. Double-click **Workflow** to open the Workflow console.
6. Search for the Publish Example, and double-click it to open it. The **Publish Example** workflow appears. You will notice that the Publish Example workflow has two steps—Validate Content and Publish Content.

   The Validate Content step is a participant step assigned to the user admin. The Publish Content step is a process step with the Implementation Property set to ActivatePageProcess. You can validate the values of the step properties by double-clicking the step to open the dialog box.

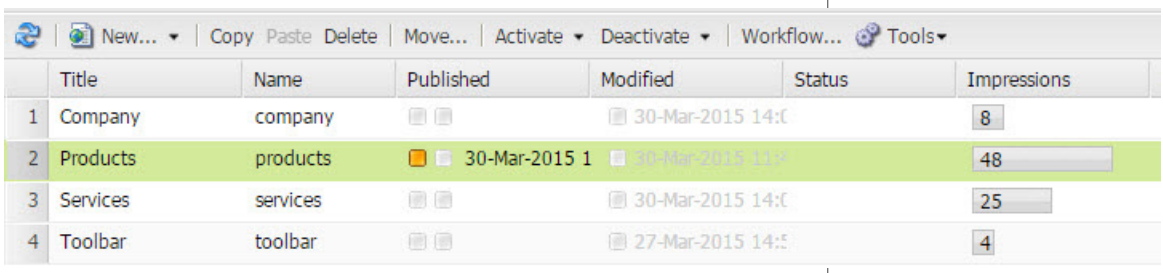


7. Switch to the **Inbox** by changing to the Welcome Screen or the Site Admin Screen, and clicking the **Inbox** tab to change context. The Inbox will show the WorkItem entry for the page you put into workflow.



8. Select the WorkItem Validate Content, and click **Complete**. In the confirmation dialog box, click **OK**. Notice that the WorkItem disappears from the Inbox.

   You will also notice that the page is now no longer in workflow (the workflow icon is gone from the Status column) and the page has been published or is pending publication (depending on whether you have an AEM publish instance running).



Create a Workflow Implementation Step

A workflow is made up of steps. The steps that define a workflow are participant steps or process steps. Participant steps require manual intervention by a person to advance the workflow. Process steps however are automatic actions that are executed by the system if certain specified conditions are met.

AEM provides a number of predefined process steps that perform common actions, which an administrator can use when building a new workflow. Custom process steps can also be added for tasks not covered by the built-in steps. Process steps, also called automated steps, can be defined by using an ECMA script or a service (a Java class in a bundle). Services can be developed to listen to special workflow events and perform tasks according to the business logic.

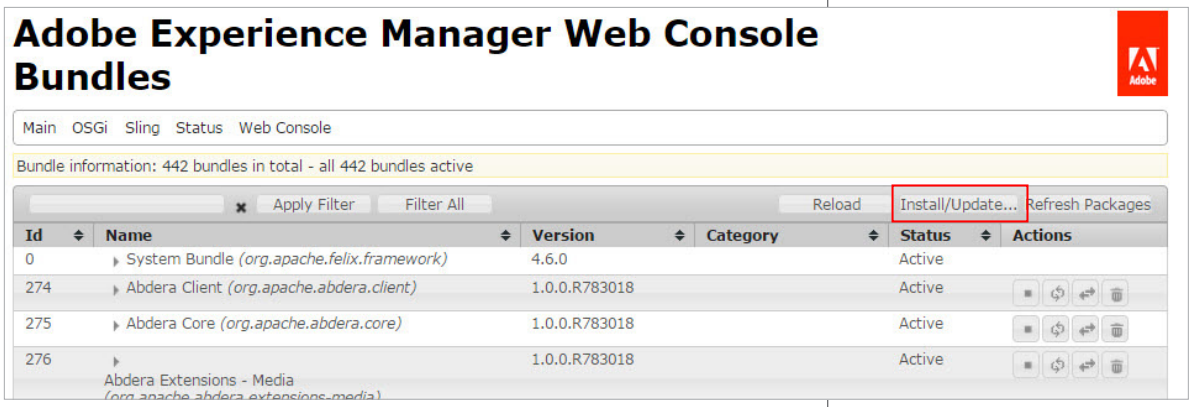
Exercise 12.3
Define a Process Step Using Java

In this exercise, you will install a bundle that creates a process step. Creation of a bundle is beyond the scope of the course. You will use the bundle that is provided in the exercise folder. Following is the Java class that is used to create the bundle.

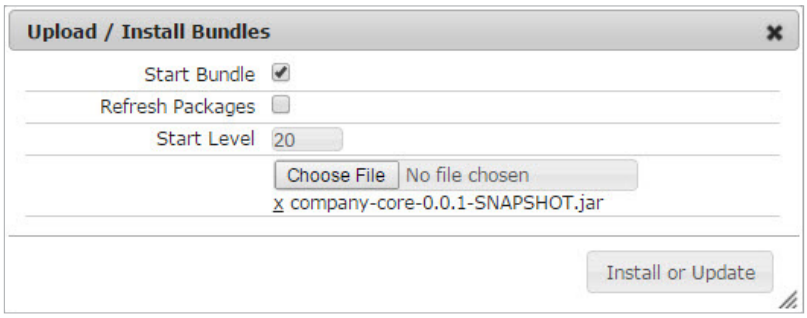
```
package com.mycompany.test;
    import com.day.cq.workflow.WorkflowException;
    import com.day.cq.workflow.WorkflowSession;
    import com.day.cq.workflow.exec.WorkItem;
    import com.day.cq.workflow.exec.WorkflowData;
    import com.day.cq.workflow.exec.WorkflowProcess;
    import com.day.cq.workflow.metadata.MetaDataMap;
    import org.apache.felix.scr.annotations.Component;
    import org.apache.felix.scr.annotations.Properties;
    import org.apache.felix.scr.annotations.Property;
    import org.apache.felix.scr.annotations.Service;
    import org.osgi.framework.Constants;
    import javax.jcr.Node;
    import javax.jcr.RepositoryException;
/**
    Sample workflow process that sets an <code>approve</code> property to the
    payload based on the process argument value.
    */
@Component
    @Service
```

```
@Properties({
        @Property(name = Constants.SERVICE_DESCRIPTION, value = "A sample workflow
process implementation."),
        @Property(name = Constants.SERVICE_VENDOR, value = "Adobe"),
        @Property(name = "process.label", value = "My Sample Workflow Process")})
public class MyProcess implements WorkflowProcess {
    private static final String TYPE_JCR_PATH = "JCR_PATH";
    public void execute(WorkItem item, WorkflowSession session, MetaDataMap args)
throws WorkflowException {
        WorkflowData workflowData = item.getWorkflowData();
        if (workflowData.getPayloadType().equals(TYPE_JCR_PATH)) {
          String path = workflowData.getPayload().toString() + "/jcr:content";
          try {
                Node node = (Node) session.getSession().getItem(path);
                 if (node != null) {
                    node.setProperty("approved", readArgument(args));
                     session.getSession().save();
                }
            } catch (RepositoryException e) {
                throw new WorkflowException(e.getMessage(), e);
            }
        }
    }
    private boolean readArgument(MetaDataMap args) {
        String argument = args.get("PROCESS_ARGS", "false");
        return argument.equalsIgnoreCase("true");
    }
}
```

1. Navigate to the system console.
   http://localhost:4502/system/console
2. In the upper-right corner, click **Install/Update**.



3. Select the **Start Bundle** check box, and click **Choose File**.



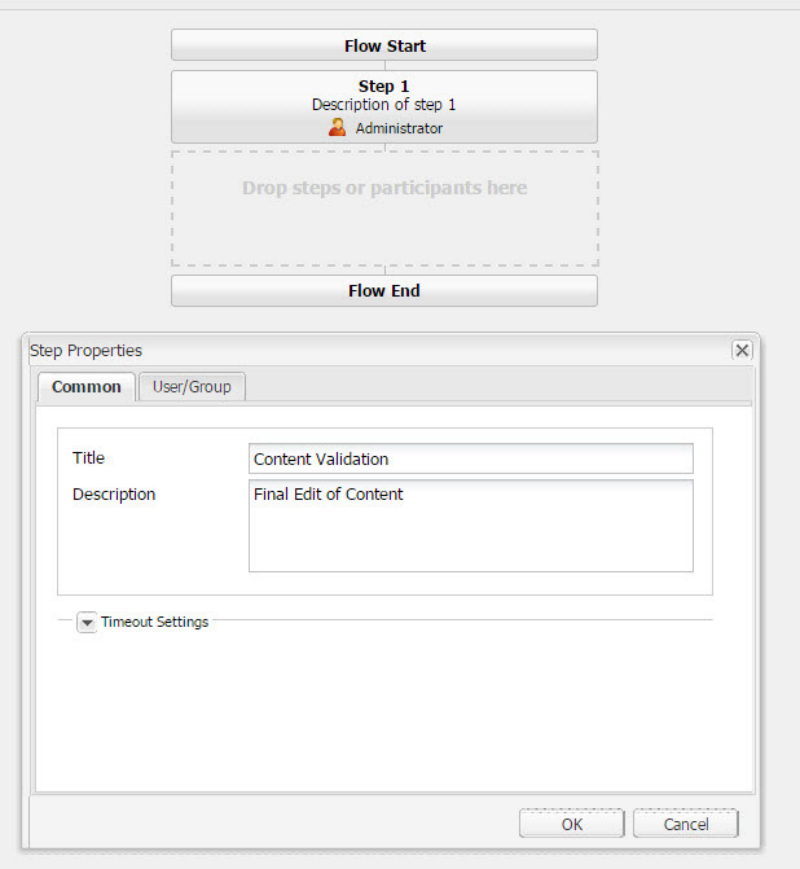
4. Select the bundle from the exercise folder, and click **Install or Update**.
5. In the browser, search for Company Portal, and verify that the Company Portal bundle that is installed.
6. Navigate to the Components tab (**OSGi** > **Components**). In the browser, search for `My Process`. Note that the My Process component is installed.

Exercise 12.4
Implement a Process Step

In this exercise, you will use the process step you created in the previous exercise in a workflow.

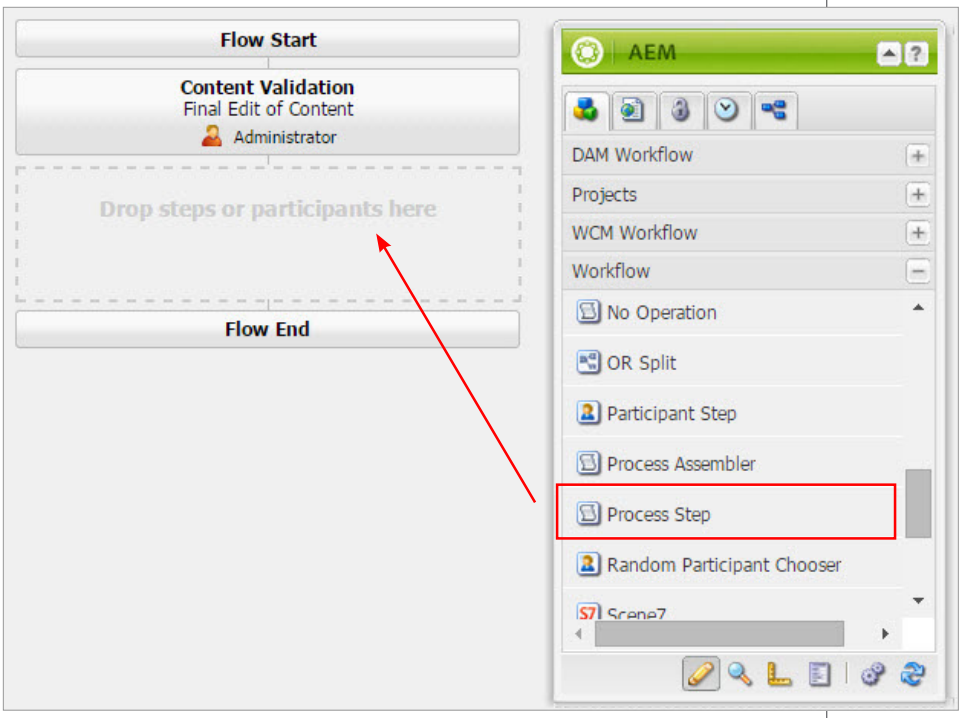
1. Navigate to the Workflow console.
   http://localhost:4502/libs/cq/workflow/content/console.html
2. On the **Models** tab, select **New**. Enter the title as **Approval**.
3. After the workflow is created, double-click it. The new workflow model has Start and End steps.
4. Double-click Step 1. In the **Common** tab, add the following details:

Property	Value
Title	Content Validation
Description	Final Edit of Content



5. Go to the **User/Group** tab and select User/Group as **admin**, click **OK**.

6. Add a process step by dragging and dropping a process step from the sidekick.

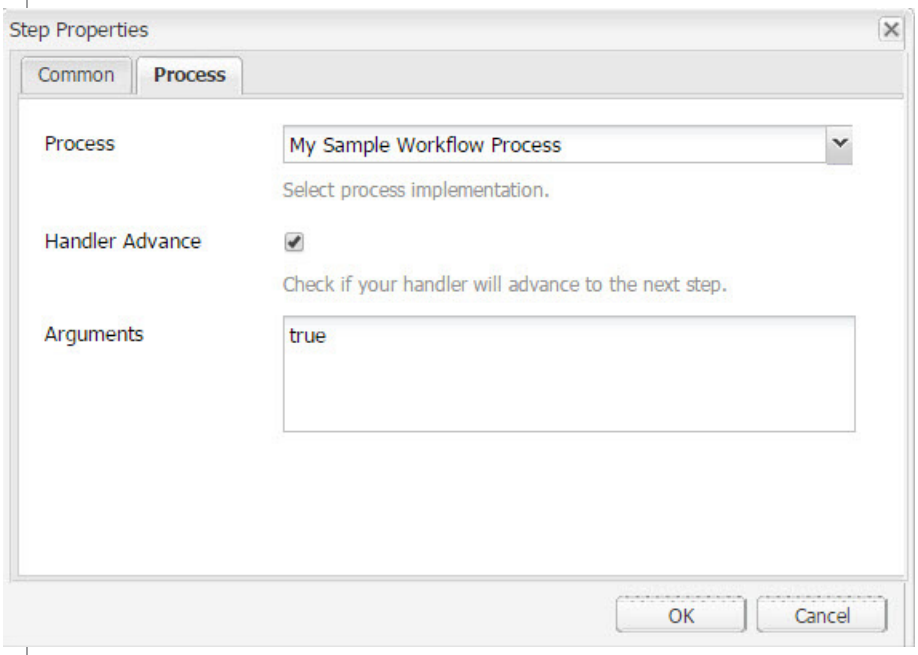


7. Double-click the process step.
8. In the **Common** tab, enter the following details:

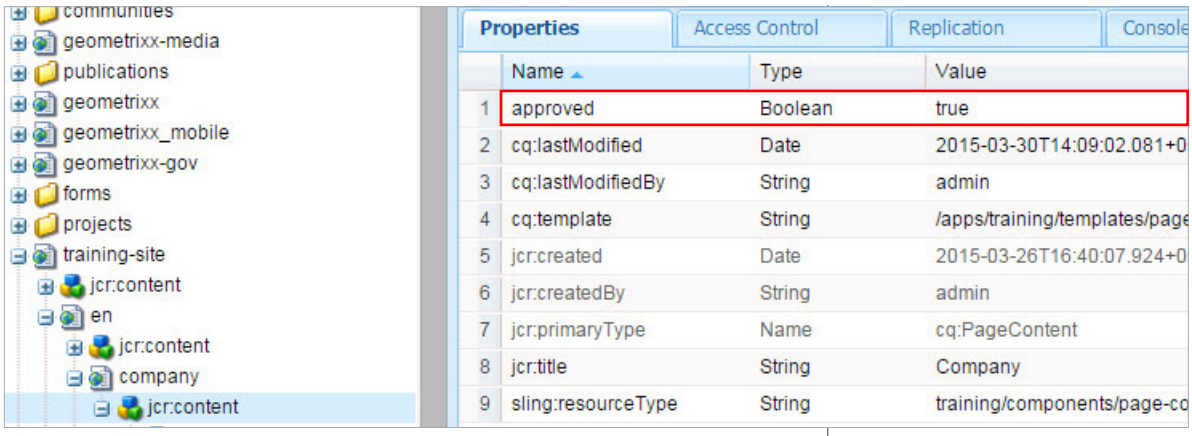
Property	Value
Title	Final Approval
Description	Set the approved property

9. In the **Process** tab, enter the following details, and click **OK**.

Property	Value
Process	My Sample Workflow Process
Arguments	true



10. Select the **Handler Advance** check box, and click **OK**.
11. Ensure that you saved the workflow by clicking the **Save** button on the upper-left corner.
12. Go to Site Admin (http://localhost:4502/siteadmin).
13. Right-click any one of the pages you created in the training site, and select **Workflow**.
14. Select the **Approval** workflow from the drop-down list, and click **Start**.
15. Click the **Inbox** icon at the top.
16. Select the workflow you created. Right-click, and click **Complete**. In the confirmation dialog box, click **OK**.
17. Go to CRXDE Lite, and navigate to the page. Note that the approved property is added to the page.



AEM Environment

Overview

This chapter helps you with the installation and setup of an AEM environment on your system. It also provides some basic information about package managers and tools that you would find useful during the development of your pages.

Objective

In this chapter, you will learn how to:

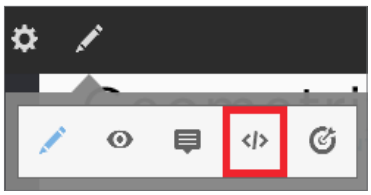
- create packages using Package Manager.
- access some useful tools in the touch-optimized UI.
- deploy AEM.

Useful Tools

The following parameters will help you monitor what is being rendered:

Developer Mode in Touch-Optimized UI

You can access the Developer tab in the touch-optimized UI as shown below:

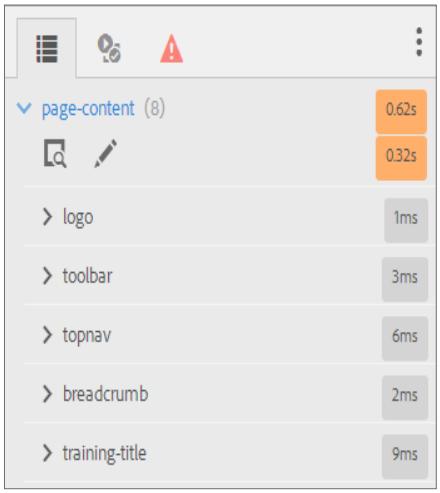


The Developer mode provides you with the following options:

- Components
- Error
- Tests

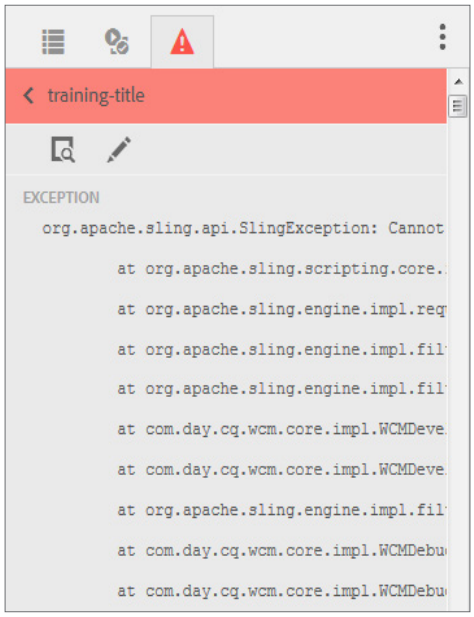
Components

The Components tab provides you with a components tree of used components on the page. For each component, it provides you with the associated script. You can click the Edit icon to navigate directly to the script in CRXDE Lite. The component tree also provides you with the server-side computation time for the component. It helps you to identify the slow and heavy components that need further optimization.



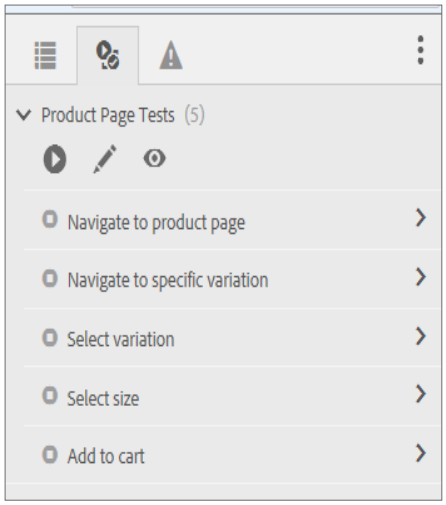
Error

The Error tab displays the error details of the broken components. You can see the required details without inspecting the log files.



Tests

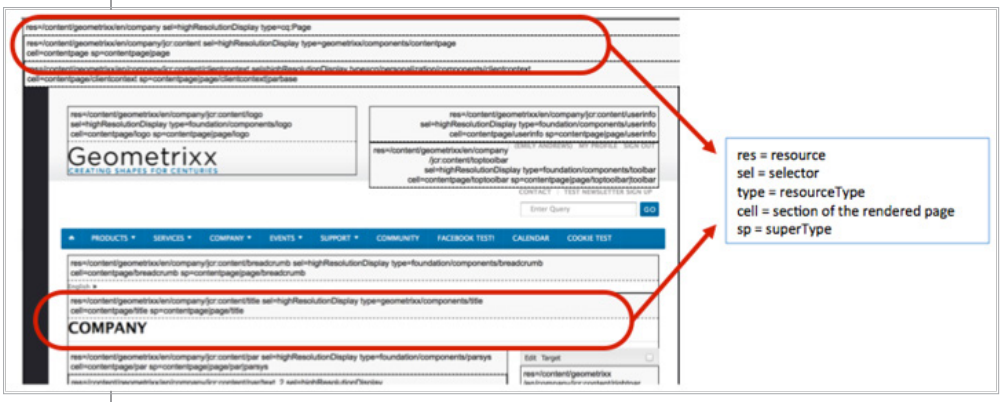
The Tests tab allows you to write and execute tests directly in AEM. You can use *Hobbes.js* as the functional UI Testing framework. You can create various test suites and add test cases. Test cases can also be run in the Demo mode. The Demo mode is slow compared to the actual Run mode. You can also click the Edit button to navigate to the scripts that you wrote for the tests. The tab also displays the test results on the same page.



?debug=layout

Use this parameter to get information about renderers, selectors, resources, and so on.

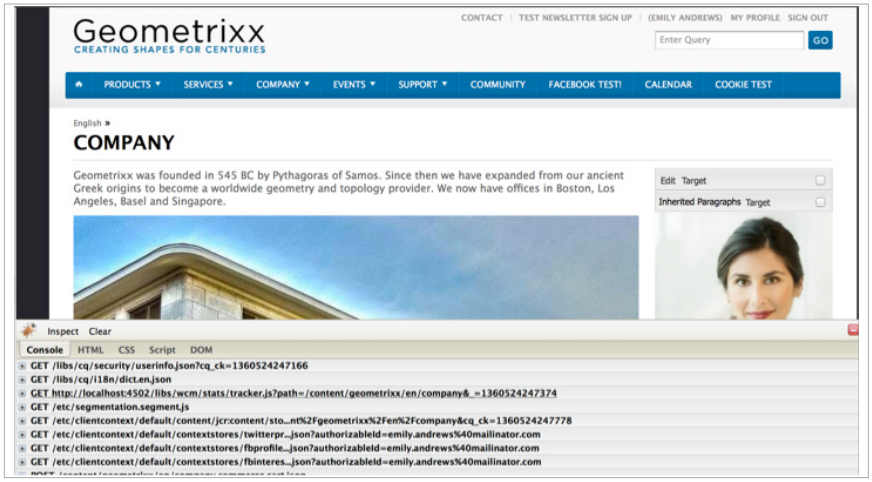
For example:
http://localhost:4502/content/geometrixx/en/company.html?debug=layout



?debugConsole=true

Use this parameter to open the onboard Firebug Console (even when using browsers that do not support Firebug, or in cases where AEM-rendering interferes with Firebug).

For example:
http://localhost:4502/geometrixx/en/company.html?debugConsole=true



?debugClientLibs=true

Use this parameter to have the list of client libraries loaded onto the page shown in the source.

For example:
http://localhost:4502/content/geometrixx/en/company.html?debugClientLibs=true

```
<!DOCTYPE HTML>
<html>
<head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8"
    <meta name="keywords" content="Business, Investor">
    <meta name="description" content="">
    <meta http-equiv="Last-Modified" content="11 Nov 2010 10:34:16 EST
<script type="text/javascript">
GraniteClientLibraryManager.write([{
    "p": "/etc/clientlibs/granite/jquery.js?debug=true",
    "c": [
    ]
  },{
    "p": "/etc/clientlibs/granite/utils.js?debug=true",
    "c": [
    ]
  },{
    "p": "/etc/clientlibs/granite/jquery/granite.js?debug=true",
    "c": [
    ]
  },{
    "p": "/etc/clientlibs/foundation/jquery.js?debug=true",
    "c": [
    ]
  },{
    "p": "/etc/clientlibs/foundation/shared.js?debug=true",
    "c": [
    ]
  },{
    "p": "/etc/clientlibs/foundation/personalization/jcarousel.css?deb
    "c": [
    ]
  },{
    "p": "/etc/clientlibs/foundation/personalization/jcarousel.js?debu
    "c": [
    ]
  },{
    "p": "/etc/clientlibs/foundation/main.css?debug=true",
    "c": [
    ]
  },{
    "p": "/etc/clientlibs/foundation/main.js?debug=true",
    "c": [
    ]
  },{
    "p": "/etc/designs/geometrixx/clientlibs.css?debug=true",
    "c": [
    ]
  },{
    "p": "/etc/designs/geometrixx/clientlibs.js?debug=true",
    "c": [
    ]
  },{
```

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.

Exercise 13.1
Monitor Page Response

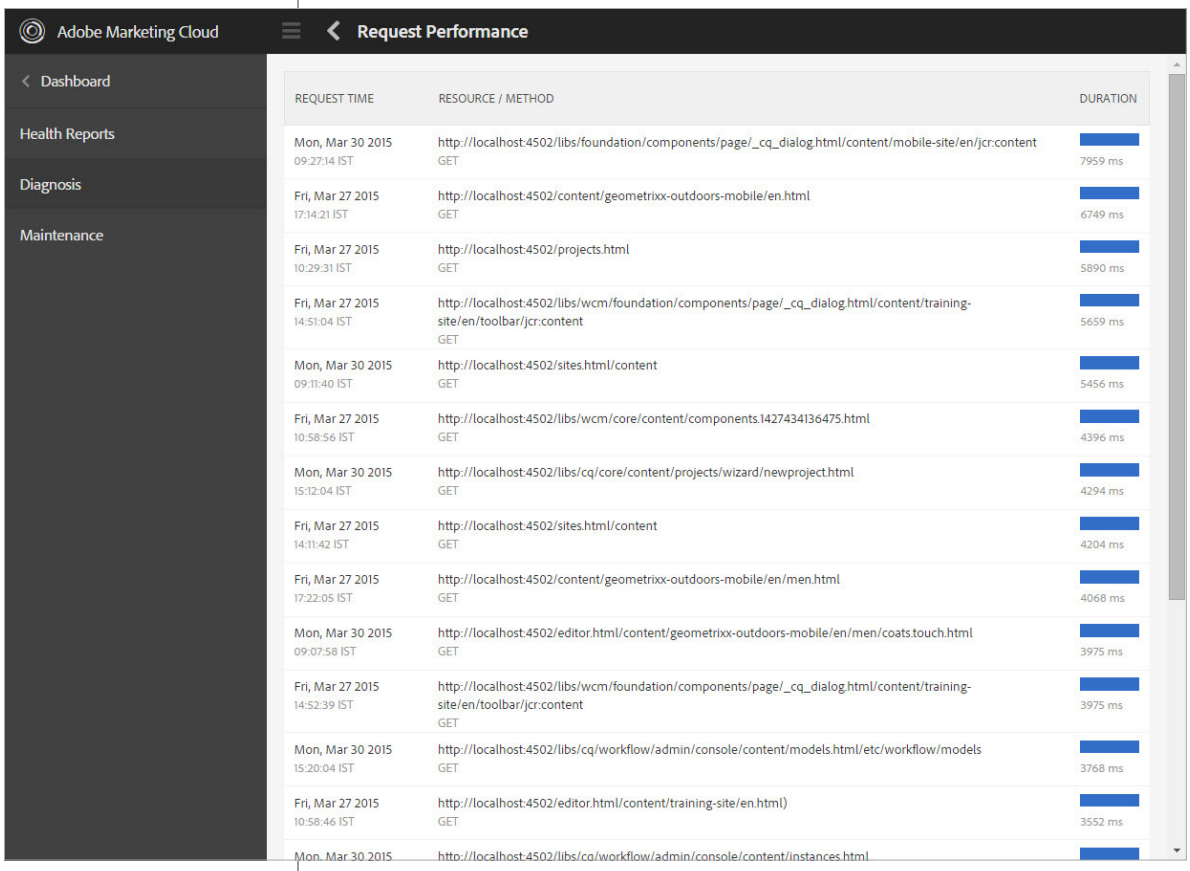
1. Navigate to and open the file `request.log` located at `<cq-install-dir>/crxquickstart/logs`.
2. Request a page in author that utilizes your Training Template and components.
   For example, */content/training-site/en/company*
3. Review the response times directly related to the previous step's request. (A page request of the page: /content/training-site/en/company)

```
+0530 [9447] <- 200 application/json 33ms
+0530 [9448] -> GET /bin/wcm/siteadmin/tree.json?_dc=1427687878880&ncc=100&_charset_=utf-8&path=%2Fcontent%2Ftraining-site&node=xnode-821 HTTP/1.1
+0530 [9448] <- 200 application/json 19ms
+0530 [9449] -> GET /bin/wcm/siteadmin/tree.json?_dc=1427687879087&ncc=100&_charset_=utf-8&path=%2Fcontent%2Ftraining-site%2Fen&node=xnode-824 HTTP/1.1
+0530 [9449] <- 200 application/json 14ms
+0530 [9450] -> GET /content/training-site/en/products.pages.json?_dc=1427687879189&start=0&limit=30&predicate=siteadmin HTTP/1.1
+0530 [9451] -> GET /content/training-site/en/products.pages.json?_dc=1427687879194&start=0&limit=30&predicate=siteadmin HTTP/1.1
+0530 [9452] -> GET /bin/wcm/siteadmin/tree.json?_dc=1427687879196&ncc=100&_charset_=utf-8&path=%2Fcontent%2Ftraining-site%2Fen%2Fproducts&node=xnode-8

+0530 [9452] <- 200 application/json 13ms
+0530 [9450] <- 200 application/json 36ms
+0530 [9451] <- 200 application/json 30ms
+0530 [9453] -> GET /bin/wcm/siteadmin/tree.json?_dc=1427687880656&ncc=100&_charset_=utf-8&path=%2Fcontent%2Fmobile-site&node=xnode-823 HTTP/1.1
+0530 [9453] <- 200 application/json 12ms
+0530 [9454] -> GET /content/mobile-site.pages.json?_dc=1427687881584&start=0&limit=30&predicate=siteadmin HTTP/1.1
+0530 [9454] <- 200 application/json 27ms
+0530 [9455] -> GET /libs/cq/ui/widgets/themes/default/widgets/wcm/SiteAdmin/status-livecopy.gif HTTP/1.1
+0530 [9455] <- 304 - 20ms
+0530 [9456] -> GET /home/users/d/dJVofCBI3fK7-ztmtSLn.permissions.json?path=%2Fcontent&_charset_=utf-8&cq_ck=1427687881694 HTTP/1.1
+0530 [9456] <- 200 application/json 25ms
+0530 [9457] -> GET /home/users/d/dJVofCBI3fK7-ztmtSLn.permissions.json?path=%2Fcontent%2Fmobile-site&_charset_=utf-8&cq_ck=1427687883101 HTTP/1.1
+0530 [9457] <- 200 application/json 9ms
+0530 [9458] -> GET /libs/foundation/components/page/dialog.overlay.infinity.json HTTP/1.1
+0530 [9458] <- 200 application/json 14ms
+0530 [9459] -> GET /libs/foundation/components/page/tab_basic.overlay.infinity.json HTTP/1.1
+0530 [9459] <- 200 application/json 14ms
+0530 [9460] -> GET /libs/foundation/components/page/tab_advanced.overlay.infinity.json HTTP/1.1
+0530 [9460] <- 200 application/json 14ms
+0530 [9461] -> GET /libs/foundation/components/page/tab_image.overlay.infinity.json HTTP/1.1
+0530 [9461] <- 200 application/json 13ms
+0530 [9462] -> GET /libs/foundation/components/page/tab_cloudservices.overlay.infinity.json HTTP/1.1
+0530 [9462] <- 200 application/json 9ms
+0530 [9463] -> GET /libs/foundation/components/page/tab_blueprint.overlay.infinity.json HTTP/1.1
+0530 [9463] <- 200 application/json 12ms
+0530 [9464] -> GET /libs/foundation/components/page/tab_livecopy.overlay.infinity.json HTTP/1.1
+0530 [9464] <- 200 application/json 10ms
+0530 [9465] -> GET /libs/wcm/msm/components/rolloutconfig/chooser.overlay.infinity.json HTTP/1.1
+0530 [9465] <- 200 application/json 10ms
```

You successfully reviewed the response time of a page in AEM 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.

Exercise 13.2
Find the Response Performance

1. Log in to AEM Sites.
   http://localhost:4502
2. Go to **Tools** > **Operations** > **Dashboard** > **Diagnosis** > **Request Performance**. The Request Performance page appears. The page displays requests in the order of duration taken.



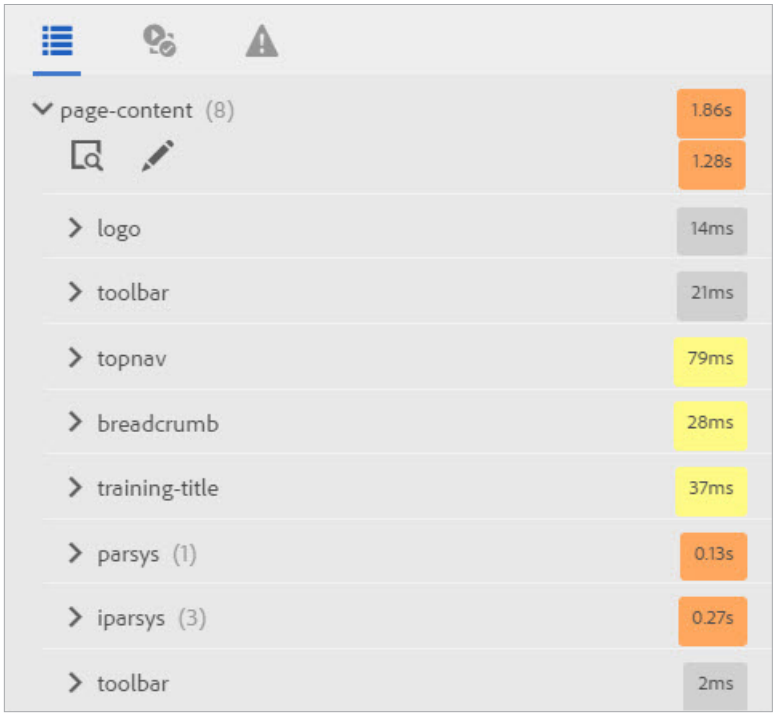
You can use this information to further investigate the cause of long responses.

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.

Exercise 13.3
Monitor Component-based Timing

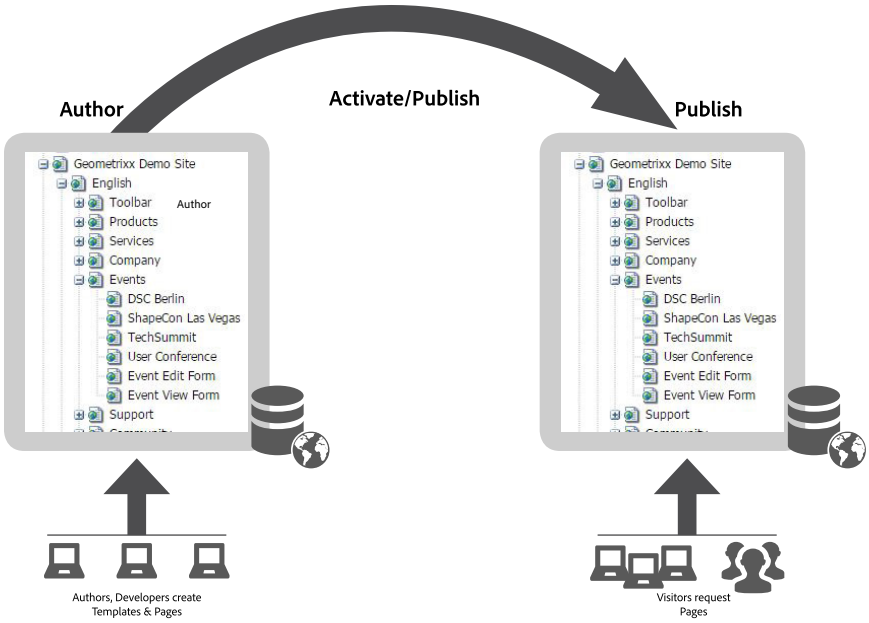
In the Developer mode of the page, you can view the time taken by the components to load.

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.



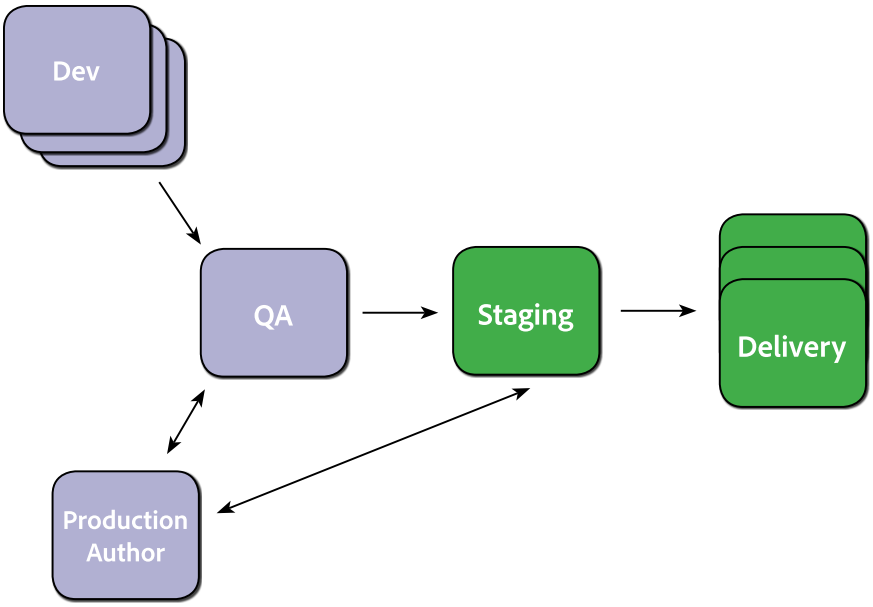
AEM Deployment

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.



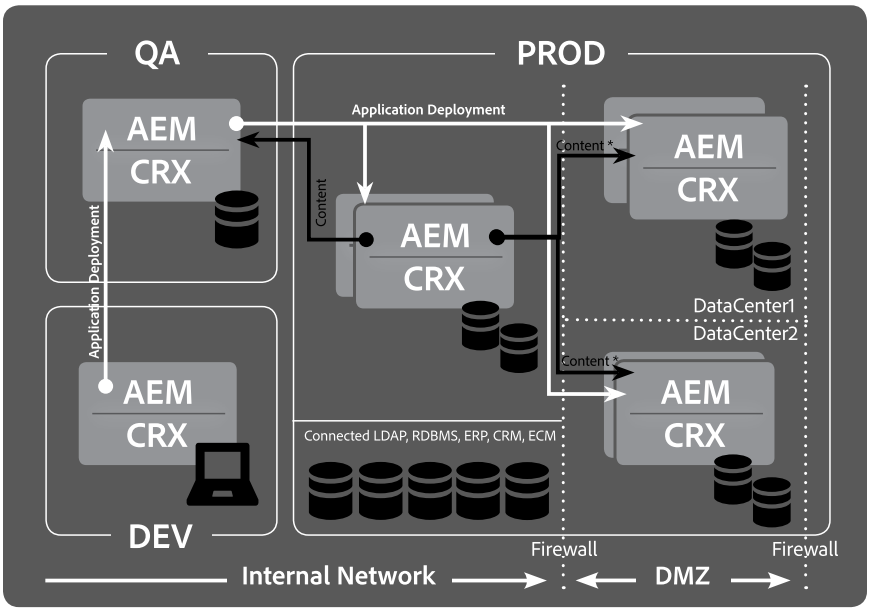
Depending on the scale of a project, a production environment may consist of several author and/or publish instances, and at a lower level, the CRX repository may be clustered among several instances as well. Additionally, separate development and test environment levels may also consist of author and publish instances, mirroring the production environment to a varying extent.

There are two types of AEM installations—Author and Publish. Each AEM 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.



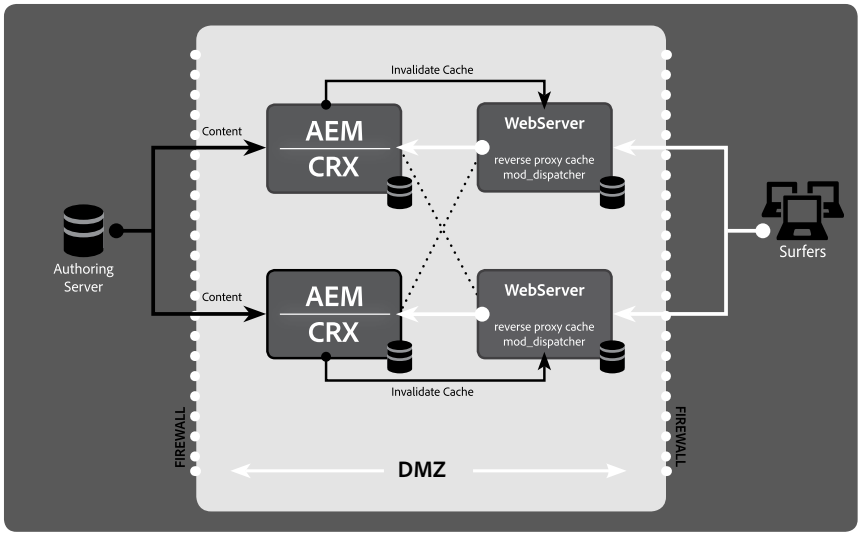
Content and code often move in different directions and get to their destination through different mechanisms. Content typically is moved by use of the Replication Agents. Content will move forward from Author to Publish, but also may move back to QA. This allows testing to be done on real data.

Deployment of code and configuration information is typically done through automated processes that use content packages, Replication Agents, and/or FileVault.



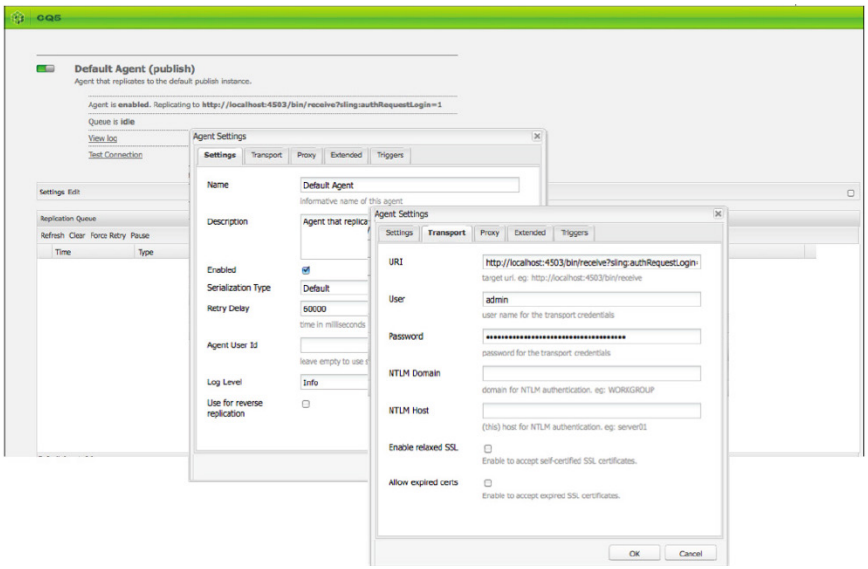
NOTE: Although this is clearly the response time of a page while in author, it is a good practice to be able to identify where response times are located (request.log) and how to identify a specific page request/response in the log files itself. Ideally, this test would occur in the publish instance to get a better idea of its expected behavior in production.

There are many designs for deployment of the AEM application itself. One of them is pictured below:



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.

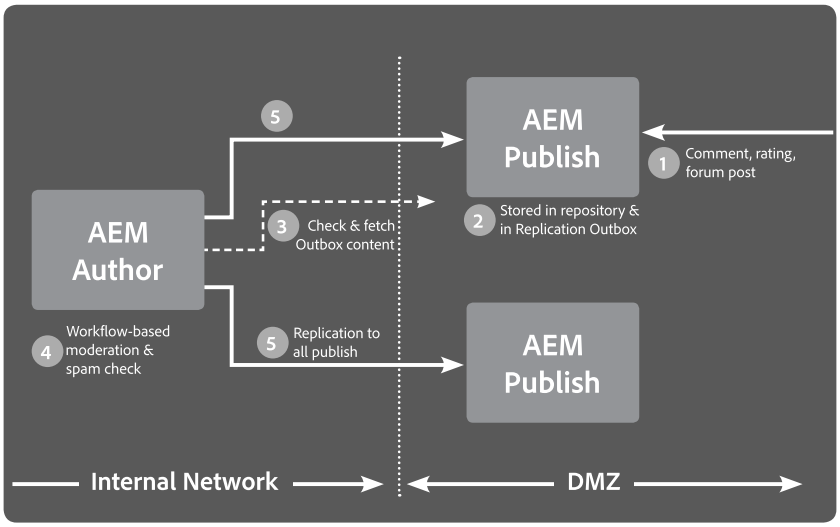


Details on the creation and management of Replication Agents can be found in the documentation at:

http://docs.adobe.com/docs/en/aem/6-0/deploy/configuring.html

Reverse Replication

Reverse Replication allows the transfer of content from the website visitors (the publish instance) to the author instance for review, moderation, or spam check without violating any firewall rules.



AEM

Agents on author

Default Agent (publish)
Agent that replicates to the default publish instance.

Agent is **enabled.** Replicating to **http://localhost:4503/bin/receive?sling:authRequestLogin=1**

Queue is **blocked - 5 pending**

Dispatcher Flush (flush)
Agent that sends flush requests to the dispatcher.

Agent is **disabled.** Replicating to **http://localhost:8000/dispatcher/invalidate.cache**

Queue is **not active**

Agent is triggered when on-/offtime reached

Reverse Replication Agent (publish_reverse)
Agent that retrieves reverse replicated content from the default publish instance's outbox.

Agent is **enabled.** Replicating to **http://localhost:4503/bin/receive?sling:authRequestLogin=1**

Queue is **idle**

Agent is ignored on normal replication

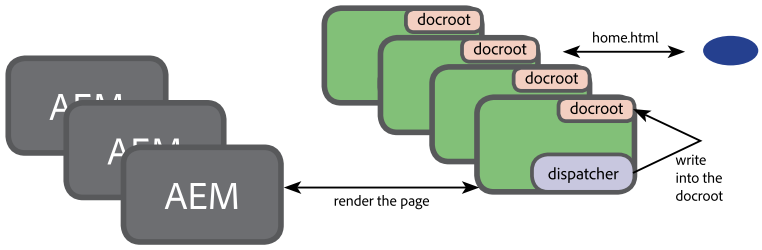
Static Agent (static)
Agent that stores a static representation of a node into the filesystem.

Agent is **disabled.** Replicating to **static:///**

Queue is **not active**

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:

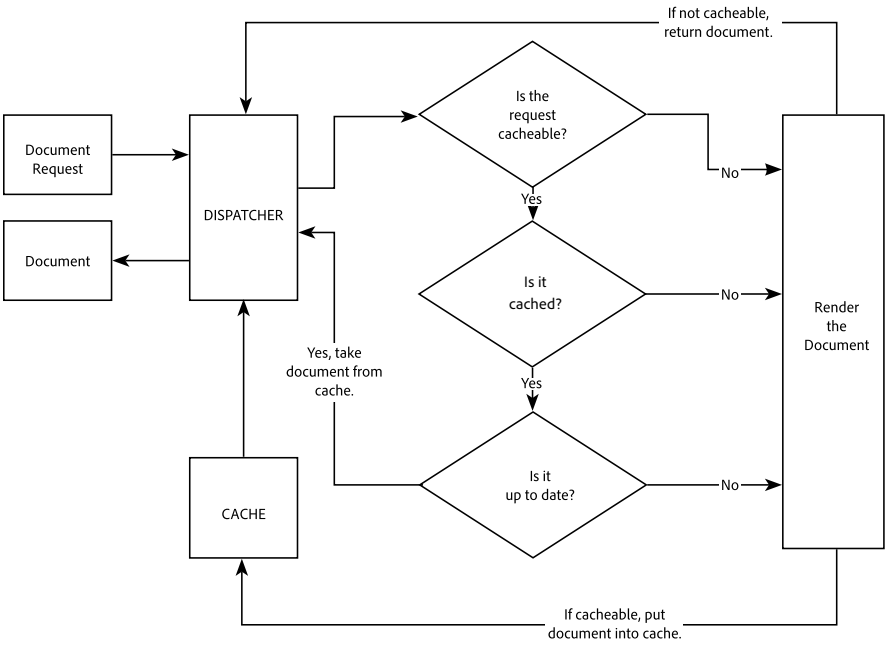
- storing (or caching) as much of the site content as possible, in the form of a static website
- accessing the layout engine as little as possible

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

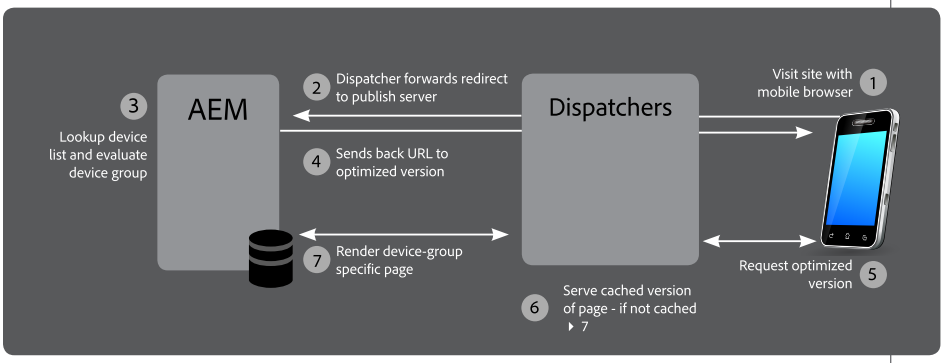
The Dispatcher contains mechanisms to generate and update static HTML, based on the content of the dynamic site. You can specify in detail which documents are stored as static files and which are always generated dynamically. The figure below demonstrates the algorithm that the Dispatcher uses to determine whether an object should be served from the web server cache or rendered dynamically.

NOTE: If anything went wrong during the publishing process, check the author instance for the related Replication Agent log files using the configuration panel you used because you edited the settings.



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.



Extract for Brackets (Extra Credit)

Overview

Brackets' new Extract functionality is a great tool for web designers and front-end developers as they can easily move from design to development and speed up the workflow.

Extract for Brackets speeds up the process of extracting design information such as colors, font styles, measurements, and assets out of a Photoshop Document (PSD) and display them as clean, minimal CSS through contextual code hints. You can also extract layers as images, use information from the PSD to define preprocessor variables, and easily get dimensions between objects.

Prerequisites

As Extract for Brackets is a service provided by the Creative Cloud, you need to ensure that you have the following:

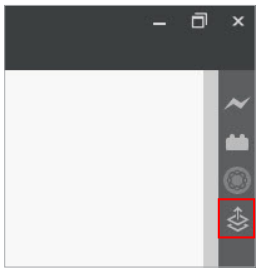
- a connection to the internet
- an account with the Creative Cloud
- a PSD with multiple layers

Setting up the Development Environment

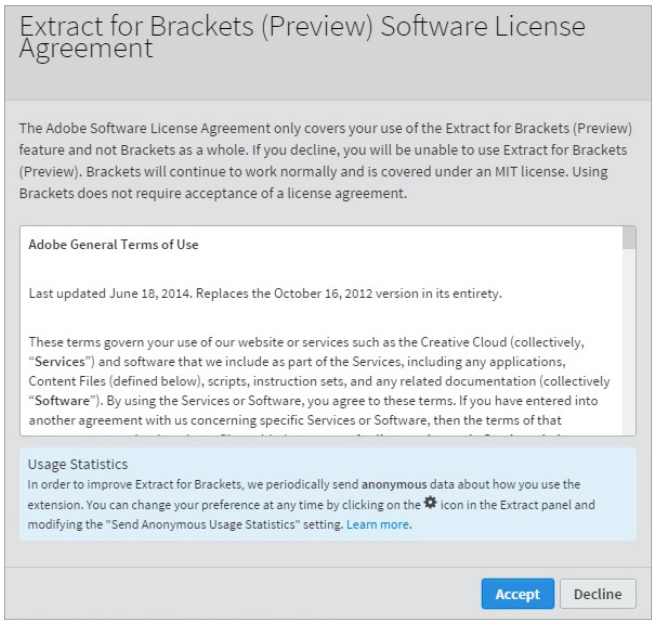
In an earlier exercise, you had already installed Brackets. The current version of Brackets includes the Extract extension, so you do not have to install the extension separately.

Exercise 14.1
Configure the Extracts Extension

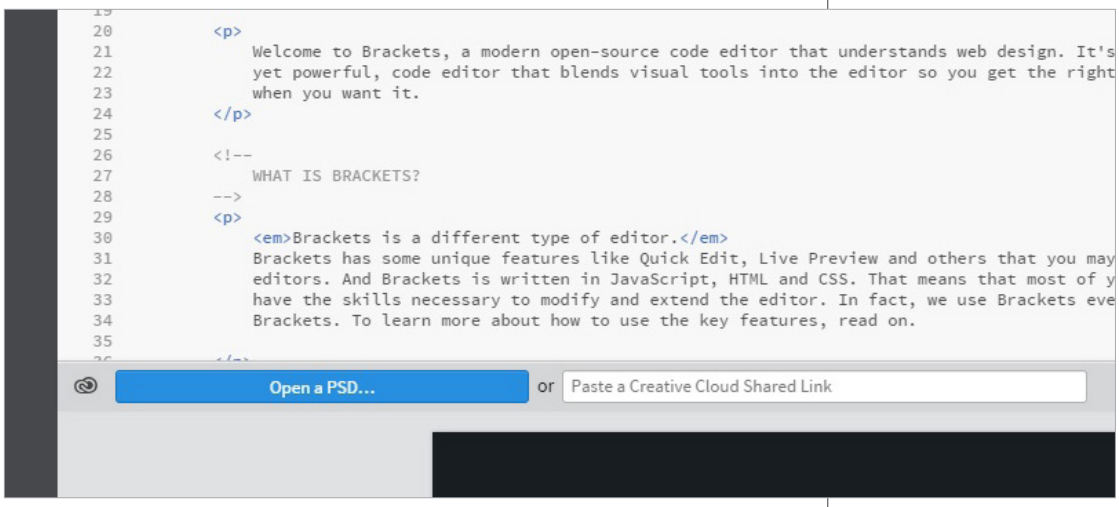
1. Open **Brackets**.
2. In the toolbar on the right, click the **Extract for Brackets (Preview)** icon.



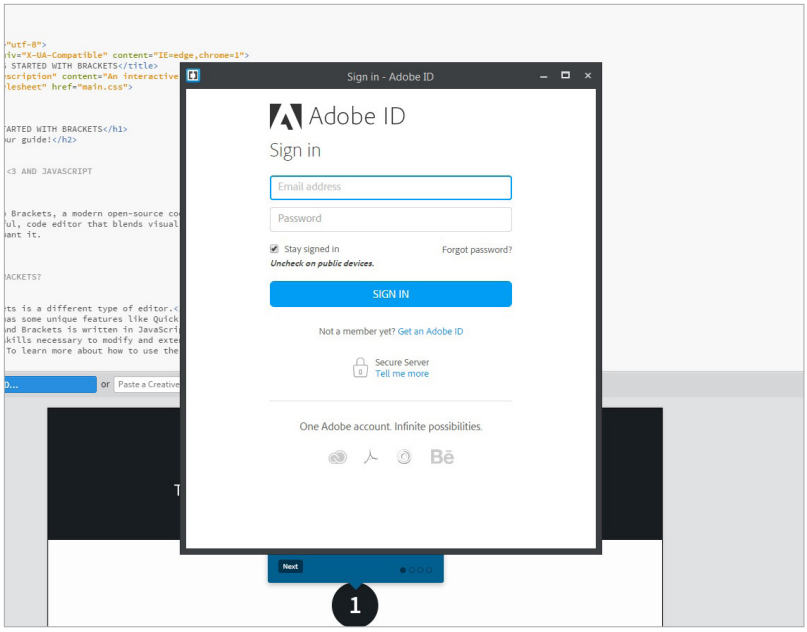
3. Click **Accept** to accept the Software License Agreement.



4. Click **Open a PSD...**.

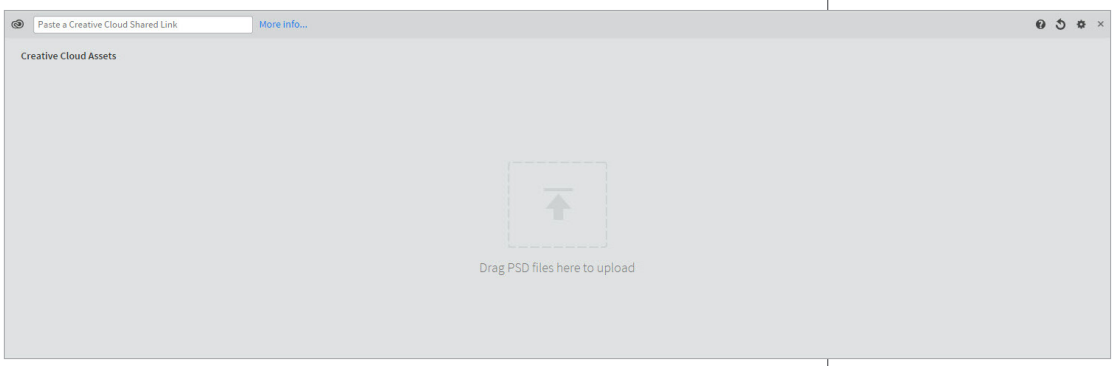


5. Sign in to the Creative Cloud with your Adobe ID.



NOTE: If you do not already have an Adobe ID, you can create one for free.

You are now ready to work with the PSD files.



Extracting from a PSD

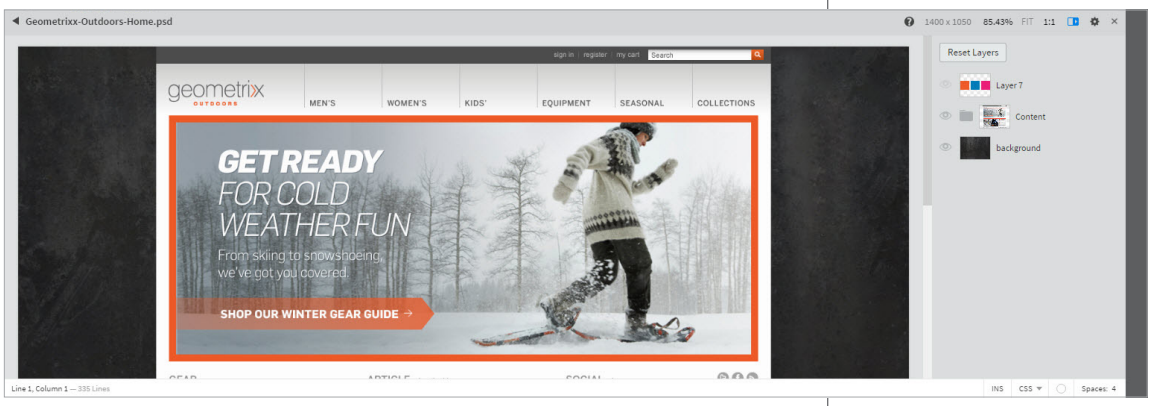
You have already configured the Extract for Brackets extension in the previous exercise. The following sections use Geometrixx Outdoors to demonstrate how content and styles are extracted from a PSD file.

Exercise 14.2
Modify Geometrixx Outdoors Page by Extracting Content From PSD

You already downloaded the Geometrixx Outdoors project files in a previous exercise and modified the contents using Brackets. (Exercise 5.4). The goal of the following exercise is to change the background of the Geometrixx Outdoors page by extracting the background from the PSD file provided in your USB contents.

1. In Brackets, open the Geometrixx Outdoors project. To do this, go to **File** > **Open Folder**. Browse and select the jcr_root folder.
2. Configure the **Project Settings** in the **AEM** menu.
3. Open the **Extract for Brackets (Preview)** window. The icon is located on the right toolbar as shown in the following figure.

4. Click and drag the `Geometrixx-Outdoors-Home.psd` file from the USB contents into the **Extract for Brackets (Preview)** window. This uploads the file into the Creative Cloud Assets folder. All the layers of the PSD file are available in the right panel.

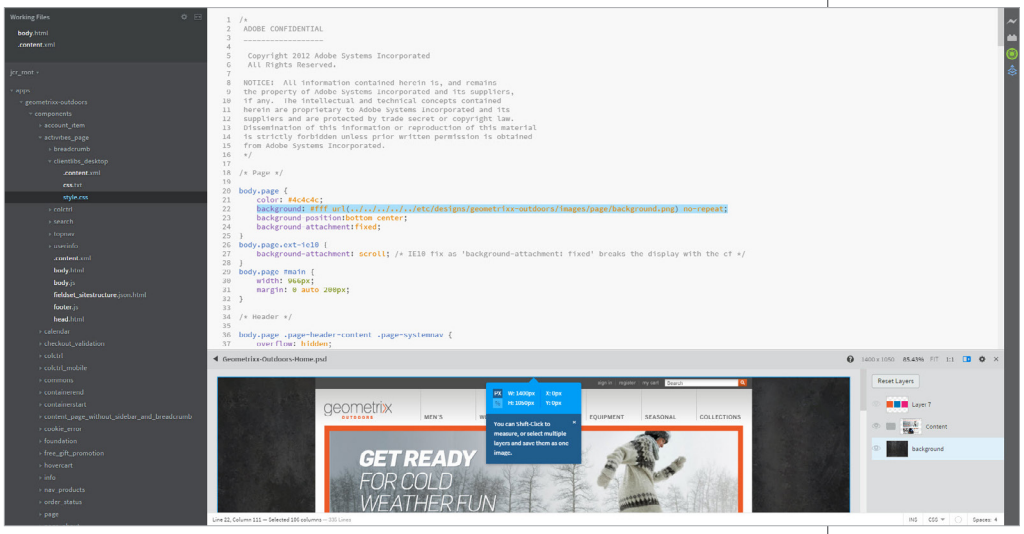


5. Select the last layer called "background" in the right panel.

6. From the left panel, navigate to `apps/geometrixx-outdoors/components/activities _ page/clientlibs _ desktop/style.css`.

7. Open the `style.css` file and locate the following line (line 22) that sets the background of the page. You need to apply the dark background of the PSD to the Biking page of Geometrixx Outdoors.

```
background: #fff url(../../../../../etc/designs/geometrixx-outdoors/images/page/background.png) no-repeat;
```



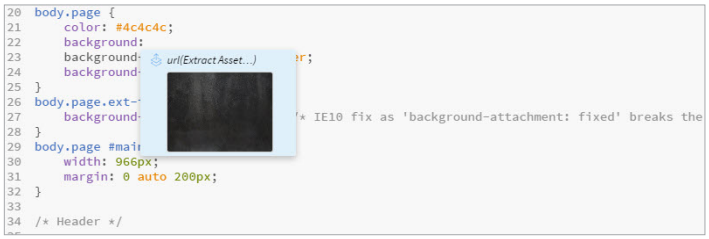
8. Select and copy the path within the paranthesis. That is, `../../../../../etc/designs/geometrixx-outdoors/images/page/background.png`. This is the path where the image is stored, and you need to keep a copy of this path so that you can use it later.

9. Delete the colon and everything that follows on that line (that is, delete the text : `#fff url(../../../../../etc/designs/geometrixx-outdoors/images/page/background.png) no-repeat;`).
When you are done, it should look as shown below.

```
body.page {
    color: #4c4c4c;
    background
    background-position:bottom center;
    background-attachment:fixed;
}
```

10. Type a colon after background, followed by a space. You are prompted with the asset URL. Select the **url(Extract Asset...)** in the auto-suggestion menu.

    This option is an extract of the layer from the PSD file. Selecting this option enables you to insert the selected object of the PSD layer into the CSS file.



NOTE: If required, you can change the name of the file. For example, you can replace the text `background` with `dark _ background`.

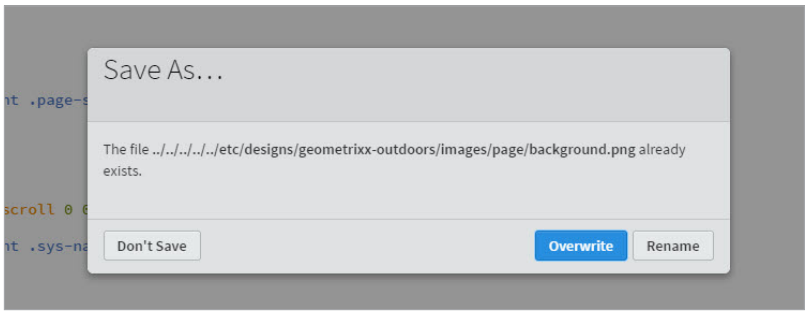
11. Replace the `images/background.png` text with the path that you copied earlier. You paste the same path as was existing before you made the changes so that the new image is stored in the same location in your project.
Press **Enter** on your keyboard or click outside the line to set the changes.

    What you have just done in the above steps is, you have replaced the existing image with the one extracted from the PSD layer.

```
body.page {
    color: #4c4c4c;
    background: url(../../../../../etc/designs/geometrixx-outdoors/images/page/background.png)
    background-position:bottom center;
    background-attachment:fixed;
}
```

NOTE: Images extracted from the PSD layers are stored in your project folder. For this exercise, you can see the image in `/etc/designs/geometrixx-outdoors/images/page`

If a file already exists in your project folder with the same name (that is, background.png), you will get a confirmation dialog box asking you to overwrite the file. Click **Overwrite**.



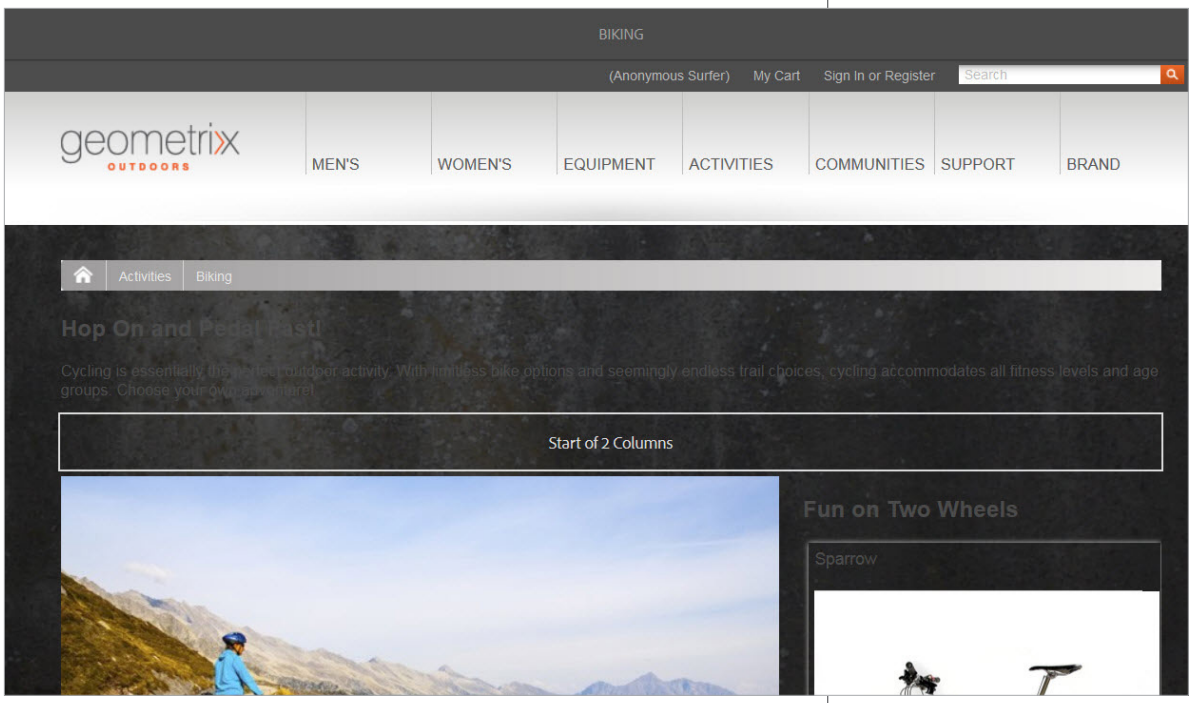
12. Just before the URL, add `#fff`, and at the end of the line, add `repeat`. The line should now read as:

    ```
    background: #fff url(../../../../../etc/designs/geometrixx-outdoors/images/page/background.png) repeat;
    ```

13. Save your changes.

14. These changes are immediately reflected on AEM. To confirm, go the Geometrixx Outdoors page or click the following link: http://localhost:4502/editor.html/content/geometrixx-outdoors/en/activities/cajamara-biking.html

    You will notice that the dark background is applied to the page.



NOTE: If you do not see the expected output, please check your CSS once more. Also, try clearing your browser cache.

15. As the text is still dark and not visible, you need to apply a white background to the body. In line 31, under the `body.page #main` section, add the following line:

    `background: #fff;`

```
20  body.page {
21      color: #4c4c4c;
22      background: #fff url(images/background.png) repeat;
23      background-position:bottom center;
24      background-attachment:fixed;
25  }
26  body.page.ext-ie10 {
27      background-attachment: scroll; /* IE10 fix as 'background-attachment: fixed' breaks the display with the cf */
28  }
29  body.page #main {
30      width: 966px;
31      background: #fff;
32      margin: 0 auto 200px;
33  }
34
35  /* Header */
```

16. Save your changes. Refresh the AEM page and see the changes applied. You will see a page similar to the following: