# Adobe® Experience Manager Sites: Developer

## Student Workbook

ADOBE® TRAINING SERVICES

AEM Sites: Developer



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



201409

# Table of Contents

# 01

# Getting Started

This chapter provides you with an introduction to AEM. You will learn the following:

- Installation and deployment
- User interfaces
- Various web console

# Introduction to AEM

Adobe® Experience Manager helps you organize, create, 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, more securely share and track personalized client documents, 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 API 2.5 (or higher). AEM comes pre-configured with its own built-in servlet engine, but can also be installed in any compatible third-party application server.

Since the system can be accessed from any computer with a modern web browser, no installation of client software is required; so, even 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 web site, which is simply a collection of documents that reside on the server, and that can be viewed through a browser over the web. A web application, on the other hand, typically assembles the data to be sent back to the browser, on the fly with each request (or even using AJAX, between requests). Most modern web sites nowadays 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, in order to install and start the server.
For more details, see Supported Platforms. (http://dev.day.com/docs/en/cq/current/deploying/technical_requirements.html)

Since AEM is Java-based, it can run on any system for which a Java Runtime Environment is available. This means, for example, 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:

- Administer the web properties
- Input your content
- Configure the layout and design your content
- Activate 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—content authors will login to and manage pages. This includes creating, editing, deleting, moving, etc. In addition, it is the instance run mode, you will use for development, as you can easily observe both 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 web site and interact with it, be it public or within your intranet.

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

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

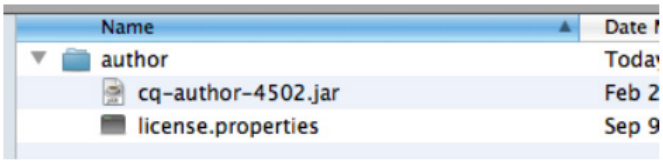
## Installing AEM

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:

- CRX repository (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. Both CRX and AEM WCM ship with 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, as the applications have been installed and the repository is already created.

### EXERCISE 1.1 - Install AEM

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

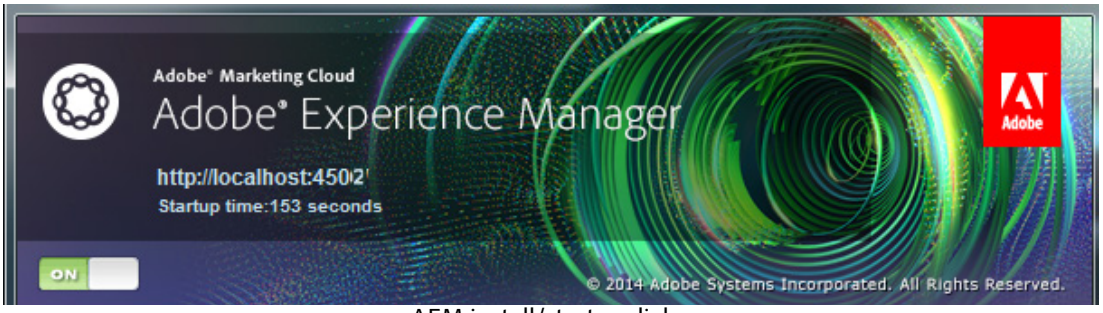



AEM author install folder structure

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

3. Rename the AEM quickstart JAR to ***cq-author-4502.jar***.
   - cq = the application
   - author = the WCM mode it will run in (e.g. author or publish)
   - 4502 = the port it will run in (any available port is acceptable)
4. In a Windows or MacOS X environment, you can simply double-click the ***cq-author-4502.jar*** file.

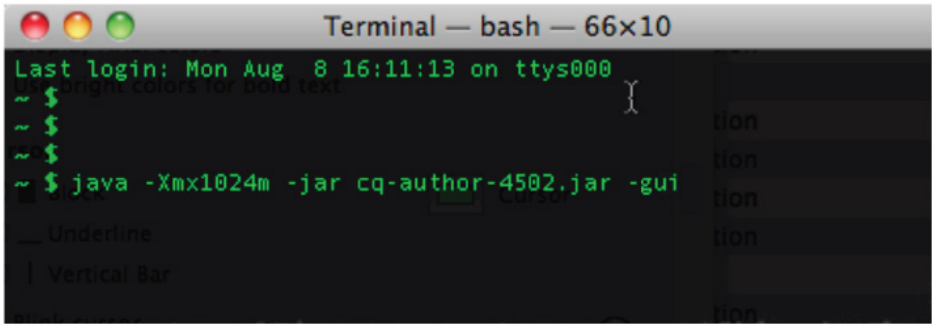
› Installation will take approximately 5-7 minutes, depending on your system's capabilities
› A dialog will pop-up similar to the one below:




AEM install/startup dialog

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.

There are two ways to install AEM: graphical and by command line. The latter is more powerful since the user has the possibility to provide additional performance-tuning parameters to the 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 image below:




AEM command line start

Typical command line start:

```
$ java -Xmx1024m -jar cq-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, etc) and workload (content, requests, etc). For now, it will be sufficient to know that, you can start your instance (either author or publish) using the following parameters:

**-Xms** --> assigns initial heap size

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

-**Xmx** --> assigns the maximum heap size

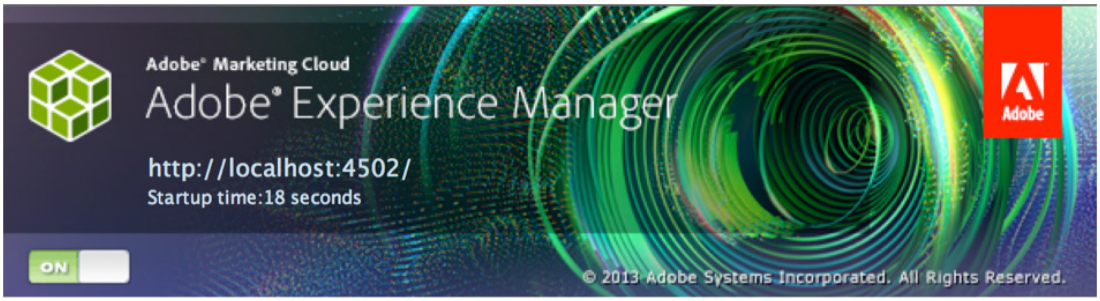
| | |
|---|---|
| Default value | 64MB for a JVM running on 32bit machines, or 83MB for 64bit machines |
| Recommended | Specific to physical memory available and expected traffic, but should be<br>equal or greater than the initial size. To run CQ5 it's recommended to<br>allocate at least 1024MB of heap size. |
| Syntax | **-Xmx1024m** (sets the maximum heap size to 1024MB) |

**-XX:MaxPermSize** --> assigns the heap to hold reflective data of the VM (e.g.Java objects)

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

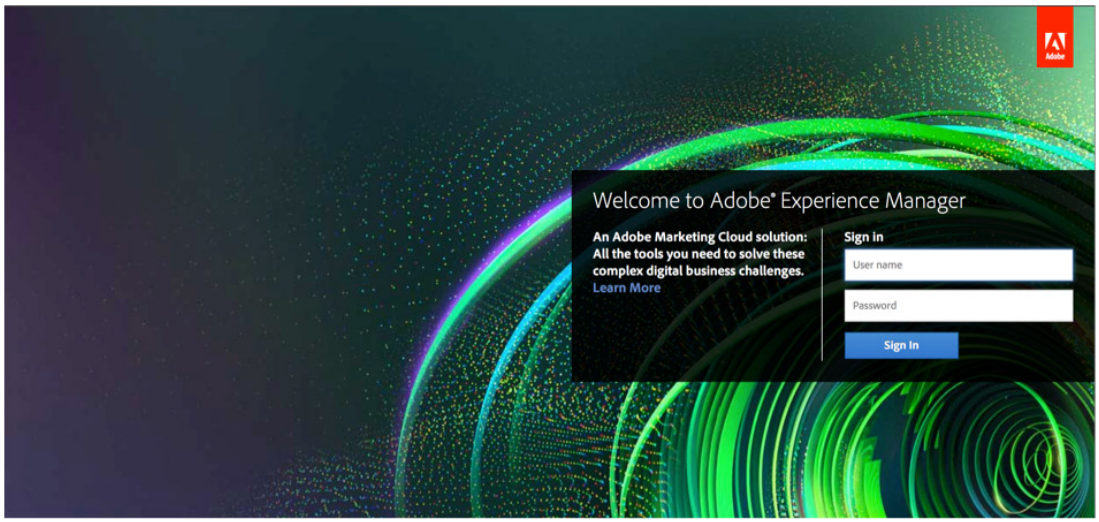
You can now install or start AEM from the command line together with increasing the Java heap and perm gen size, which will improve performance.

Once AEM has started successfully, the dialog will change to something similar to the following:




AEM startup dialog

In addition, once the AEM is started, your default browser will automatically open to AEM's start URL (where the port number is the one you defined on installation), e.g., http://localhost:4502.



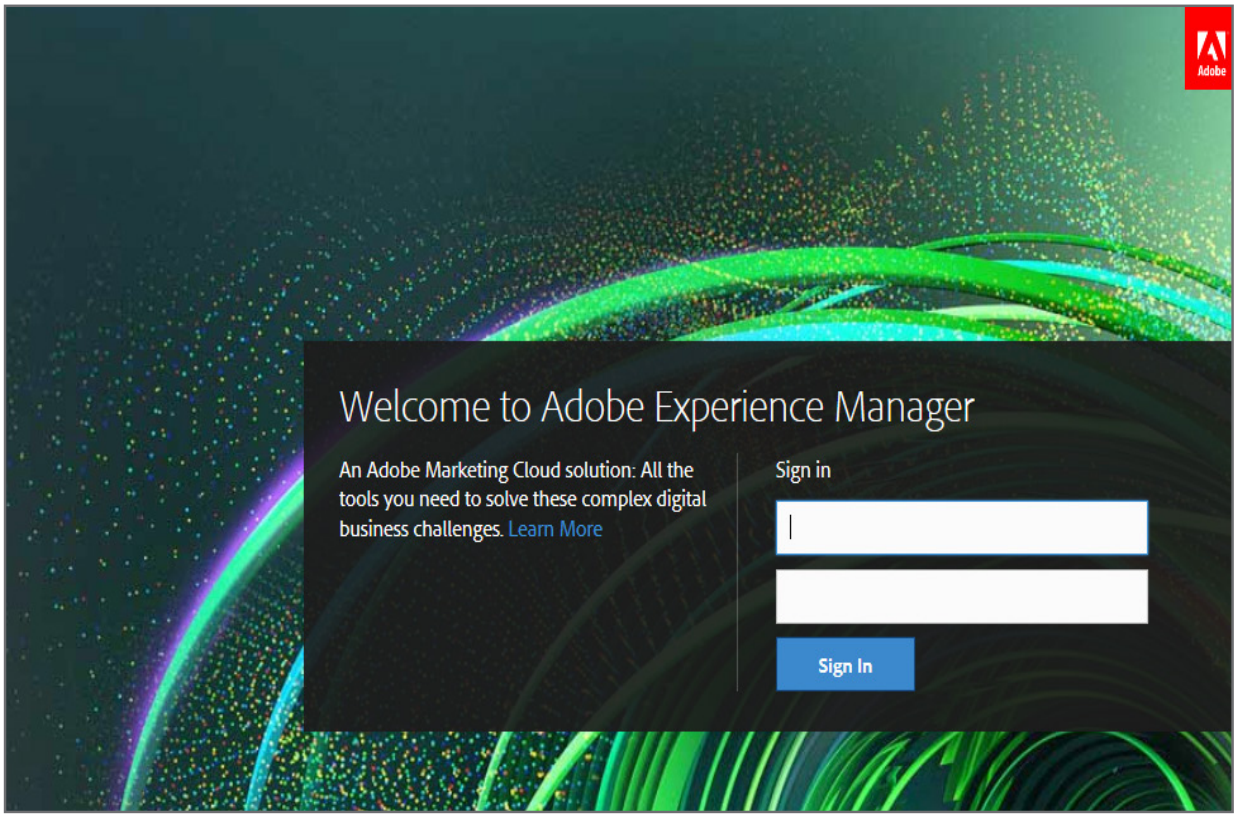

Use the following command to install AEM, without installing the Geometrixx sites:

```
java -jar AEM-author-p4502.jar -r author, nosamplecontent
```

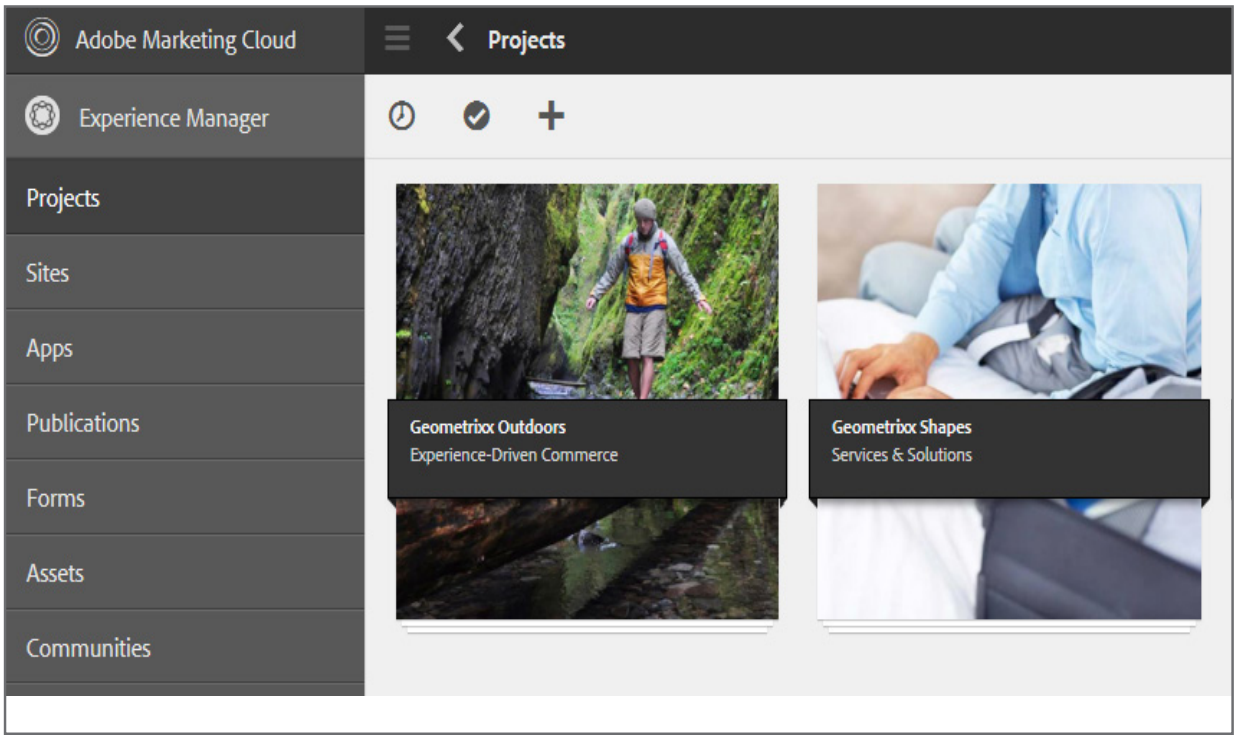
You have now successfully installed and started AEM. To start AEM in the future, double-click the renamed AEM quickstart JAR file (e.g. cq-author-4502.jar).

### EXERCISE 1.2 - Log in to AEM

1. After you start AEM, 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 by default. The Sites section displays various sites you have created.




## The AEM User Interfaces

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

- Projects
- Sites: Displays all existing sites. These sites are the sample implementation that you can refer to when you work with your actual sites.
- Publications
- Forms: Allows you to create forms using Adobe LiveCycle.
- Assets: Provides you with your digital assets. It was formerly known as DAM.
- Communities: Provides you with various communities' support.
- Tools: Provides you with various tools to manage operations and assets. It also provides you a code inspection tool named CRXDE Lite and a web console.

## Authoring in AEM

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

- Classic UI—provides you with a desktop-like UI
- Touch-optimized UI—provides you UI that is optimized to use in various hand-held 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 of time, 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.

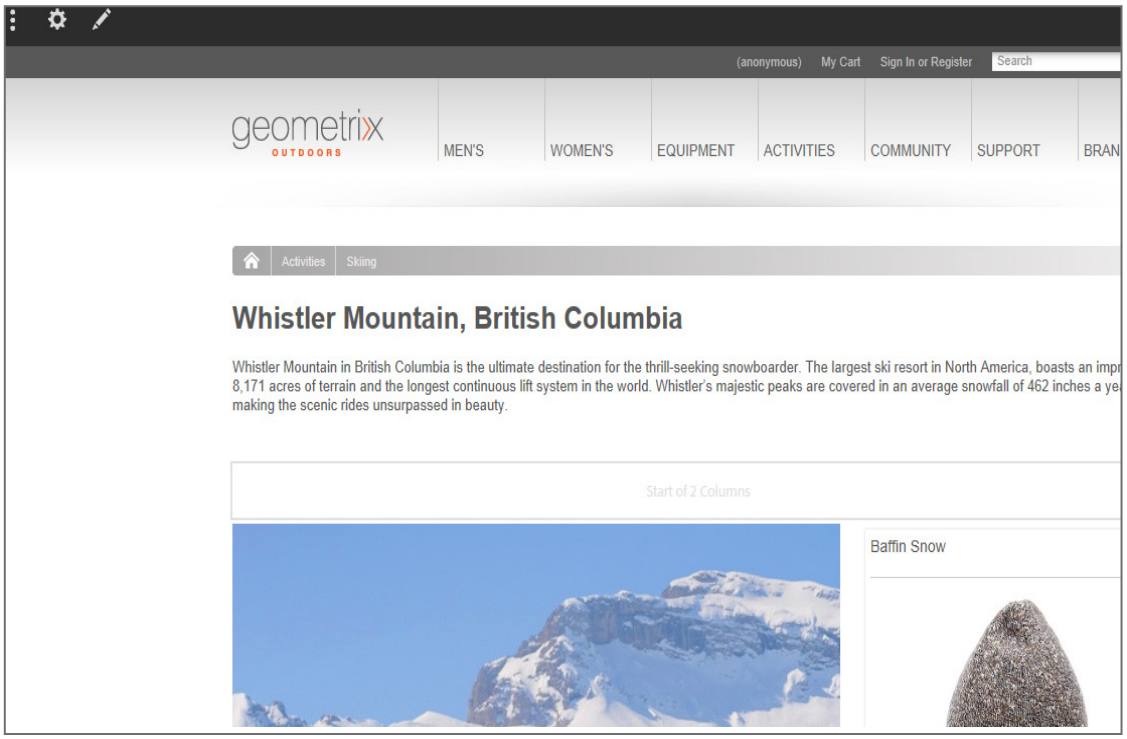
### Touch-Optimized UI

By default, when you open a page, it appears in the Touch-Optimized UI.

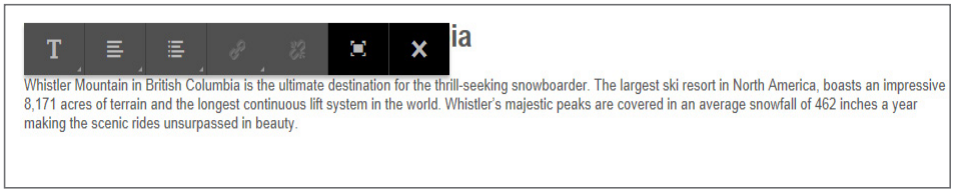
**EXERCISE 1.3 - Open a page in Touch-Optimized UI**

1. Log into Site Admin.
   *http://localhost:4502/siteadmin*
2. Using the left navigation pane, go to **Geometrixx Outdoors Site** > **English** > **Activities** > **Skiing**.

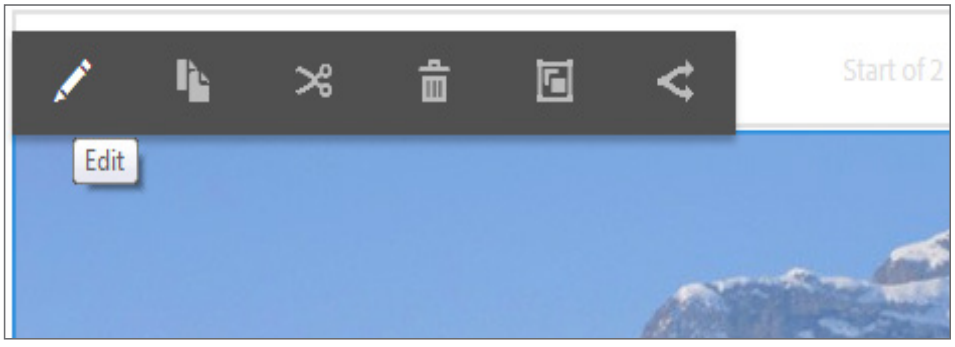
3. Double-click the Skiing page.
   By default, the page appears in the Touch-Optimized UI.



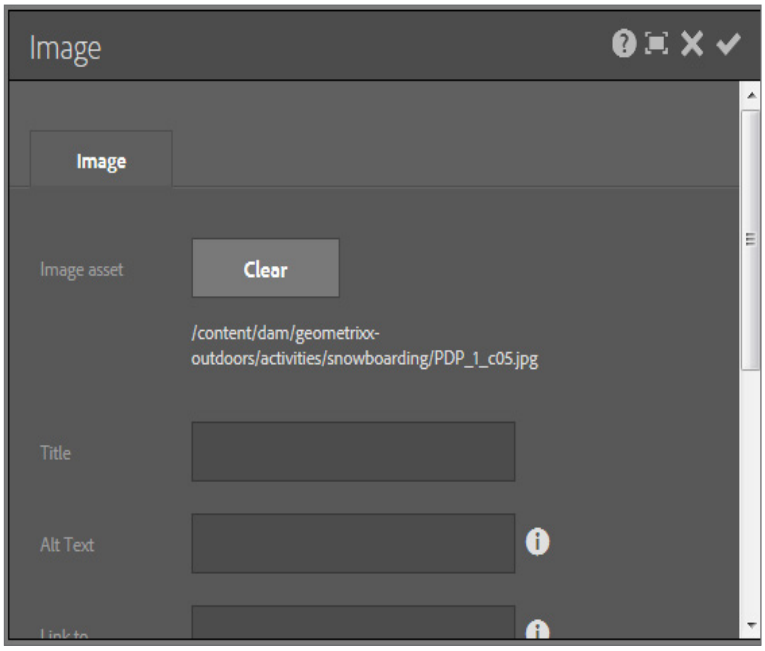

4. Double-click the paragraph that displayed below the web page's title.
   The paragraph becomes editable where you can update the text.



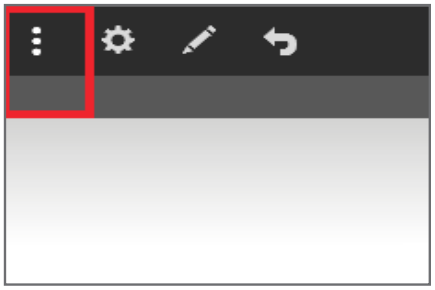

5. Select the image displayed by clicking it, and then click the **Edit** button.



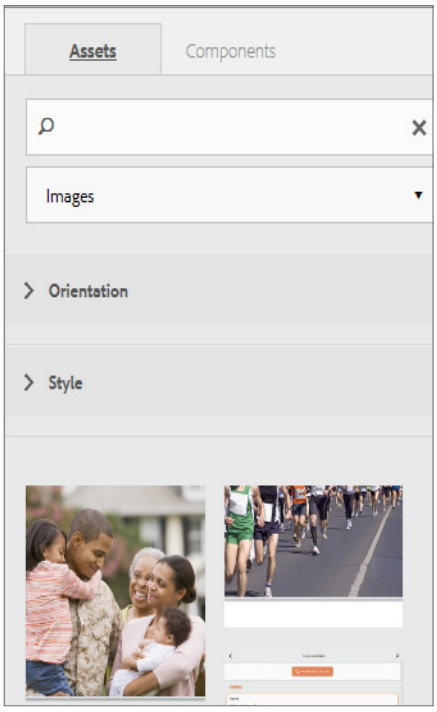
Note that the Edit dialog appears. You can update various properties of the image in the dialog. Click the check mark (or the X icon) to close the window.



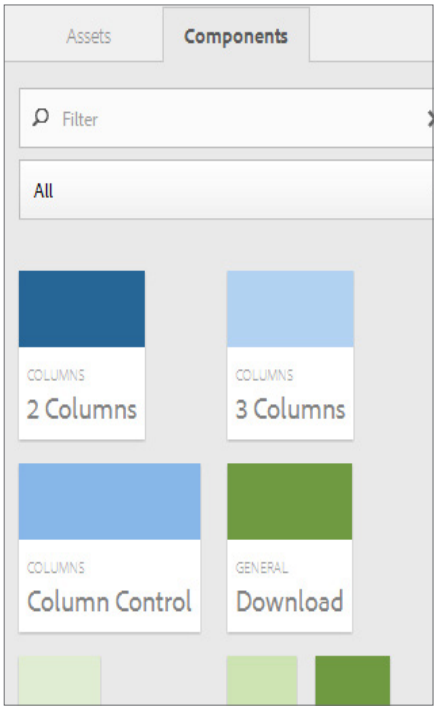

6. Click the **Toggle** button in the top-left corner.



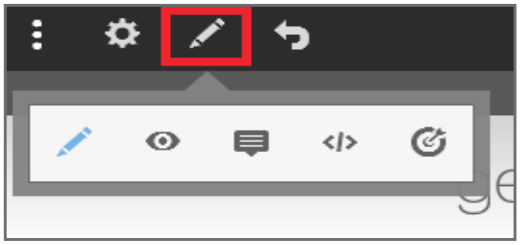
The **Assets** tab appears in the screen as shown below:




7. Click the **Component** tab.
   The Component tab appears with available components.

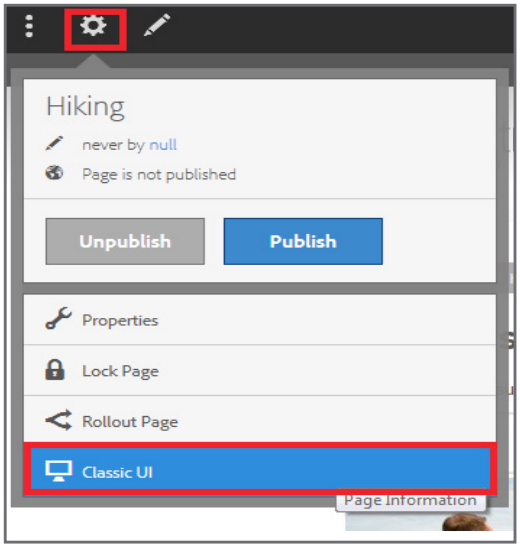


8. Click the **Edit** button on the top bar to select any of the following mode:



   - Edit: Default mode—allows you to edit the page.
   - Preview: allows you to preview the page.
   - Annotate: Allows you to select a specific location of the site and add an annotation.
   - Developer mode: 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.
   - Target: Allows you to target your content for Adobe Target.

9. Click the **Page Information** icon.
   The Page Properties dialog box appears.




10. Click Classic UI to see the page in Classic UI.

### Developer mode

The Developer mode provides you with the following options:

- Components: Displays the name of the page component used for the web page. Also, provides you a list of components used in the page. For each component, the component script and the content path are provided along 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: This tab allows you to run GUI test cases. You can write test cases using CRXDE Lite and run them here. It provides you the result page indicating the test cases that failed or passed.

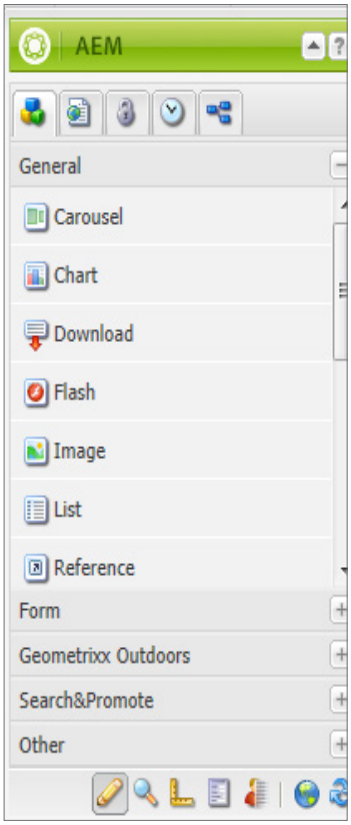
- Error: Provides with the errors in the web page.

### 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**: A floating "inspector" window, sometimes known as a floating palette, 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:

- Web sites classic console for creating and managing multiple web sites.
- Web sites touch-friendly console for creating and managing web site 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/sytem/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 do standard development tasks. It is recommended when there is no direct access to 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 JSR

# 02

# OSGi Framework

In this chapter, you will learn the following:

- OSGi and Apache Sling
- AEM functional building blocks
- Granite platform
- OSGi framework
- OSGi bundles

# 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 a JCR content repository through an HTTP-based REST API. Both AEM's native functionality and the functionality of any web site built with AEM are delivered through this framework.

## Clustering

In computing, a cluster is a group of computers linked together to work in some respects as a single computer. Every CRX instance comes pre-configured to run within a cluster, even when running a singular instance. This design feature allows the configuration of multi-node clusters with little effort.

# AEM functional building blocks

AEM consists of sets of OSGi bundles that are deployed on the CRX. The AEM application modules use the JCR 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, multi-site 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, as they are tightly integrated. Application functionality that is not used or not licensed can be disabled after the initial install.

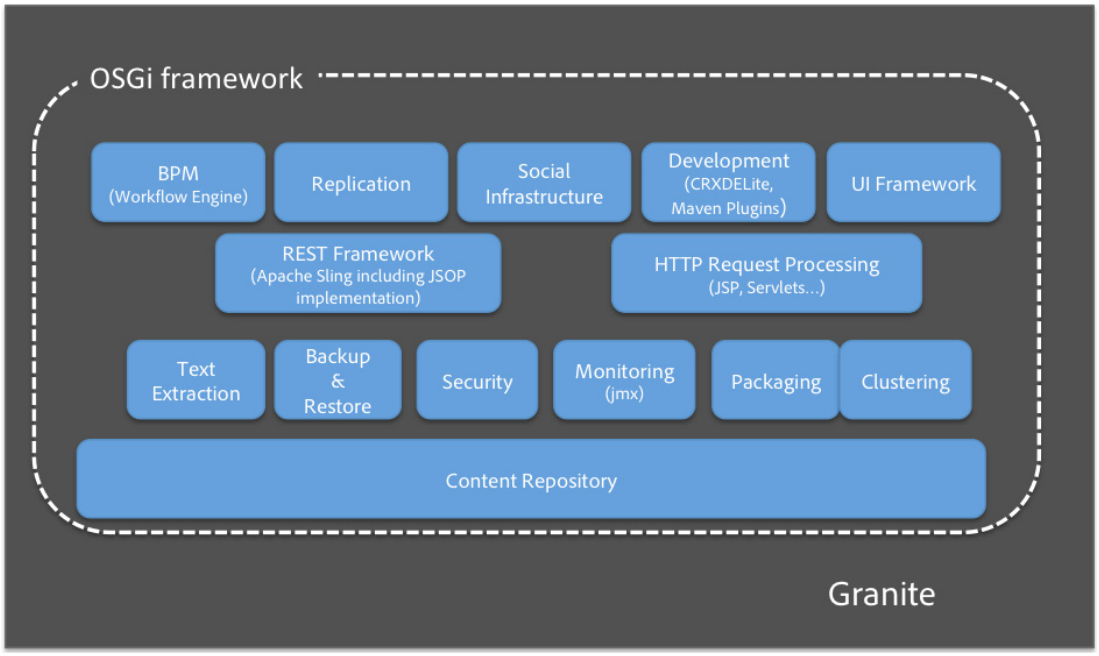
Third party repositories can be integrated with JCR connectors that expose their content into CRX and are available to AEM. Notice that the connectors are plugged in at the content repository level, which allows the content in the external repositories to appear to authors as if the content existed in 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, multi-site manager, taxonomy management, and workflow.

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

Since "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.

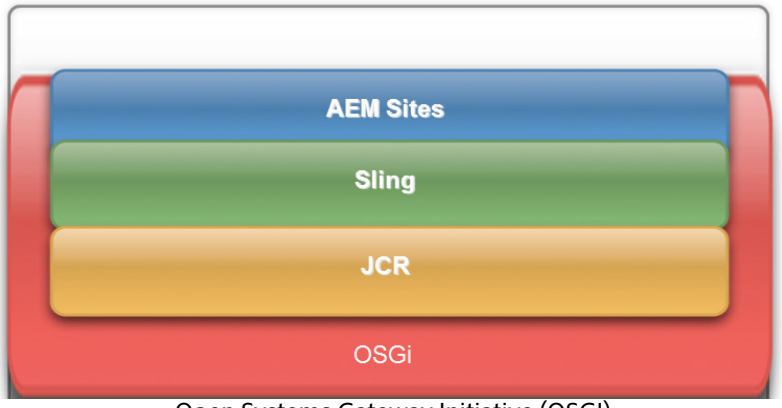



AEM is implemented as a Java web application. Since the system can be accessed from any computer with a modern web browser, no installation of client software is required; so, even large installations with thousands of users can be rolled out and upgraded easily.

The Application Runtime is OSGi, specifically Apache Felix. At this layer, you can hot deploy any code that you wrap up as an OSGi bundle. The OSGi runtime hosts Java applications that can access the repository via the JCR API.

As part of the Application Runtime, you get Apache Sling, a RESTful web application framework that exposes the full repository content via HTTP and other protocols.

# Architecture stack



Open Systems Gateway Initiative (OSGI)

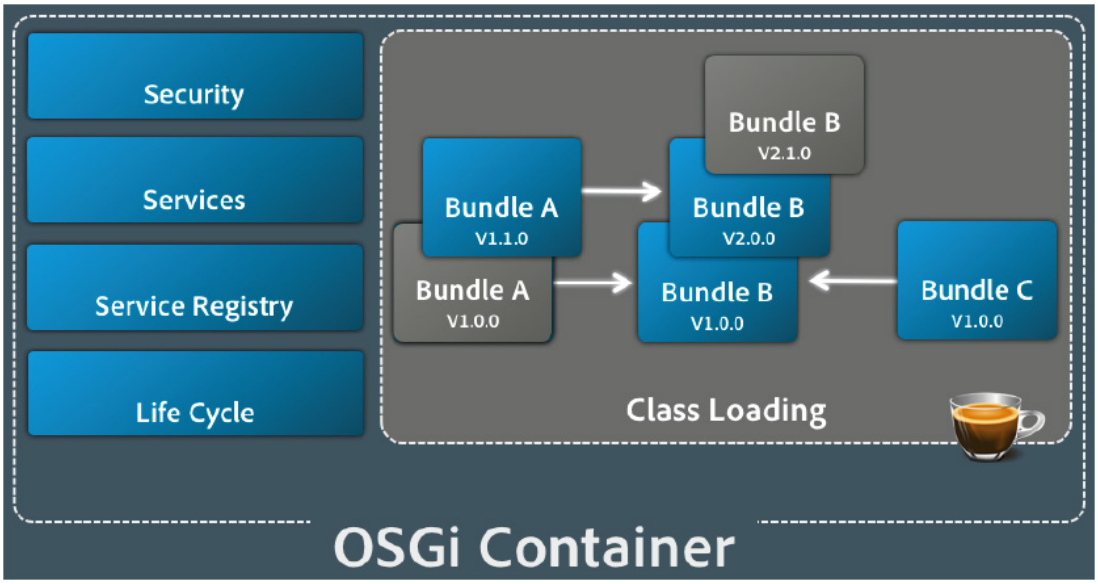
OSGi is a consortium that has developed a specification to build modular and extensible applications. The OSGi module system allows building applications as a set of reloadable and strongly encapsulated services.

Traditional Java application servers use a monolithic approach to application deployment. OSGi bundles run inside an OSGi container. The OSGi container manages relations among bundles, which are simply 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
  - Development model where applications are (dynamically) composed of many different (reusable) components
  - Components hide their implementations from other components while communicating through services, which are objects specifically shared between components

- Collaborative software environment
  - Application 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. 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 operations of the application—in other words, without taking the server down.

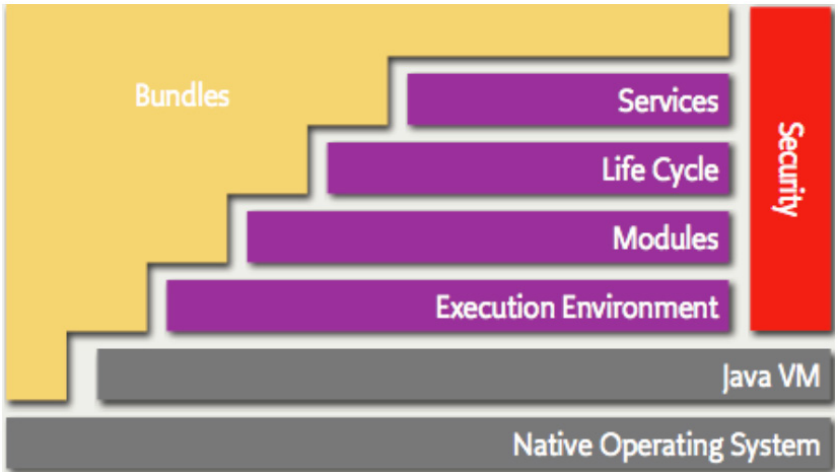
The OSGi container (Apache Felix) that is included in CRX is compliant to Version 4.2 of the OSGi specification.

Using an OSGi solution has the following benefits:

- Code is easier to write and test
- Reuse is increased; 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, Lifecycle, and Services—that define how extensible applications are built and deployed.




The responsibilities of the layers are:

- **Module**—Defines how a module, or a Bundle in OSGi-speak is defined. A bundle is just a plain old JAR file, whose manifest file has some defined entries. These entries identify the bundle with a symbolic name, a version, etc. In addition, there are headers, which define what a bundle provides—Export-Package; and what a bundle requires to be operative—Import-Package, and Require-Bundle.
- **Lifecycle**—The lifecycle 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 lifecycle 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.

Key principles of OSGi:

- 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

# 03

# Content Repository

This chapter provides you an overview of Content Repository in AEM. You will learn the following:

- Java Content Repository (JCR)
- Apache Jackrabbit
- Adobe CRX
- Repository structure
- Best Practices

# Java Content Repository (JCR)

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

A Java Content Repository 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 both 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) character, 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. Since 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 on 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 may have one primary node type that defines the characteristics of the node. They may also be associated with any 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 via a special reference type property. In this way, nodes in a JCR offer both referential integrity and 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:

- Common programmatic interface to content repositories
- API not tied directly to underlying architecture, data source, or protocol
- Content organization in 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.

# Apache Jackrabbit Oak

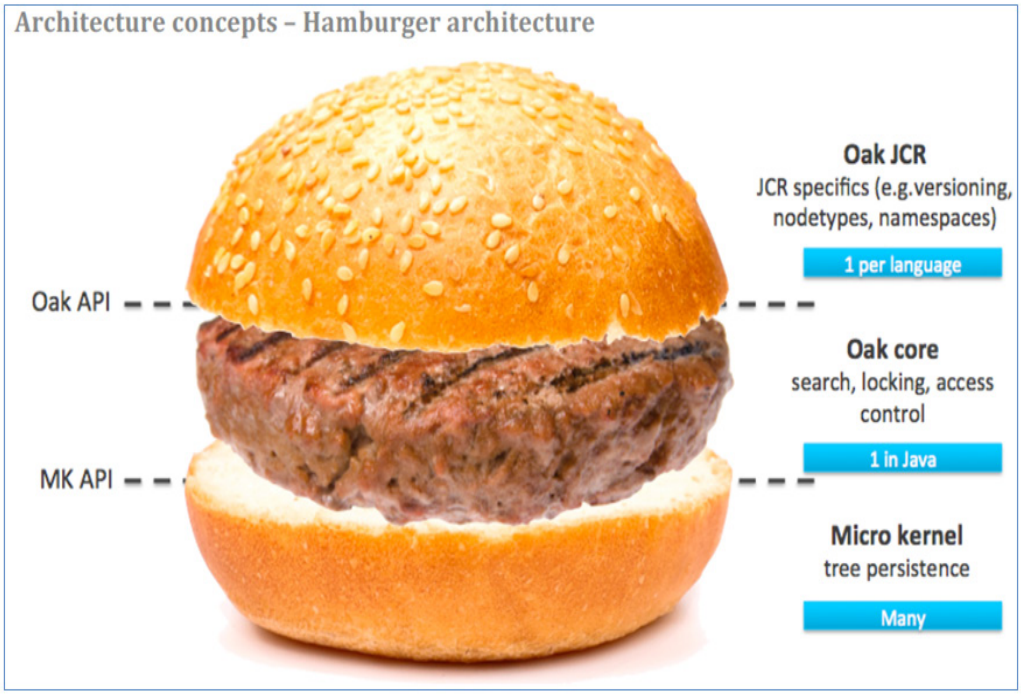
Jackrabbit Oak is an effort to implement a scalable and efficient hierarchical content repository for use as the foundation of modern, efficient web sites and other demanding content applications.

Jackrabbit Oak implements the Java Content Repository (JCR) spec. The most used parts of JSR-283 are implemented, but Jackrabbit Oak does not aim to be a reference implementation of JSR-283. AEM 6.0 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, and 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 MK's 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, e.g. node types
- It 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 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 and SegmentMK.

The MicroKernel Design Goals and Principles:

- Manage huge trees of nodes and properties efficiently
- MVCC-based concurrency control (writers do not interfere with readers; snapshot isolation)
- GIT/SVN-inspired DAG-based versioning model
- Highly scalable concurrent read & write operations
- Session-less API (there is no concept of sessions; an implementation does not need to track/manage session state)
- Easy to remote
- Efficient support for large number of child nodes
- Integrated API for efficiently storing or retrieving large binaries
- Human-readable data serialization (JSON)

## The MicroKernel Data Model

- Simple JSON-inspired data model: just nodes and properties
- A node consists of an unordered set of name to item mappings. Each property and child node is uniquely named and a single name can only refer to a property or a child node, not both at the same time.
- Properties are represented as name/value pairs
- Supported property types: string, number, Boolean, array
- A property value is stored and used as an opaque, unparsed character sequence

## 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. Currently, there are two relevant implementations of a DocumentMK:

- MongoMK: Based on MongoDB, this is currently the only DocumentMK implementation supported by AEM6
- RDBMK: Based on a relational database, currently not supported for AEM6

The discussion in the following sections is based on the MongoMK implementation because it is currently more mature than the RDBMK. While the concepts should apply for all implementations of DocumentMK, it is possible that some points are implemented differently in RDBMK.

## 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.

DocumentMK documents will be treated in detail in section "Storage of Oak Documents in MongoDB."

## Revisions

After each "commit" operation on the MicroKernel, a new revision is created.
In MongoMK, a revision is a string that consists of three pairs:

- A time stamp derived from the system time of the machine it was generated on
- A counter to distinguish revisions created with the same time stamp
- The cluster node id where the revision was created

## Branches

MicroKernel implementation supports branches, which allows client to stage multiple commits and make them visible with a single merge call.

## Previous Documents

DocumentMK adds data to a document with every modification but it never deletes any data (unless a cleanup is explicitly triggered). Old data is moved when there are 1000 commits to be moved or the document is bigger than 1 MB. Previous documents only contain immutable data, which means they only contain committed and merged revisions.

## Background operations

Each DocumentMK instance connecting to same database in Mongo server performs certain background task.

- Renew Cluster Id Lease
- Background Document Split
- Background Writes
- Background Reads

## Cluster node metadata

Cluster node metadata is stored in the clusterNodes collection. There is one entry for each cluster node that is running, and there are entries for cluster nodes that were ran. Old entries are kept so if a cluster node is started again, it gets the same cluster node id as before.

## 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 because of that it is easy to cache frequently accessed segments. This feature simplifies backups.
- Compactness—the size optimized is in a way that it reduces the IO costs and it fits 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 related nodes per session.

## Implementation

The supported (in AEM6) persistence mechanism of the SegmentMK is currently the TarMK, which is based on tar files in the local files system.

## Segments

One segment usually contains a continuous subset of a content tree; for example, node with its properties and closest child nodes. One 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 types of records and may refer to content in other segments. Bulk segments are only used for storing large binary values and they can contain only raw binary data, interpreted as a sequence of block records.

### Oak vs CRX

AEM 6.0 comes with the new JCR repository implementation based on Oak. This is enabled by default. Though there are differences in the way the new repository works internally, the applications running on AEM work in the similar manner. This is applicable to Sling and AEM as well.

### 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 and SegmentMK.

# 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

Add, consume, manage content with these interfaces:

- Java Content Repository API—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 plug-in
- Native repository interface via 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.

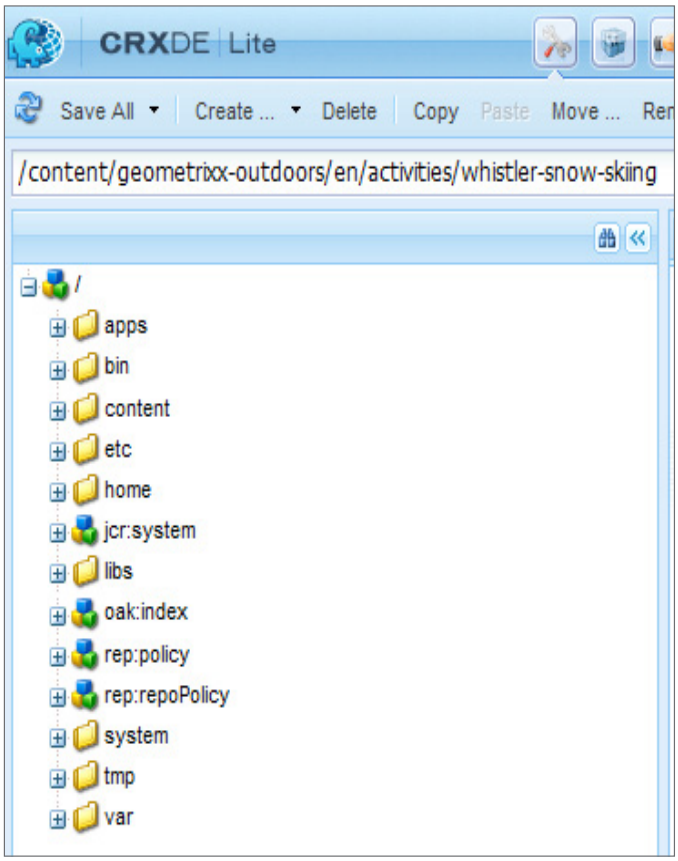
If you take a close look at the repository structure, you will notice that the repository has several major areas:

- **/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 as 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 web site. The components that you develop may be based on out-of-the-box components available at /libs/foundation/components.
- **/content—**Content created for your web site.
- **/tmp**—Temporary working area.
- **/home**—User and Group information.

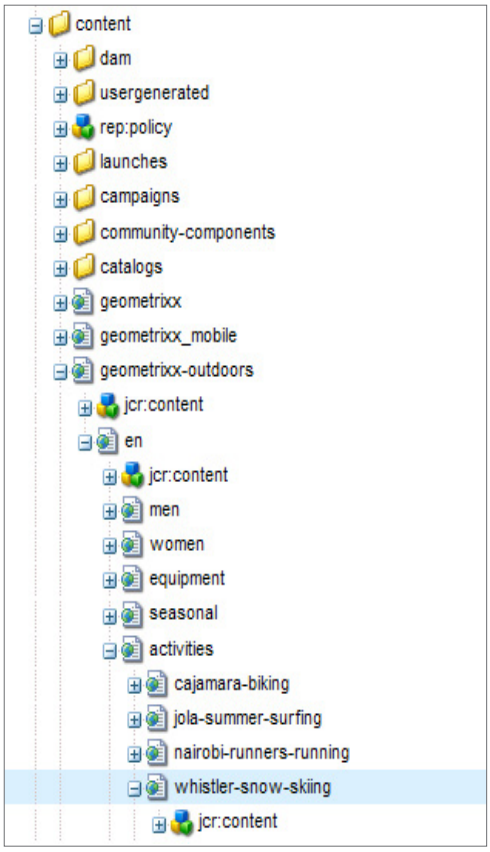
### EXERCISE 3.1 - Familiarize yourself with repository structure

1. Log in to CRXDE Lite.
   *http://localhost:4502/crx/de*

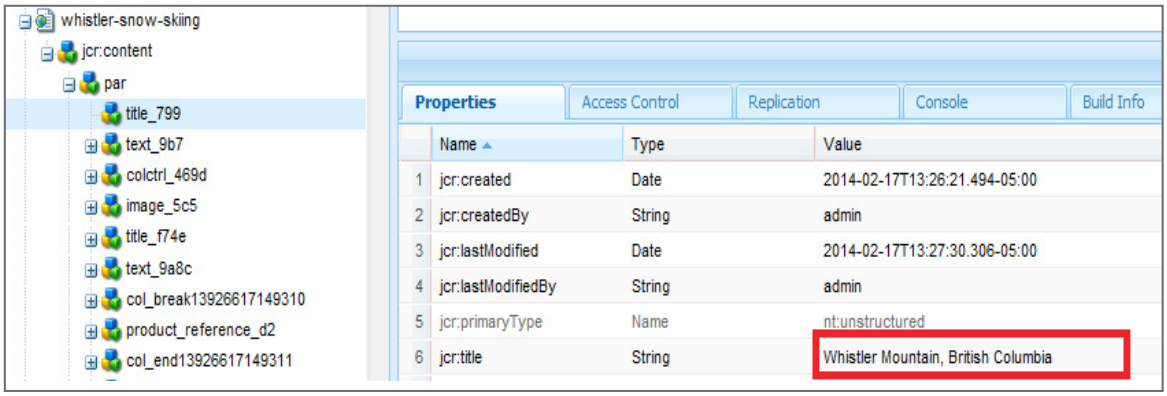
2. Observe the left pane to see the content structure that was mentioned earlier.



3. Navigate to **content** > **geometrixx-outdoors** > **en** > **whistler-snow-skiing**.

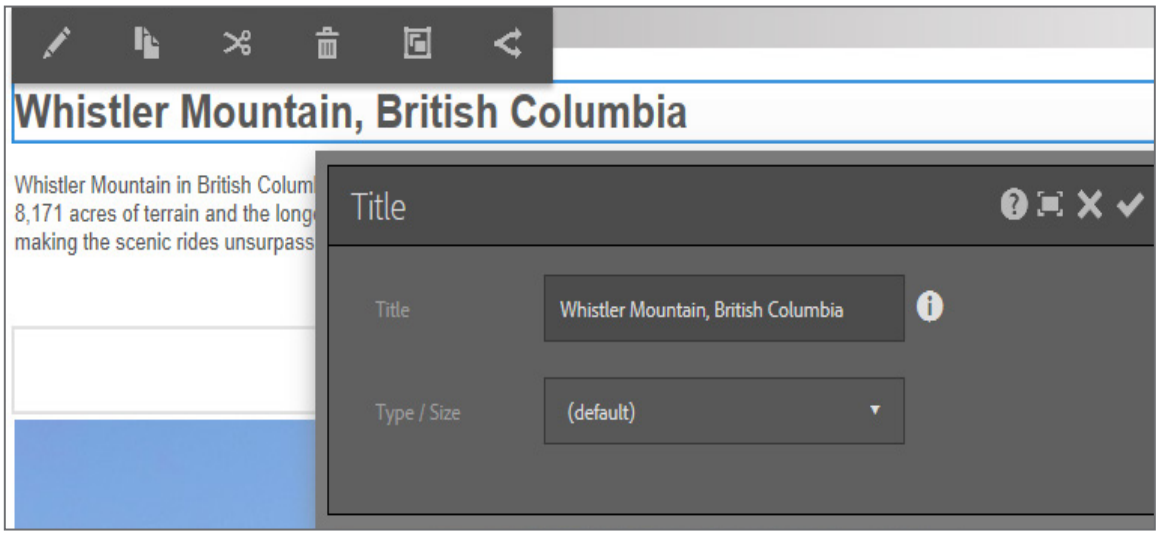



4. Click the *jcr:content* node.
   Note that all the content in the page appears as various nodes. Now, select the title node.
   Observe that it displays the title of your page.

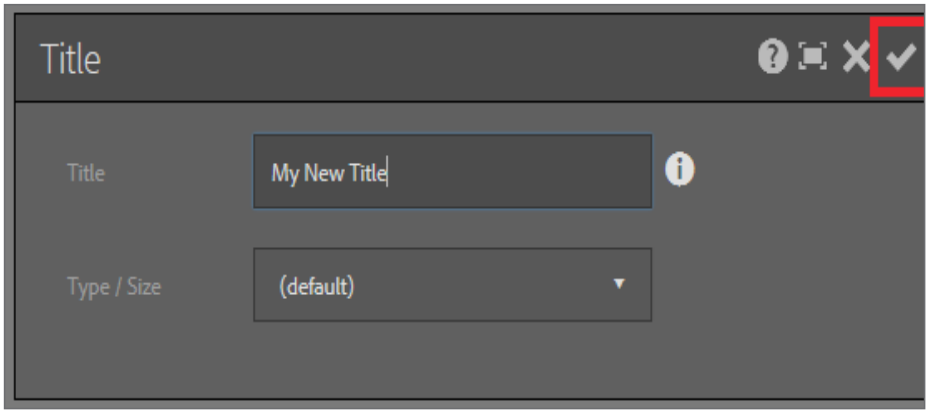



5. Open the page in a UI of your choice (Classic or Touch-Based UI).

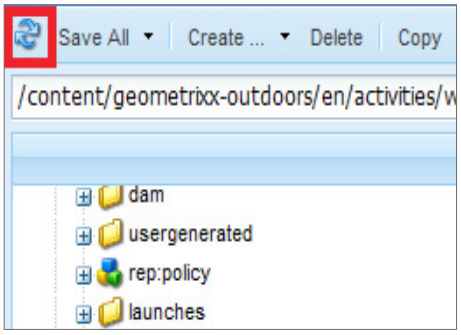
6. Double-click the Title component to open the **Edit** dialog box.



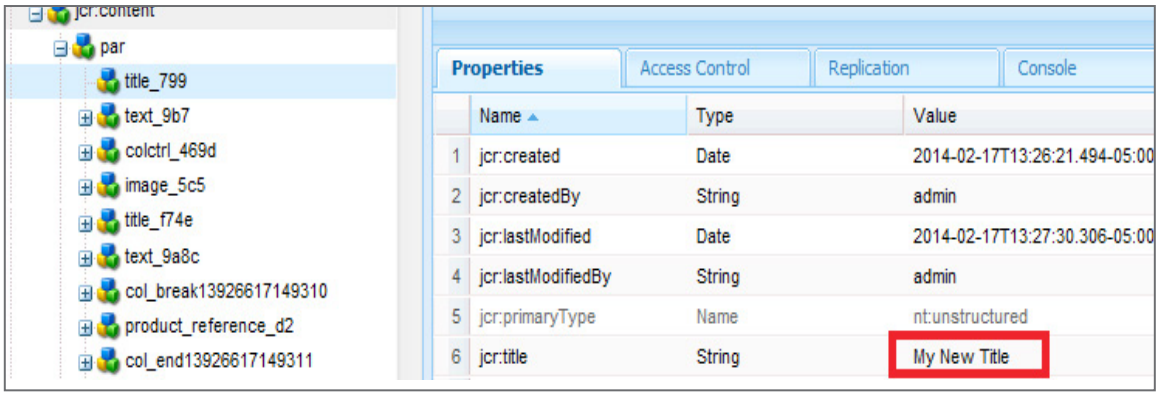

7. Update the title to "My New Title" and select **Done**.




8. Go to CRXDE Lite and select the Title node. Click the Refresh button displayed at the top-left corner.




Note that the title property is changed in CRXDE Lite.




# Best Practices

While working with CRX, keep the following best practices in mind:

- **Everything is content**: This is the idea behind AEM. Templates are content, user data is content, user ACLs are content and—of course—content is content. In addition, the compiled JSPs are also content. Moreover, where does content go? Content resides in the content repository. There are no loose files somewhere else to manage. Source code, dynamic modules, configuration, and even the state of an application reside side by side with documents and other digital assets such as images, audio and video, etc.
- **Data first, Structure later (maybe)**: Do not worry about a declared data structure in an ERD sense. Initially, Structure is expensive and in many cases, it is unnecessary to explicitly declare structure to the underlying storage.
- **Drive the content hierarchy—don't let it happen**: The content hierarchy is a very valuable asset. So, do not just let it happen—design it. If you do not have a good, human-readable name for a node, that is something you should probably reconsider. Arbitrary numbers are hardly ever a good name. While it may be extremely easy to put an existing relational model into a hierarchical model, one should put some thought in that process.
- **Names should have meaning**: IDs are evil—use meaningful names. In relational databases, IDs are a necessary means to express relations; so, people tend to use them in content models as well—mostly for the wrong reason, though. If your content model is full of properties that end in ID you probably are not leveraging the hierarchy properly. It is true that some nodes need a stable identification throughout their live cycle. Much fewer than you might think though.
- **References can be harmful if they create the need for structure**: References imply referential integrity. It is important to understand that references do not just add additional cost for the repository managing the referential integrity, but they also are costly from a content flexibility perspective.

# 04

# Web Framework

In this chapter, you will learn the following:

- Representational State Transfer (REST)
- Apache Sling

## Representational State Transfer (REST)

Addressable resources present a uniform interface that allows transfers of state (e.g. reading and updating of the resource's state). The best example of a REST-ful architecture is the web, where resources have URIs, and the uniform interface is HTTP.

For most cases, a framework like HTTP (addressable resources + "verbs" + standard ways to transmit metadata) is all you need to build a distributed application. HTTP is a rich application protocol, which gives you capabilities like content negotiation and distributed caching. REST-ful 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 Java Content Repository, such as Apache Jackrabbit or Adobe CRX to store and manage content. Sling applications use either scripts or Java servlets, selected based on simple naming conventions, to process HTTP requests in a REST-ful 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 softwares. When designing an API to be used over the Web, it therefore makes sense to adhere to these best practices. Apache 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 Java Content Repositories, 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. Apache 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 Apache 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, etc. is available from the ResourceResolver just like normal content—providing data to be rendered upon requests.

### EXERCISE 4.1 - Accessing Data in different formats

1. Access the following page:
   *http://localhost:4502/content/geometrixx*
   The page appears with a message that it redirects to the English page.

2. Change the URL as follows and access the page:
   *http://localhost:4502/content/geometrixx.xml*
   Note that you are getting an xml representation of the page.


```
▼<jcr:content xmlns:illustrator="http://ns.adobe.com/illustrator/1.0/" xmlns:social="http://www.adobe.com/social/1.0" xmlns:
 xmlns:granite="http://www.adobe.com/jcr/granite/1.0" xmlns:extensis="http://ns.extensis.com/extensis/1.0/" xmlns:xmpPLUS="h
 xmlns:mix="http://www.jcp.org/jcr/mix/1.0" xmlns:nt="http://www.jcp.org/jcr/nt/1.0" xmlns:adobe_dam="http://www.adobe.com/a
 xmlns:idPriv="http://ns.adobe.com/xmp/InDesign/private" xmlns:xmpNote="http://ns.adobe.com/xmp/note/" xmlns:album="http://n
 xmlns:tiff="http://ns.adobe.com/tiff/1.0/" xmlns:scg="http://www.adobe.com/social/scg/1.0" xmlns:crxde="http://www.day.com/
 xmlns:stMfs="http://ns.adobe.com/xap/1.0/sType/ManifestItem#" xmlns:xmpGImg="http://ns.adobe.com/xap/1.0/g/img/" xmlns:xmpT
 xmlns:exif="http://ns.adobe.com/exif/1.0/" xmlns:Iptc4xmpCore="http://iptc.org/std/Iptc4xmpCore/1.0/xmlns/" xmlns:Iptc4xmpE
 xmlns:MP="http://ns.microsoft.com/photo/1.2/" xmlns:mediapro="http://ns.iview-multimedia.com/mediapro/1.0/" xmlns:crs="http
 xmlns:stEvt="http://ns.adobe.com/xap/1.0/sType/ResourceEvent#" xmlns:xmpBJ="http://ns.adobe.com/xap/1.0/bj/" xmlns:oauth="h
 xmlns:prl="http://prismstandard.org/namespaces/prl/2.1/" xmlns:vlt="http://www.day.com/jcr/vault/1.0" xmlns:s7sitecatalyst=
 xmlns:pdfx="http://ns.adobe.com/pdfx/1.3/" xmlns:dam="http://www.day.com/dam/1.0" xmlns:prism="http://prismstandard.org/nam
 xmlns:viewerpreset="http://www.day.com/viewerpreset/1.0/" xmlns:xmpMM="http://ns.adobe.com/xap/1.0/mm/" xmlns:xmpDM="http:/
 xmlns:s7userdata="http://www.day.com/s7userdata/1.0/" xmlns:pdf="http://ns.adobe.com/pdf/1.3/" xmlns:sling="http://sling.ap
 xmlns:exifEX="http://cipa.jp/exif/1.0/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:psAux="http://ns.adob
 xmlns:acdsee="http://ns.acdsee.com/iptc/1.0/" xmlns:MicrosoftPhoto="http://ns.microsoft.com/photo/1.0" xmlns:sv="http://www
 xmlns:oak="http://jackrabbit.apache.org/oak/ns/1.0" xmlns:rep="internal" xmlns:crx="http://www.day.com/crx/1.0" xmlns:DICOM
 xmlns:stRef="http://ns.adobe.com/xap/1.0/sType/ResourceRef#" xmlns:stFNT="http://ns.adobe.com/xap/1.0/sType/Font#" xmlns:do
 xmlns:xmp="http://ns.adobe.com/xap/1.0/" xmlns:xmpRights="http://ns.adobe.com/xap/1.0/rights/" xmlns:plus="http://ns.useplu
 xmlns:MicrosoftPhoto_1_="http://ns.microsoft.com/photo/1.0/" xmlns:prismusagerights="http://prismstandard.org/namespaces/pr
 xmlns:cc="http://creativecommons.org/ns#" xmlns:slingevent="http://sling.apache.org/jcr/event/1.0" xmlns:jcr="http://www.jc
 xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:photoshop="http://ns.adobe.com/photoshop/1.0/" xmlns:lr="http://ns.adobe.com
 xmlns:xmpG="http://ns.adobe.com/xap/1.0/g/" jcr:primaryType="nt:unstructured" cq:designPath="/etc/designs/geometrixx" cq:de
 cq:lastModified="2014-04-22T10:11:24.002+01:00" cq:lastModifiedBy="admin" cq:loginPage="/content/geometrixx/en/toolbar/acco
 cq:profilePage="/content/geometrixx/en/toolbar/profiles/view" cq:signupPage="/content/geometrixx/en/toolbar/account/registe
 redirectTarget="/content/geometrixx/en" sling:redirect="true" sling:redirectStatus="302" sling:resourceType="foundation/com
```


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 02 2014 14:47:35 GMT+0530","jcr:con
["mix:versionable"],"jcr:createdBy":"admin","jcr:title":"Geometrixx Outdoors Site","jcr:versionHistory":"ea4806
78bb0d8518b8","redirectTarget":"/content/geometrixx-outdoors/en","jcr:predecessors":["7b9d46f9-1da7-408b-aa87-2
14:47:35 GMT+0530","cq:lastModified":"Tue Apr 22 2014 14:41:24 GMT+0530","sling:redirectStatus":302,"cq:cloudse
outdoors-twitter-app","/etc/cloudservices/facebookconnect/geometrixx-outdoorsfacebookapp","/etc/cloudservices/m
["/geohome"],"jcr:description":"Geometrixx Outdoors - fashion that doesn't sacrifice style for comfort","jcr:ba
2d8422cb79eb","jcr:isCheckedOut":true,"cq:deviceIdentificationMode":"client-side","jcr:uuid":"66b18959-f3c4-45d
347c14d76bb4","sling:resourceType":"foundation/components/redirect","sling:vanityOrder":1000,"cq:allowedTemplat
outdoors/templates/.*","/libs/social/blog/templates/.*"],"cq:designPath":"/etc/designs/geometrixx-outdoors","cq
{"jcr:primaryType":"nt:unstructured","data-bmap-devgroups":"browser, oldBrowser, highResolutionDisplay","cq:var
outdoors","cq:childNodesMapTo":"hreflang"},"image":{"jcr:primaryType":"nt:unstructured","jcr:lastModifiedBy":"a
GMT+0530","imageRotate":"0"}},"en":{"jcr:primaryType":"cq:Page","jcr:createdBy":"admin","jcr:created":"Fri May
{"jcr:primaryType":"cq:PageContent","jcr:mixinTypes":["cq:LiveSync"],"jcr:createdBy":"admin","jcr:title":"Engli
outdoors/en/user/account","cq:commerceProvider":"geometrixx","cq:catalogBlueprint":"/content/catalogs/geometrix
outdoors/en/toolbar/offline","cq:template":"/apps/geometrixx-outdoors/templates/page_home","jcr:language":"en_U
GMT+0530","cq:checkoutPage":"/content/geometrixx-outdoors/en/user/checkout","jcr:created":"Fri May 02 2014 14:4
GMT+0530","cq:mailboxPage":"/content/geometrixx-outdoors/en/user/mailbox","cq:cartPage":"/content/geometrixx-ou
outdoors/en/user/register","sling:resourceType":"geometrixx-outdoors/components/page_home","cq:composeMessagePa
message","cq:designPath":"/etc/designs/geometrixx-outdoors","cq:socialProfilePage":"/content/geometrixx-outdoor
profile","cq:commerceType":"section","cq:lastRolledoutBy":"admin","cq:lastModifiedBy":"admin"},"men":{"jcr:prim
02 2014 14:47:38 GMT+0530"},"women":{"jcr:primaryType":"cq:Page","jcr:createdBy":"admin","jcr:created":"Fri May
```

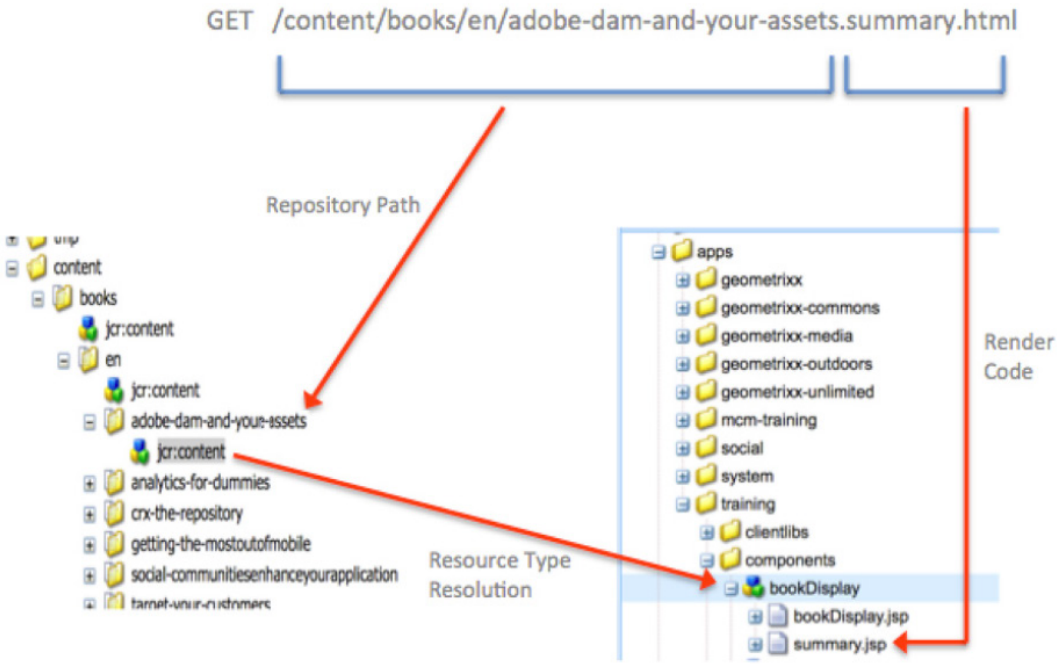

## Sling Resolution

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.

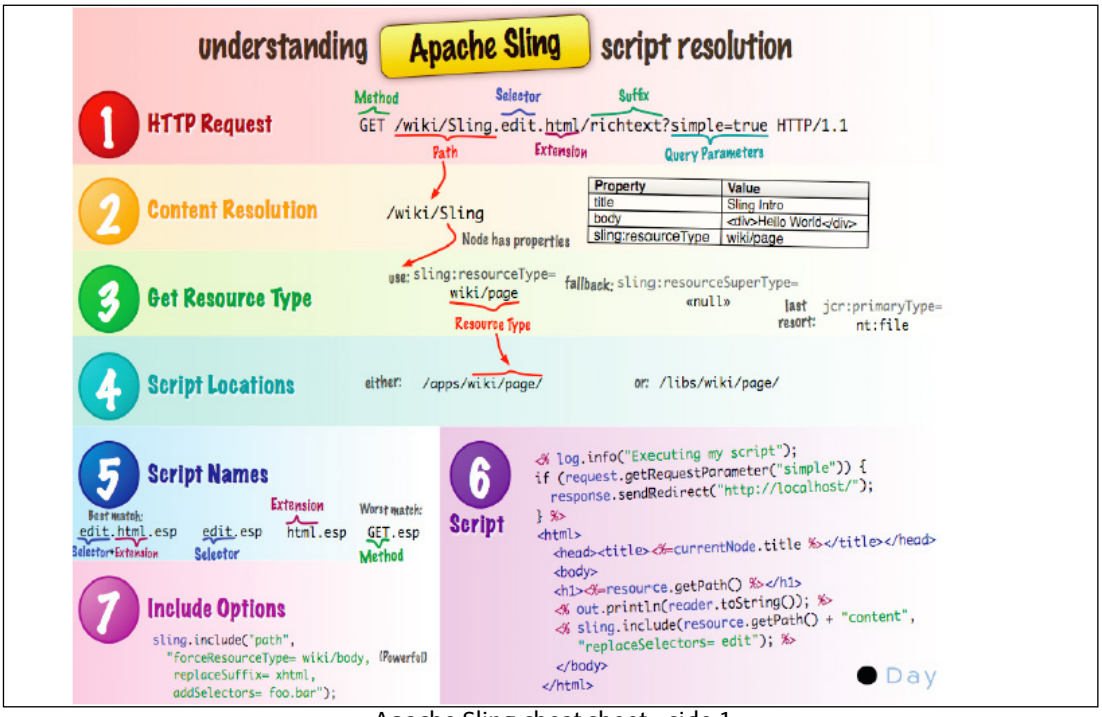
Apache Sling is included in the installation of AEM. Apache Sling was originally designed and implemented by Day Software (now Adobe Systems). Sling has since been contributed to the Apache Software Foundation—further information can be found at Apache (http://sling.apache.com).

When using Apache 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 with a wide range of different content elements. It is also useful when you need pages that can be easily customized or viewed differently.

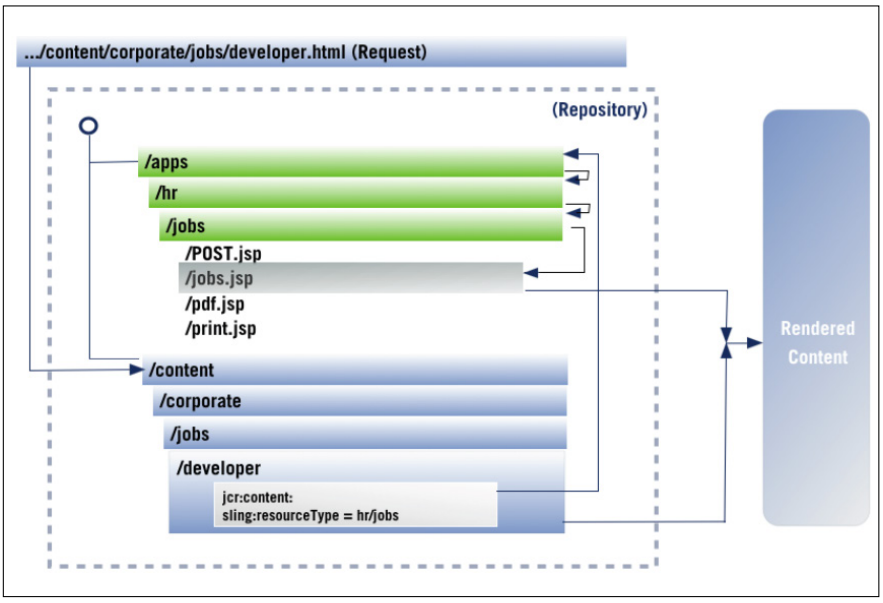



The following diagram explains the Sling script resolution. It shows how to get from HTTP request to content node; from content node to resource type; from resource type to script—and what scripting variables are available.




Apache Sling cheat sheet - side 1

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




Sling request decomposition

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)—as 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 Process four elements of a SlingHttpServletRequest:

1. 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 contained to forward slashes. Also, any backslashes contained are replaced to forward slashes. This should give a relative path. Of course a resource type may also be set to an absolute path.
2. The **request selectors** as retrieved through `request.getRequestPathInfo().getSelectorString()`.
3. The selector string is turned into a relative path by replacing all separating dots by forward slashes. For example the selector string print.a4 is converted into the relative path print/a4.
4. The **request extension** as retrieved through `request.getRequestPathInfo().getExtension()` if the request method is GET or HEAD and the request extension is not empty.
5. 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:

Not creating multiple folders matching selectors, instead use one single jsp, getting all the selectors on the request by using SlinghttpServletRequest.gerSectionRequestPathInfo.getSelectors() and then in that one jsp using a series of if..else

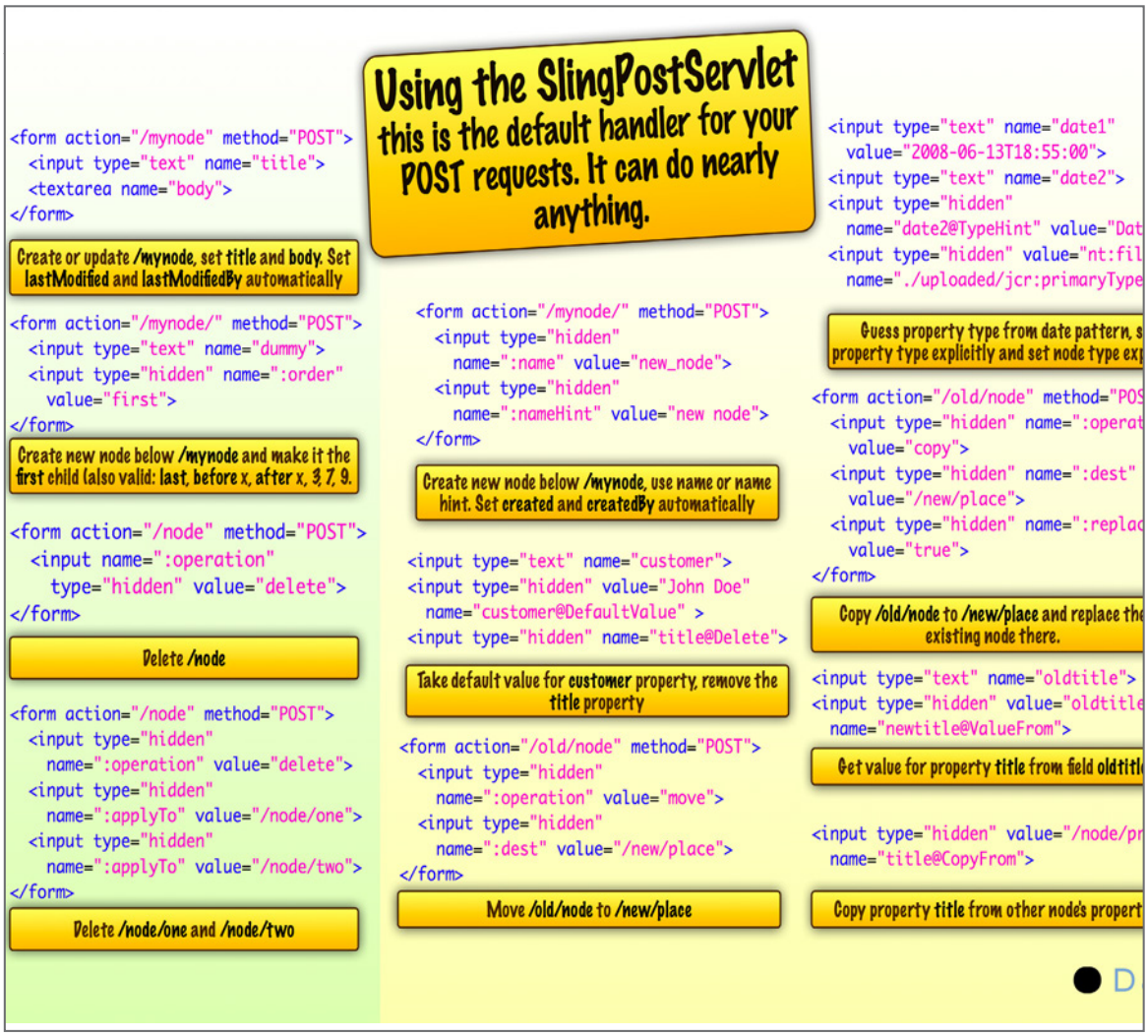
if (selector1.equals(“xxx”)) then….
else if (selector2.equals(“yyy”)) then…
else if (selector3.equals(“zzz”)) then…
etc.. etc.. Creating one single jsp to test all the different cases

| 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 | html | /c/d |

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.



# 05

# Scripting language - Sightly

Sightly is the new AEM templating system introduced with AEM 6.0. It replaces JSP (Java Server Pages) and ESP (ECMAScript Server Pages) as the preferred UI templating system for AEM. Note that AEM supports JSP and ESP as templating languages.
In this chapter, you will more about Sightly.

## 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 either 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 that 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.sly"/>
```

and 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 to 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:

```
1  <div class="sightly-example">
2      <header data-sly-include="header.html"></header>
3      <h1 data-sly-test="${properties.title}">
4          ${properties.title}
5      </h1>
6      <section data-sly-use-navigation="Navigation">
7          <h1>
8              ${navigation.breadcrumb}
9          </h1>
10     </section>
11     <ul data-sly-list-child="${resource.listChildren}">
12         <li>${child.name}</li>
13     </ul>
14 </div>
```

As you can see, the above snippet is valid HTML5 but it includes two special things:

- 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-through 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`.

## 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 simply 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 it's 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 include:

```
<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 etc.) when it is procesed 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:

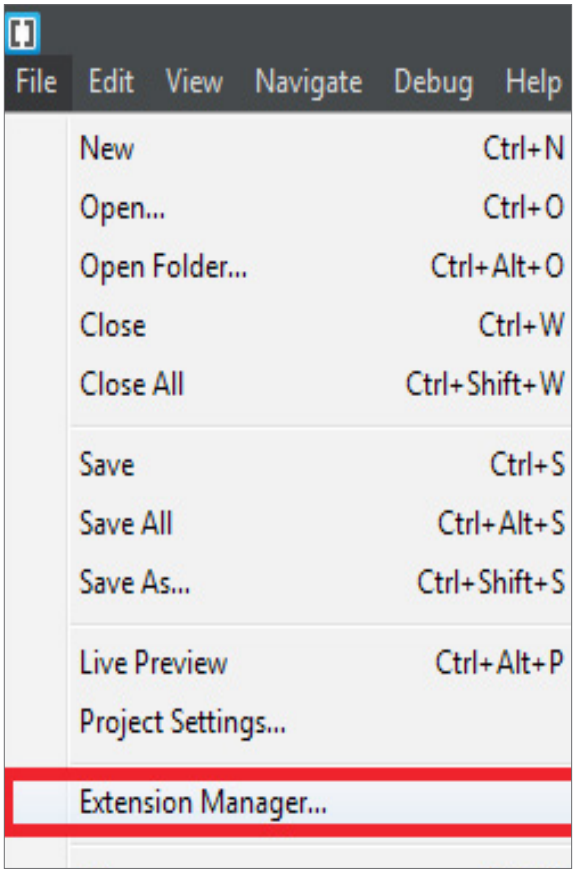
```
<section data-sly-include="path/to/template.html"></section>
```

# Sightly Tooling - AEM Sightly Brackets Extension

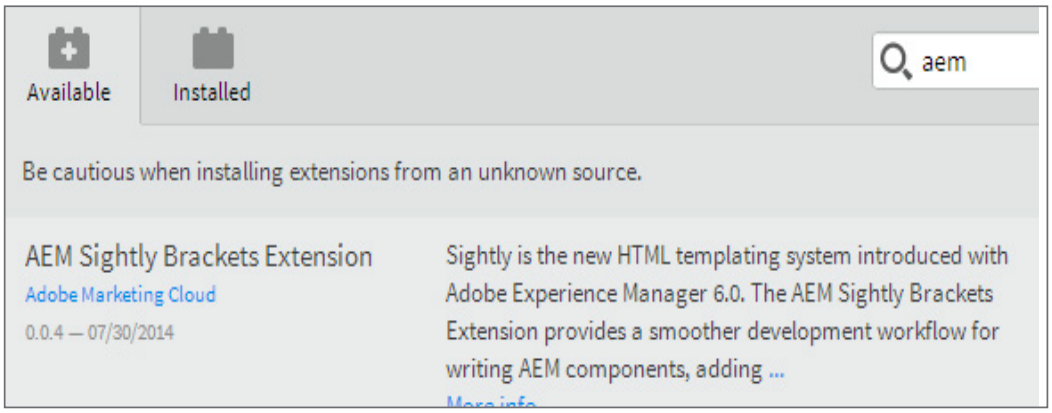
Adobe provides you with an extension for Brackets to simplify Sightly coding. Brackets is an open source code editor for web designers and front-end developers. For more details on Bracket, visit: *http://brackets.io/*

### EXERCISE 5.1 - Work with the Brackets plugin

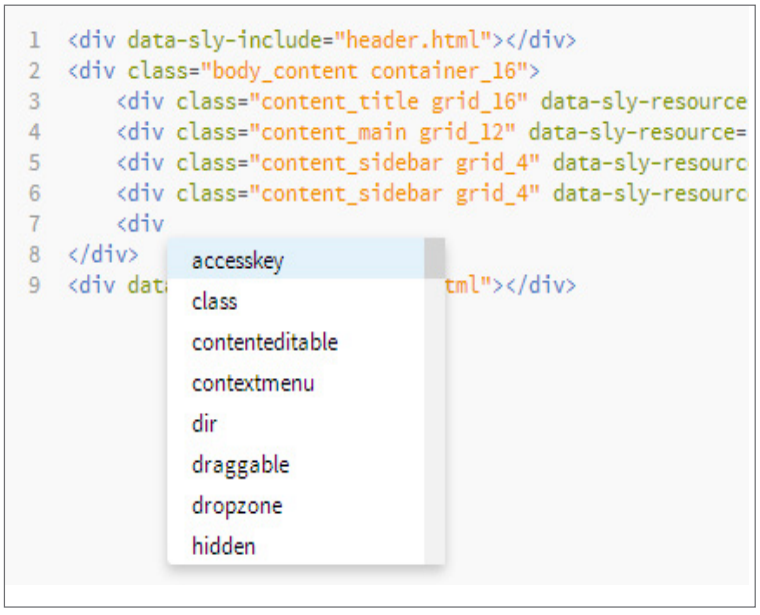
1. Download the latest version of Bracket from the site: *http://brackets.io/*
   (The installation files are provided in the USB.)
2. Install Bracket and open it.
3. Click **File** > **Extension Manager**.



4. In 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 Sightly Bracket Extension.




6. After a successful installation, the success message appears in the screen. Restart Bracket.
7. Copy the project from the USB to your local machine.
8. Select **File** > **Open Folder**. Navigate to the folder you copied and open the *jcr_root* folder.
9. Open *cr_root\apps\test\components\page-content\body.html*.
10. Try making some changes. Note that you are provided with code hinting.




You have installed AEM Sightly Brackets extension. You also have seen how to open a page and update the code using Brackets.

# 06

# AEM Authoring Framework - Templates

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

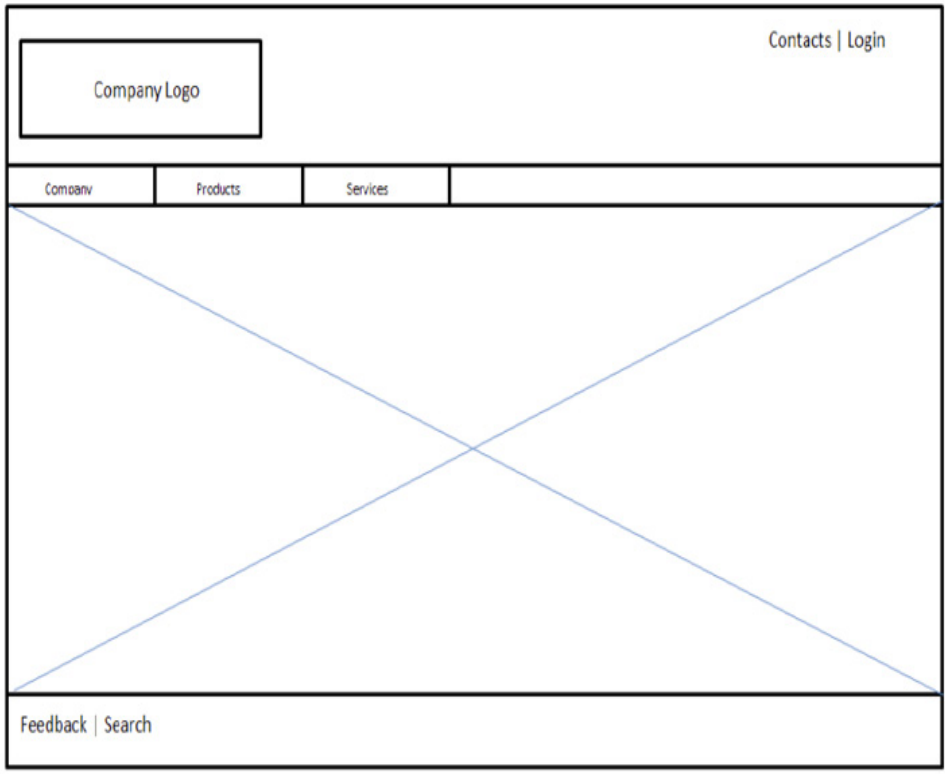
- Create an appropriate directory structure
- Create a template and a rendering component
- Display basic page information

## 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 don't have a website; you are developing the website from scratch. 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 logo: A company logo. The company logo doesn't change from page to page. You don't want the author to set the company logo.
- A contacts link: This should take the user to a page that has the contact information.
- A login link: This should take the users to a page where users 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.
- A Search link: This should take users to the search 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.
  Tip: Remember having as few templates as possible is good practice

- Sketch or white board 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

Once you have identified the templates, you can begin to deconstruct the pages, 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 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.

# 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:

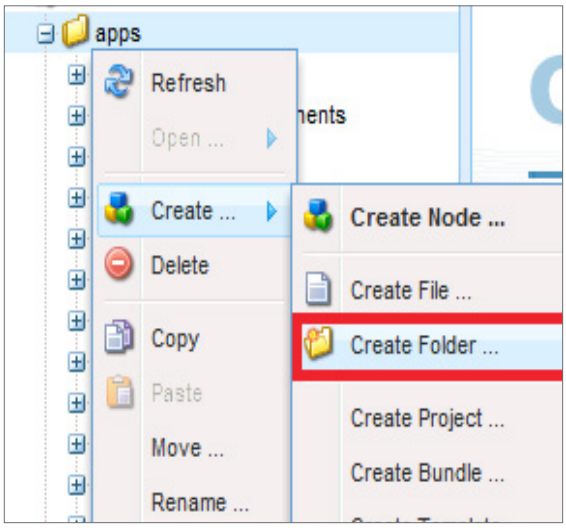
| Structure | Description |
|---|---|
| /apps/*application-name* | The website's main folder |
| /apps/*application-name*/components | The folder to hold components |
| /apps/*application-name*/templates | The folder to hold templates |

NOTE: Organize your website in ways that allow you to maintain them easily.

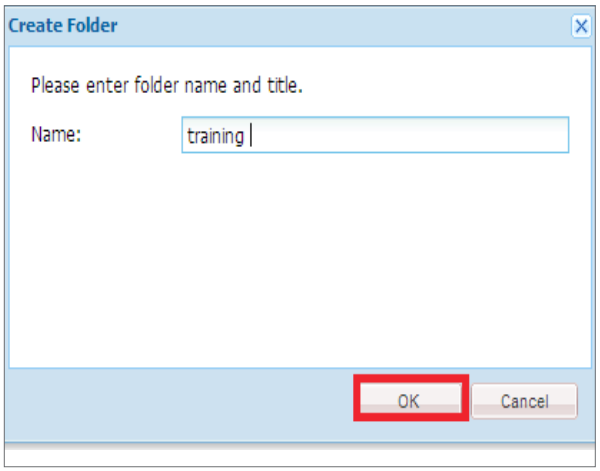
### EXERCISE 6.1 - Create the structure of your website

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

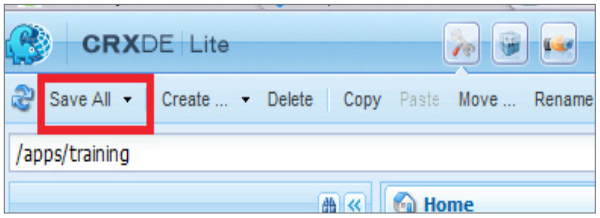
1. Log in to CRXDE Lite. In the left pane, navigate to the *apps* folder and right-click. Select **Create** > **Create Folder**.



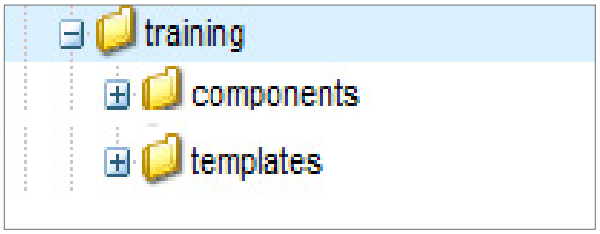
2. Enter the name as *training* and select **OK**.
   As a best practice, provide the node names in lowercase, because they become a part of the URL.




3. From the top bar, select **Save All**.




4. 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 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 (a*pps/<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.

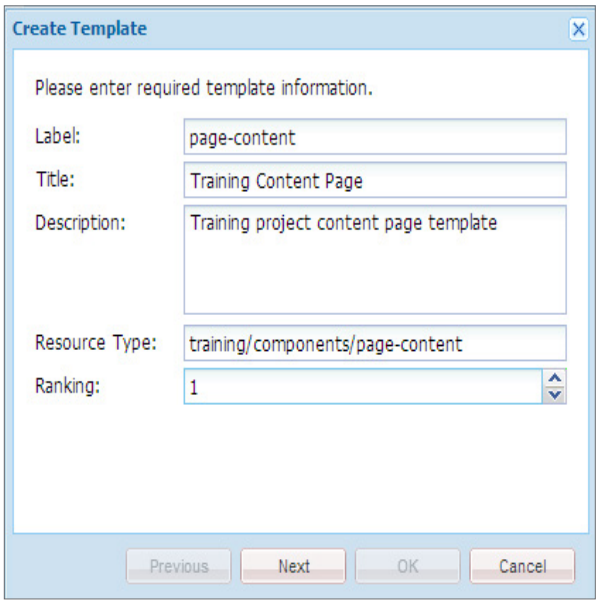
When you create a template, the following information is saved in the repository. The template creation widget provides you with options to enter these information:

- Label—cq:Template node name
- Title—jcr:title property
- Resource Type—sling:resourceType property
- Ranking—ranking property
- Allowed Paths—allowedPaths property

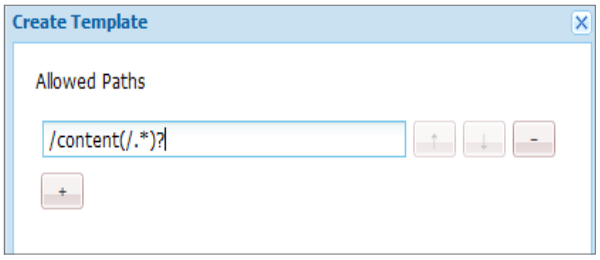
### EXERCISE 6.2 - Create a template for your website

In the following exercise, you will create a template.

1. Right-click the templates folder created in the previous exercise.
2. Select **Create** > **Create Template**.
3. Enter the following details and click **Next:**
   - 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 template appears in 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(/.*)?

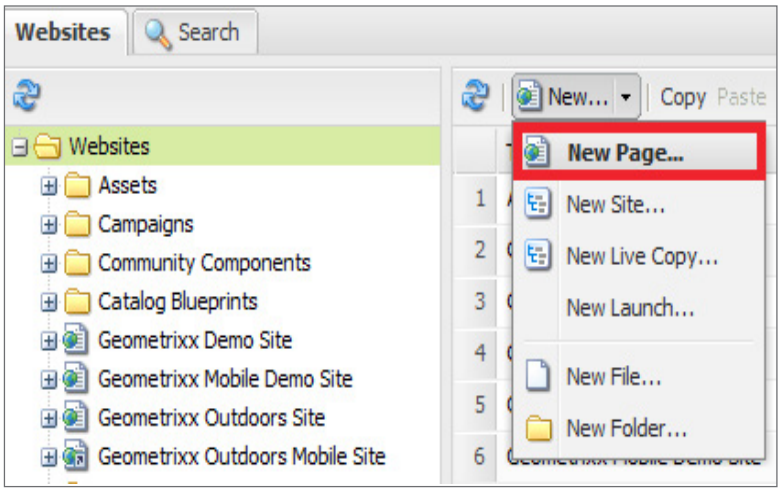



5. Click **Next** in the Allowed Parents screen.
6. Click **OK** in the Allowed Children screen.
7. Click **Save All** in the top bar.

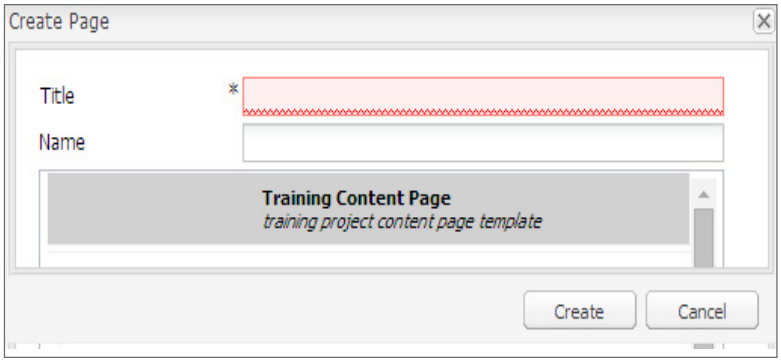
## Testing the template

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 SiteAdmin and select the root node, *websites*.
   *http://localhost:4502/siteadmin*
2. Select **New** > **New 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.



# Create a page-rendering component

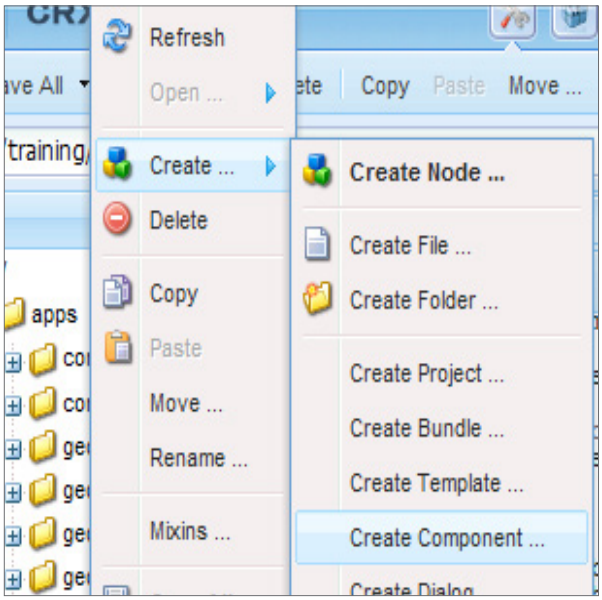
Components are modular, re-usable units that implement specific functionality or logic to render the content of your web site. They have no hidden configuration files, can include other components, and can run anywhere within the AEM or in isolation (e.g. portal). A component could be described as a collection of scripts (e.g. Sightly files, JSPs, Java servlets, etc.) 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 (e.g. *contentpage.html* or *contentpage.jsp*).

### EXERCISE 6.3 - Create a page-rendering component

1. Right-click the */apps/training/components* folder and select **Create** > **Create Component**.
2. Enter the following details:
   - 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. 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**.




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

5. Open the script by double-clicking.
   The script displays with some sample code that you can delete for the time being.

6. 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 page-rendering component.

# Create pages

A Page is many things:

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

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

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, 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

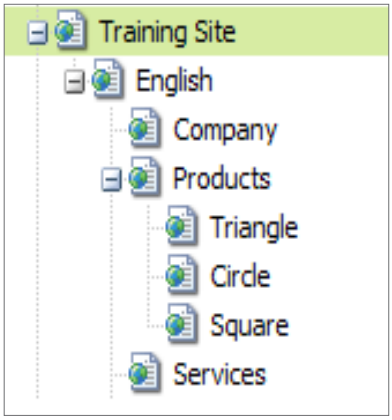
In this exercise, you will create the basic structure of your website.

1. Log in to AEM.
   *http://localhost:4502/siteadmin*
2. Click Websites in the left pane.
3. Select **New** > **Create Page**.
4. Select the template that you want to use and click **Next**.
5. Enter the title of the page as "Training Site" and click **Create**.
6. Double-click the page to open it.
   The page doesn't display in the Touch-Optimized UI.
7. Move to the Classic UI, by replacing the editor.html to cf# in the URL.
   http://localhost:4502/cf#/content/training-site
   The page appears as follows:

   Note that the page displays the texts added in the default script of the component.
8. Go to Site Admin again and select the page you created.
9. Click **New** > **New Page**. Enter the details as follows:
   Title: English
   Name: en
10. Create one more page with the following details:
    Title: French
    Name: fr
    This is for the French locale.
11. Select the English page create the following pages underneath:
    - Company
    - Products
    - Services
12. Select the Product page and create the following pages underneath:
    - Triangle
    - Circle
    - Square

13. Ensure that you create a page 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 Java Server Page (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 will learn more about Sightly in the next chapter. 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 web site. The course will also provide you with additional information that you need if you are developing rendering scripts using JSP.

### EXERCISE 6.5 - Modify page-rendering script to use 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 select **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. Save the file you created.
8. In the AEM Admin tool, open the newly created Training Site page by double-clicking it.
9. 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*


10. Note that the page has been 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. In order 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:

- Via 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();
    ```

- Via 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");
    ```

- Via currentNode object
  - The currentNode object is an instance of the Node (see JCR API) class, which provides access to content via the getProperty() method.
  - Sightly

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

  - JSP

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

**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.

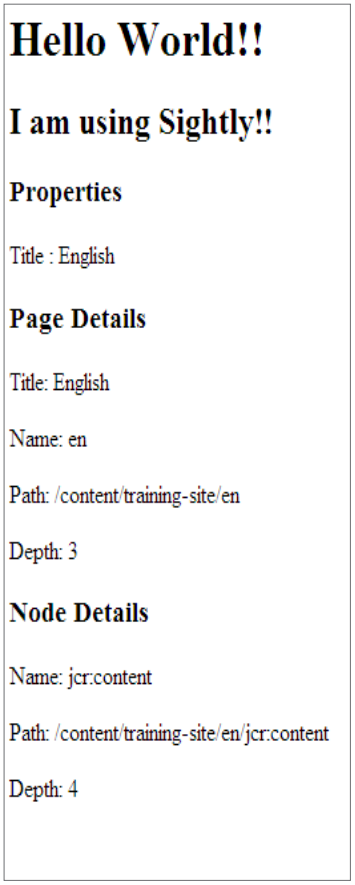
### EXERCISE 6.6 - Display basic page content using available APIs (in Sightly)

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

1. Open the *page-content.html* script.
2. Enter the following code, and then **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>
```

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




## Displaying the 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.

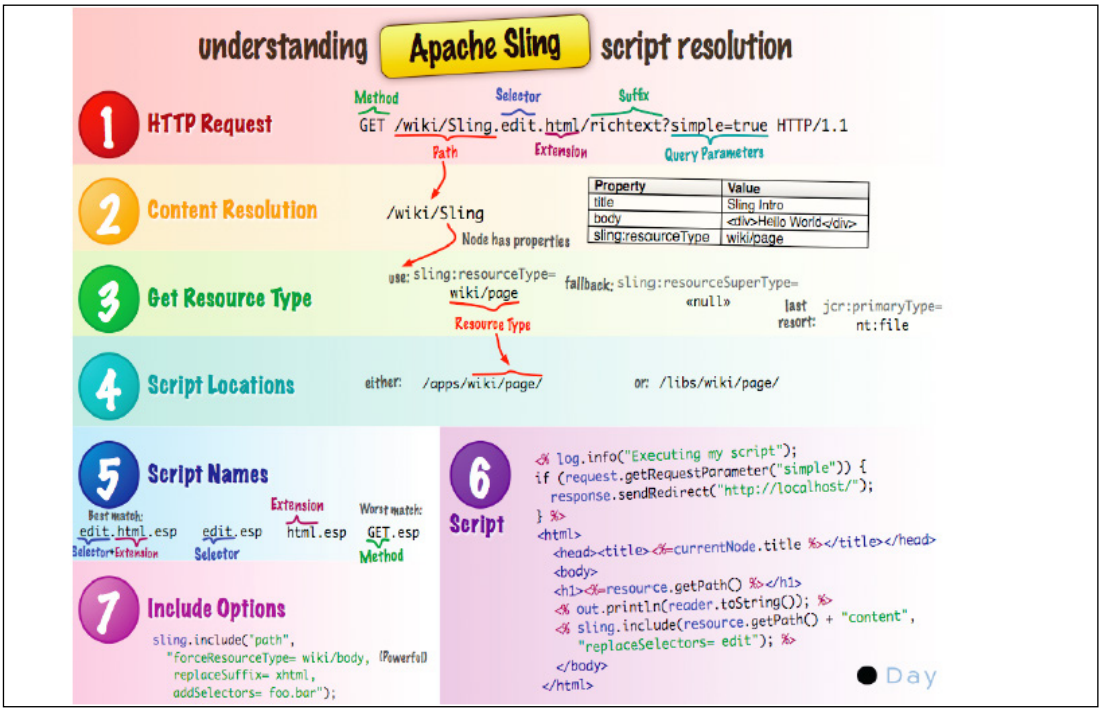
- ComponentContext
  - The current component context object of the request (*com.day.cq.wcm.api.components.ComponentContext* interface).
- Component
  - The current AEM component object of the current resource (*com.day.cq.wcm.api.components.Component* interface).
- CurrentDesign
  - The current design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

Use the following code to test the same using a JSP script. Create *page-content.jsp* and then copy and paste the following code. Refresh the page you created.

```
<%@include file="/libs/foundation/global.jsp"%>
<html>
    <head>
        <title>Hello World</title>
    </head>
<body>
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/>
    </body>
</html>
```

# Recap of Sling framework

Now, you have created a template and web pages based on the template. Let us take a quick recap of 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, locate the *page-content* node.
2. (If you have created a **JSP** file) Ensure that you deleted the *page-content.jsp* script you created by right-clicking and selecting **Delete**.
3. Right-click, select **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 won't use these scripts to build the sample website.

# 07

# AEM Authoring Framework - Components & Design

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

- Modularize the template
- Extend 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 the compilation time.

```
<%@ include file="myScript.jsp" %>
```

Adobe recommends you to include a file at run time 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 the runtime. However, Adobe recommends you to 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 have seen 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:* to hold the content that changes from page to page.
- *header.html*: to hold the content that should appear in the header. It does not change from page to page.
- *footer.html*: to hold 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. Using CRXDE Lite and locate the page component.
2. Open *page-content.html* and the following code:
   Note that you use the *data-sly-include* tag to include an additional file to your template.

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

3. Right-click the component and select **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>
```

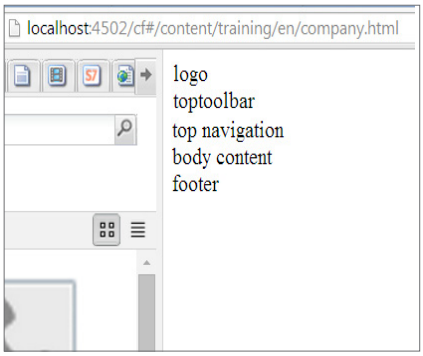
5. Create a new file—*header.html*, and add the following code:

```
<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>
```

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. Refresh the web page that you created:
   *http://localhost:4502/cf#/content/trainingsite/en.html*
   The page appears as shown below:




**Example: Including additional files in JSP**

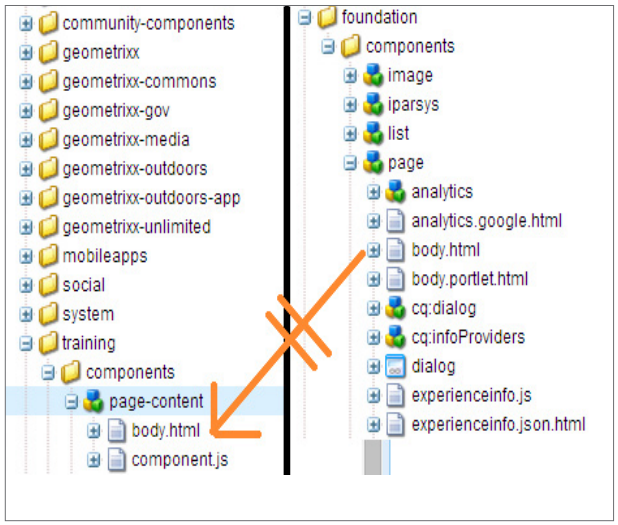
```
<%@include file="/libs/foundation/global.jsp"%>
<html>
   <head>
      <title><%= currentPage.getTitle() == null ? currentPage.getName():currentPage.
getTitle() %></title>
      <cq:include script="header.jsp"/>
   </head>
</html>
```

# Inherit the component hierarchy

Components can be given a hierarchical structure to implement the inheritance of included script files, Dialogs, etc. 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 (e.g. the <head> section).
Components within AEM are subject to 3 different hierarchies:

- 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:
  - scripts (resolved by Sling)
  - Dialogs
  - Descriptions (including thumbnail images, icons, etc.)

  - It is important to note that a local copy/instance of a Component element (e.g. *body.html*) will take precedence over an inherited element.




- 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 (editbars, rollover, etc.), Dialog layout (inline, floating, etc.) 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.

- Include Hierarchy
  - This is imposed at runtime by the sequence of includes.
  - This hierarchy is typically used by the Designer, which in turn acts as the base for various design aspects of the rendering—including layout.
  - Information, CSS information, the available components in a paragraph system, etc.

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:

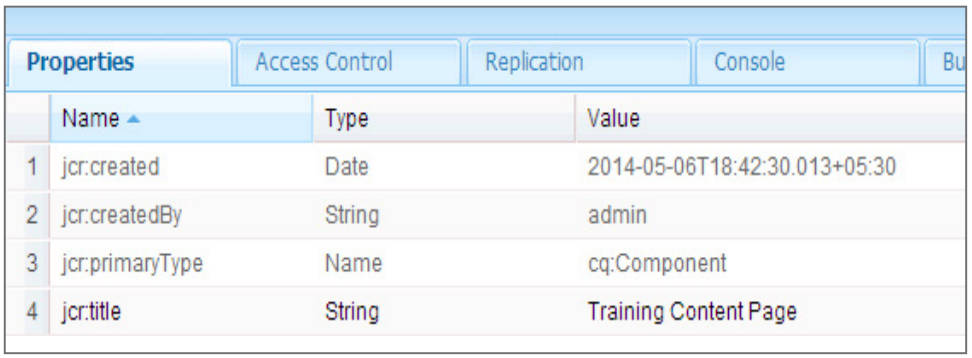
- *libs/foundation/components*: All components developed using JSP are available here.
- *libs/wcm/foundation/components*: All components developed using Sightly are available here.

  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.

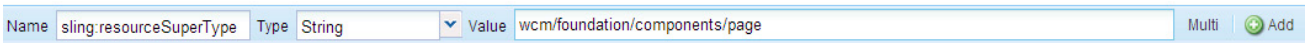
### EXERCISE 7.2 - Inherit the Sightly foundation component page

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

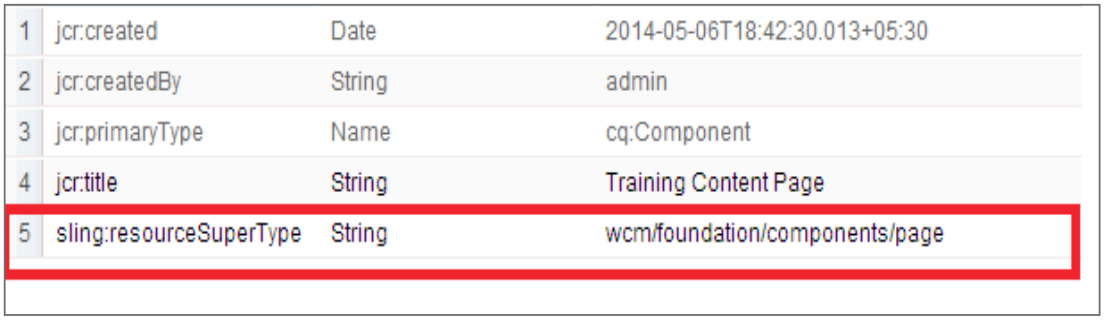
1. In CRXDE Lite, select **Training** > **Components** > **page-content.**
   The properties of the component appears as shown below:




2. Enter the property "Name" (***sling:resourceSuperType***), "Type" (***String***), and "Value" (**wcm/*foundation/components/page***) in the dialog—then click **Add**.



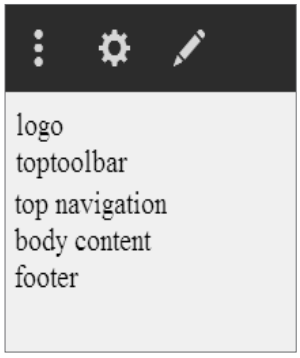
3. Observe that the property you added appears in the bottom bar.



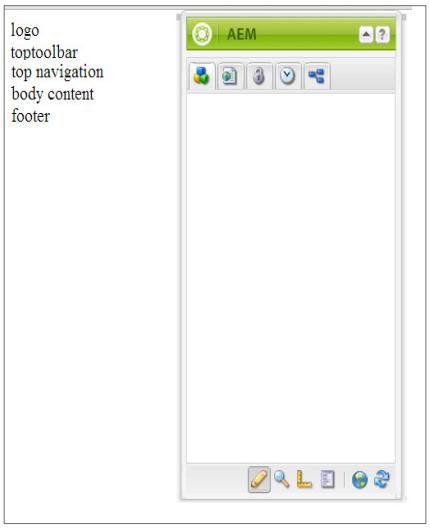
| | | | |
|---|---|---|---|
| 1 | jcr:created | Date | 2014-05-06T18:42:30.013+05:30 |
| 2 | jcr:createdBy | String | admin |
| 3 | jcr:primaryType | Name | cq:Component |
| 4 | jcr:title | String | Training Content Page |
| 5 | sling:resourceSuperType | String | wcm/foundation/components/page |

4. 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*. The *body.html* in your component overrides the page component's *body.html*. It includes the *head.html* from the page component and provides additional functionalities needed for performing various operations.

5. Go to AEM Siteadmin, open the page you created in Touch-Optimized UI.
   - If successful, the web page 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.



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




# Add the design

Creating and assigning a Design(er) in AEM allows you to enforce a consistent look and feel across your web site, 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 that 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.

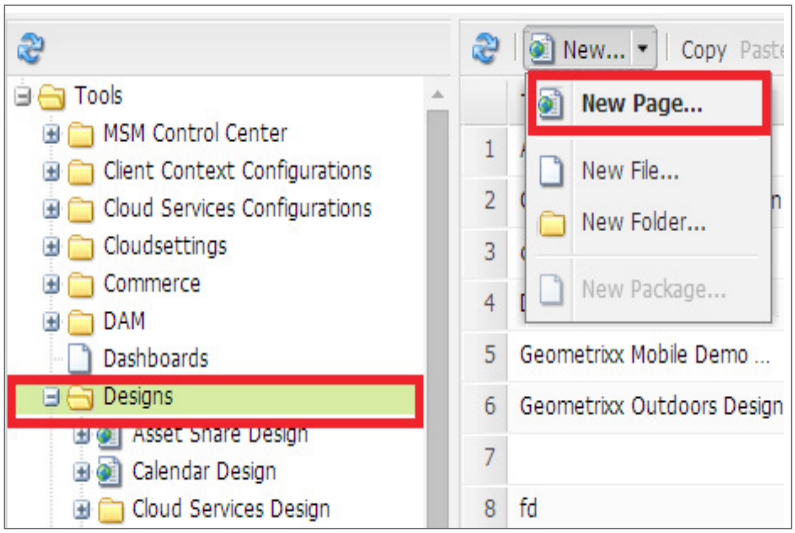
### EXERCISE 7.3 - Add a design to the page

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

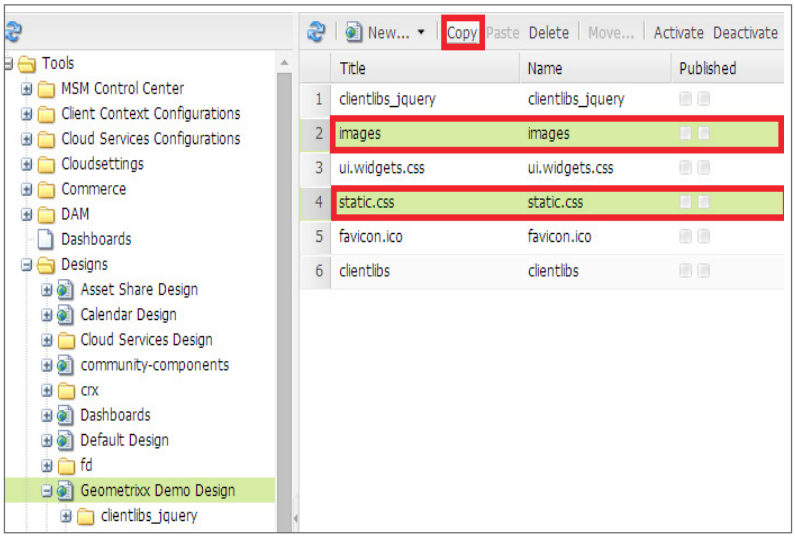
1. Access Tools using the following URL:
   *http://localhost:4502/libs/wcm/core/content/misc.html*

2. Select the Design folder and select **New** > **New Page**.

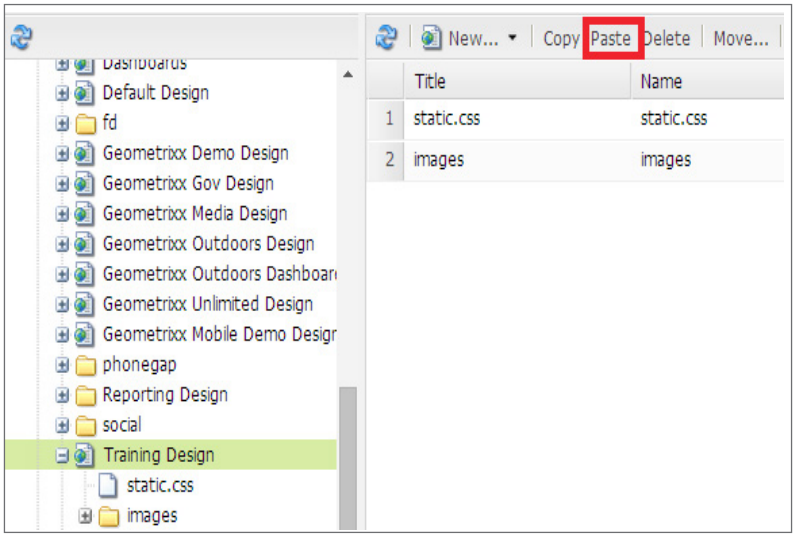
3. Enter the following details:
   Title: Training Design
   Name: trainingDesign




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

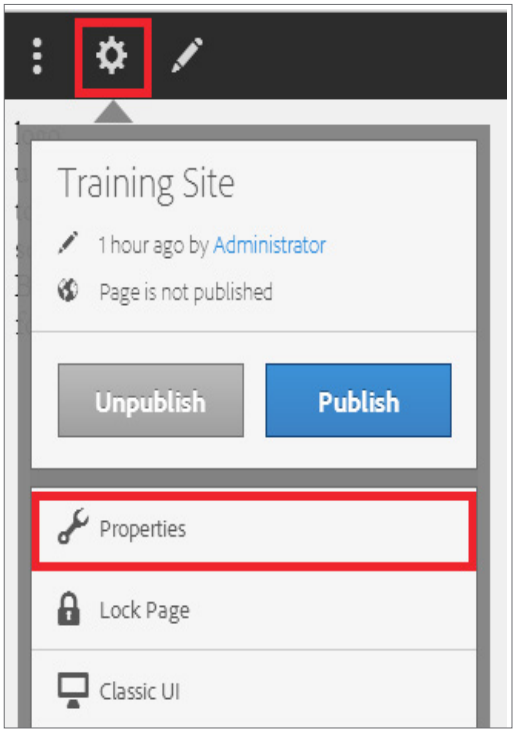


5. Select the newly created **Training Design** page and click **Paste**.

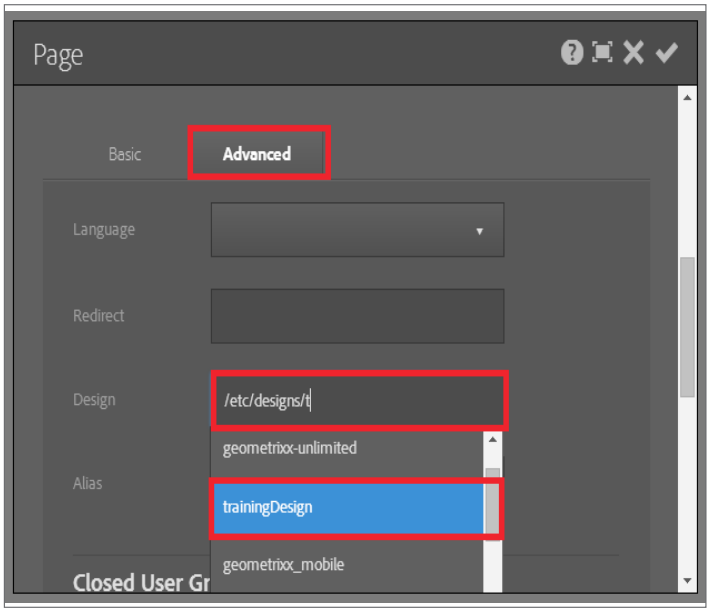



If you believe your implementation will have more than one Design(er), which is quite common, it is recommended you create a design structure that will allow multiple Design(er)s to be associated with one project. For example, */etc/designs/trainingDesign*, */etc/designs/trainingDesign/default*, and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

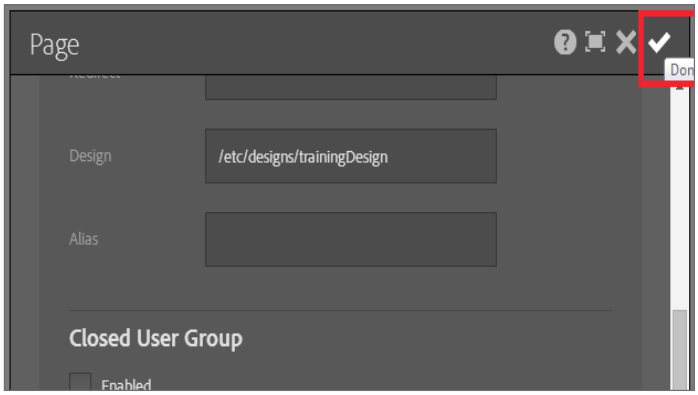
6. Open the root page of your website (Training Site).
7. Click the **Page Information** button on the top bar.
   The Page screen appears.



8. In the Page screen, go the **Advanced** tab. Type / in the Design field. The field displays all the designs available. Select **trainingDesign**.




9. Select **Done** from the top-right corner.




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

# Create components and include them into script

You can include a component into 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 the Design mode

- In JSP:

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

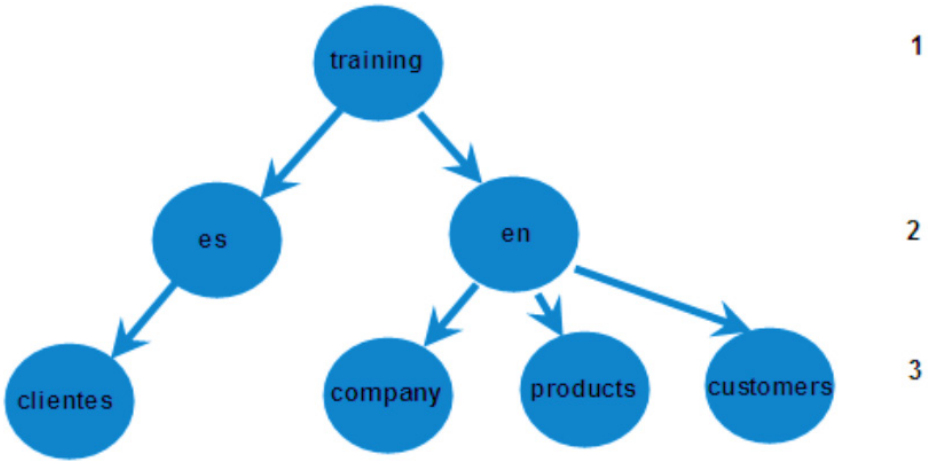
  - path: The path to the resource object to be included in the current request processing. If this path is relative, it is appended to the path of the current resource whose script is including the given resource.
  - resource type: The resource type of the resource to be included. If the resource type 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 web site 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 CQ5. Consider the following image, which represents a simple web site structure:

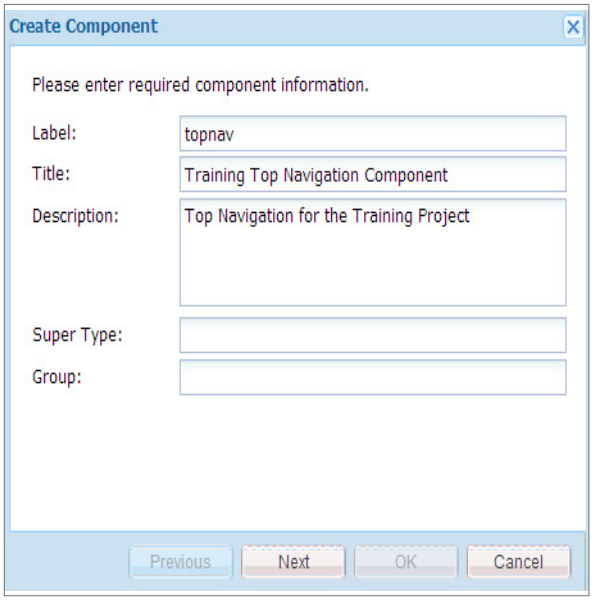



### EXERCISE 7.4 - Create a top navigation component and include it into script

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 and select **Create** > **Create Component**.
3. Provide the following details:
   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.
6. Right-click and select Rename. Change the extension from *.jsp* to *.html*.
   The name of the file becomes *topnav.html*.
7. Open *topnav.html* by double-clicking the file. Copy and paste the following code:

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

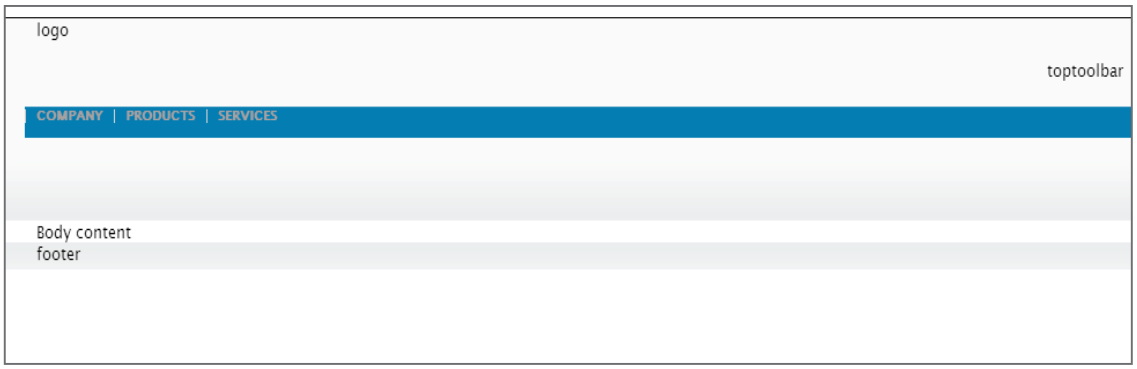
   The *data-sly-list* tag creates an array of the child nodes. The array then iterated one by one.

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

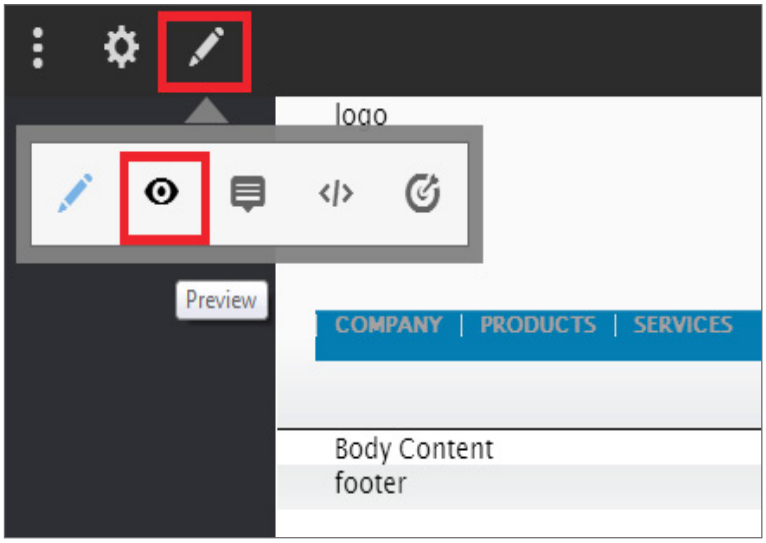
```
<div class="body_header header container_16">
  <div class="header_logo grid_8">logo</div>
  <div class="header_topnav grid_8 search_area">
      <div>toptoolbar</div>
 </div>
```

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

9. Open the */training-site/english* page that you created.
   If you have successfully created the top navigation component, it displays as follows:
   Note that the child pages are displayed as links.




10. Go to the **Preview** mode by clicking the **Edit** icon > **Preview** icon.




11. Move 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 have clicked the links and have navigated to the corresponding pages, you would have noticed that the navigation component disappears. In this case, you navigate to a page that doesn't have any children. The requirement was to display the child pages of the root node (English). So, ideally you need to do the iteration in the root node.

You need to use *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.
2. Right-click and select **Create** > **Create File**.
3. Create a new file *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*.

4. 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>
   ```

   The *data-sly-use* tag creates a reference to the JavaScript using the *topnav* variable. Using the *topnav* variable, you called the root variable the JavaScript returns. You then iterated 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.
2. Right-click and select **Create** > **Create File**.
3. Create a new file *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;
    }
}
```

4. 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>
```

5. Refresh the page you created.
   You shouldn't see any difference. The Java class you created works in the same way the JavaScript worked.

## Add 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 may 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:

- log 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 log message using the JavaScript file of a script

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

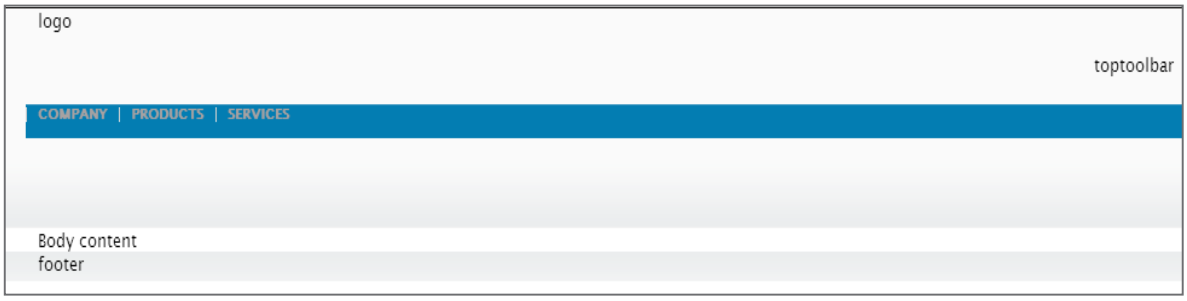
```
<ul class="topnav" data-sly-use.topnav="topnav.js" data-sly-list="${topnav.root.
listChildren}">
    <li data-sly-test="${!item.isHideInNav}"><a href="${item.path}.html">${item.
title}</a></li>
</ul>
```

2. Open *topnav.js* 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)
    };
});
```

4. Refresh one of the pages you created.




5. Go to the installation folder of AEM.
6. Navigate to *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.

---

**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());*

---

# 08

# AEM Authoring Framework: Dialogs

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:

- Creating dialog boxes for component
- Create design dialogs 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 as one could argue that rendering content comprises 50% of what you do as a 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 is similar to other dialogs you have used/created in the past: it gathers user input via a "form", potentially validates it, and then makes that input available for further use (storage, configuration, etc.).

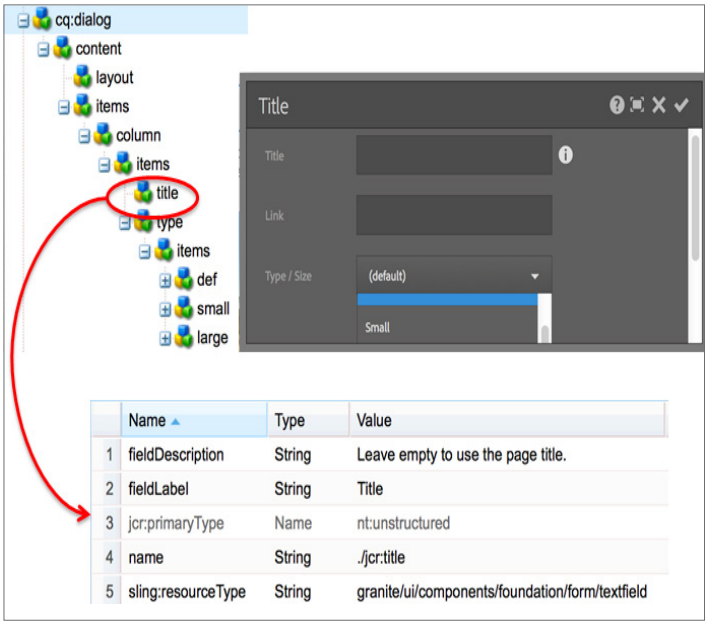
AEM uses two types of dialog boxes:

- One for Classic-UI
- One for Touch-Optimized UI

Both the Dialogs explained above 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 Dialogs are saved in the */libs/granite/ui/components/foundation/form* directory.
The root node of a Dialog should extend cq/gui/components/authoring/dialog.
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 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 let's examine the properties on the title widget itself. In the figure above, the title widget has 5 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". The "./" _construct, similar to the same convention in the U*x world, signifies "here, on this node".So, what this means is "When the Author enters content into the textfield widget and click OK, write the content into a property named title on this node."
- **sling:resourceType** - defines the type of input form, in this case, a "textfield" - single line, alphanumeric

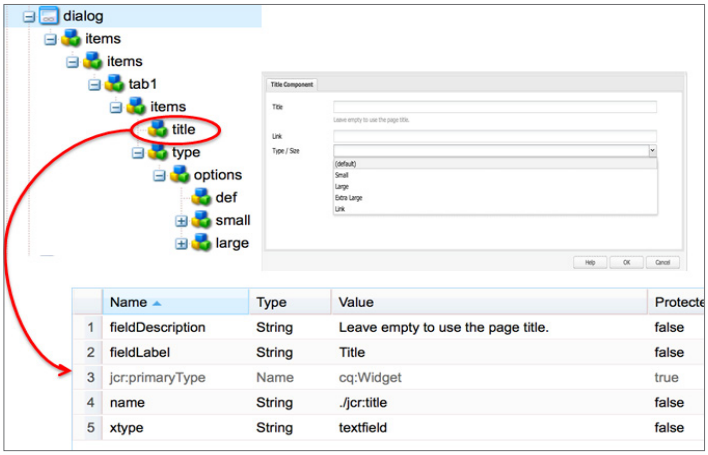
## Classic UI

AEM's Classic UI uses a widget library called ExtJS. ExtJS provides user interface 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) through the use of AJAX. It includes:

- High performance, customizable UI widgets
- Well designed and extensible Component model
- An intuitive, easy to use API

ExtJs is a cross-browser JavaScript library for building interactive web applications. It supports the rapid development of high performance, customizable User Interface widgets.

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. The input widgets for each panel are placed subordinate to the panel nodes. The "widget nodes" are be of node type "cq:Widget".

You will notice the cq:WidgetCollection node that is a child of the Tab Panel and the Widget Collection node that 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

But, 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 AEM6 environment where specific names must be used. The dialog box is one of those places. Rules:

- Dialog nodes must be of type cq:Dialog and must be named "dialog" ("design_dialog" for design dialog boxes" )
- Widget collections are always named "items" . Other nodes may be named "items", but Widget Collections are always named "items".

---

NOTE: There is one exception when a Widget Collection is not named "items". For a Selection Widget, the subordinate Widget Collection to the Widget is named "options" ..

---

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

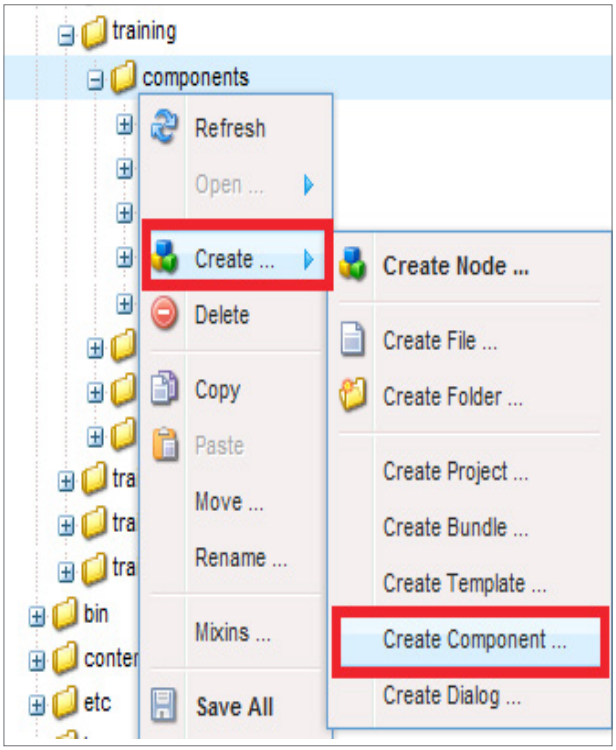
Now let's examine the properties on the widget itself. In the figure above, the title widget has 5 properties:

- **jcr:primaryType** - defines the node type, in this case: nt:unstructured fieldSubLabel - gives the Author more information, in this case: "(default equal to Page title)"
- **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". The "./" _construct, similar to the same convention in the U*x world, signifies "here, on this node".So, what this means is "When the Author enters content into the textfield widget and click OK, write the content into a property named title on this node."
- **xtype** - defines the type of input form, in this case, a "textfield" - single line, alphanumeric

## EXERCISE 8.1 - Create a Training Title component

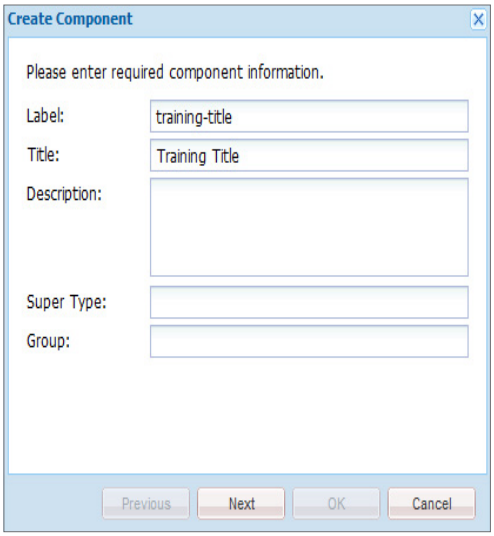
In this section, you will create a Title component that displays the page's title.

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



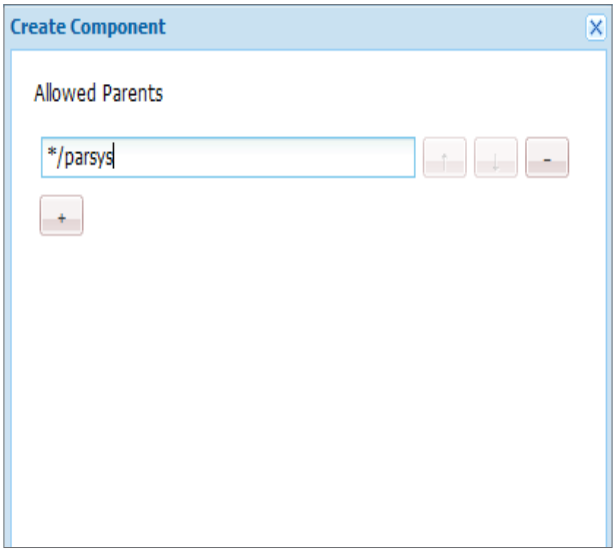

3. Add the following details:
   - Label: training-title
   - Title: Training Title

› Description: This is the training title component.




4. Click **Next**.
   The Advanced Component Settings screen appears.
   Click **Next**.

5. Click the + symbol for Allowed Parents. Enter the following:
   */parsys
   Click **Next**.



6. Click **OK** in the Allowed Children screen.
   The component is created.

7. Save the changes.

8. Rename the *training-title.jsp* page created to the *training-title.html*.

9. Double-click *training-title.html* to open it. Add the following:

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

10. Create a *title.js* file. Add the following:

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

11. Open *body.html* of the template 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>
```

12. Open one of the pages created. Note that the title appears on the page.

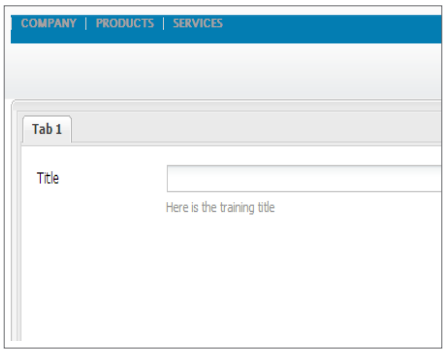
## EXERCISE 8.2 - Create a dialog for a Classic UI

In the previous exercise, you have 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 to change the title.

1. Select the training-title node. Right-click and select **Create** > **Create Dialog**. Create a dialog with the following properties:
   Label: dialog
   Title: Title Component
2. Save the node created.
3. Expand the node created in the previous step and locate Tab1. Right-click the tab1 node. Select **Create** > **Create Node**.
   Name: items
   Type: cq:WidgetCollection
4. Right-click the items node you created and create a node with the following details:
   Name: title
   Type: cq:widget
5. Add the following properties to the node:

| Name | Type | Value |
|---|---|---|
| name | String | ./custom-title |
| fieldLabel | String | Title |
| fieldDescription | String | Here is the training title |
| xtype | String | textfield |

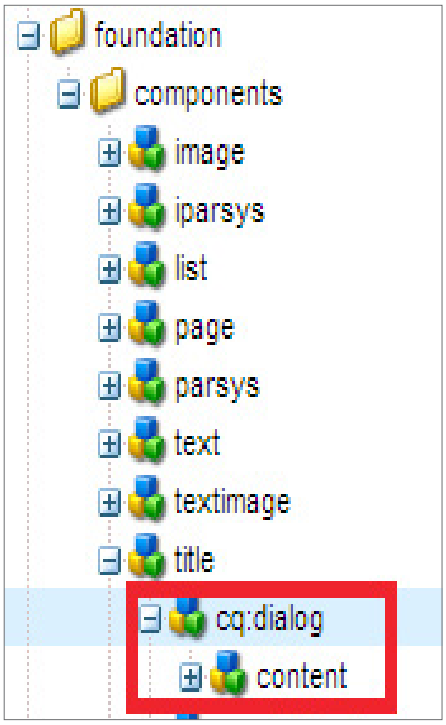
   You have created a dialog box for Classic UI now. In the next exercise, you will see how to add this to your page. The component that you created in the previous exercise is not added to the template, because you may need to change the title based on your needs. In this exercise, you will add the component to your page using the Content Finder.
6. Open one of the pages you created in Classic UI. Double-click the title you entered. Note that the dialog box appears.



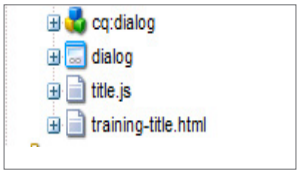
7. Now open the page in Touch-Optimized UI. Double-click the title you entered. Note that the dialog box appears. However, this dialog box is not optimized for Touch-Optimized devices. In the next exercise, you will create a dialog box for Touch-Optimized UI.

## EXERCISE 8.3 - Create a dialog for Touch-Optimized UI

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

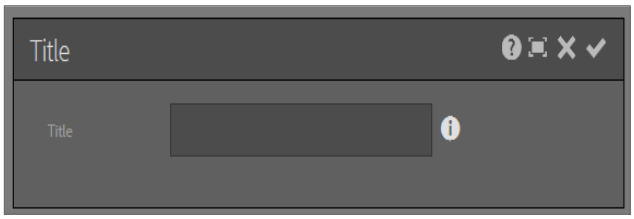



3. Paste it in the training-title node.




4. Move to the */apps/training/components/training-title/cq:dialog/content/items/column/items* node.
5. Delete the type node.
6. Select the title node and change the name property to *./custom-title*.
7. Open the page in the Touch-UI.

8. Double-click the Title displayed.
9. Note that a dialog box appears.




10. Switch back to Classic UI.
11. Double-click the Title component to open the Dialog box.
    Observe that the dialog box you saw in the previous exercise appears.

### 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 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.

- Include/import the appropriate java classes.
- Get a property whose value will become the parent of the list. The property should be a path.
- If the property is empty, use a default. ( perhaps a static path or path to current page )
- 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)
- Set up a page iterator, filtering by valid pages
- Iterate over the children
- For each child, write out a link
- Create a dialog box
- Place an input widget on the dialog box so the author can enter the list parent.
- Edit *body.html* to include the component.

# Use Design Dialogs for global content

As discussed earlier, a Design(er) can be used to enforce a consistent look and feel across your Web site, as well as share global content. This global content is editable when using a Design Dialog. 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 is almost identical to creating a "normal" Dialog - the only difference being the name of the Dialog itself (i.e. dialog vs. *design_dialog*).

- 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 than using the properties object as we have done so far.
Design dialogs are almost identical to component dialogs 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, 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.

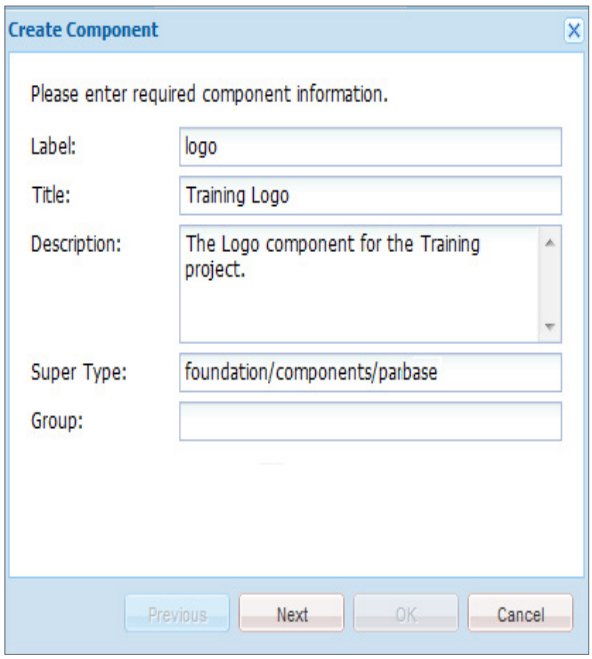
Therefore, it is important to note is the use of the resource SuperType foundation/components/parbase. The sling:resourcesSuperType property is a key component as it allows components to inherit attributes from other components, similar to subclasses in object oriented languages such as Java, C++, and 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 8.4 - Create a Logo component

In this exercise, you will create a logo component. You will use a design dialog to create the logo component, because a logo is a global content. It doesn't 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 get updated in all the pages that use the template.

1. Log in to CRXDE Lite. Right-click the Components folder of the Training site. Select **Create > Create Component.**
2. Enter the following details:
   - Label: logo
   - Title: Training Logo
   - Description: The Logo component for the training project.

› Super Type: foundation/components/parbase




Click **Next** till you reach the last screen, and then click **OK**.

3. Open the file *logo.jsp*, enter the following code, and then **Save**.

logo.jsp

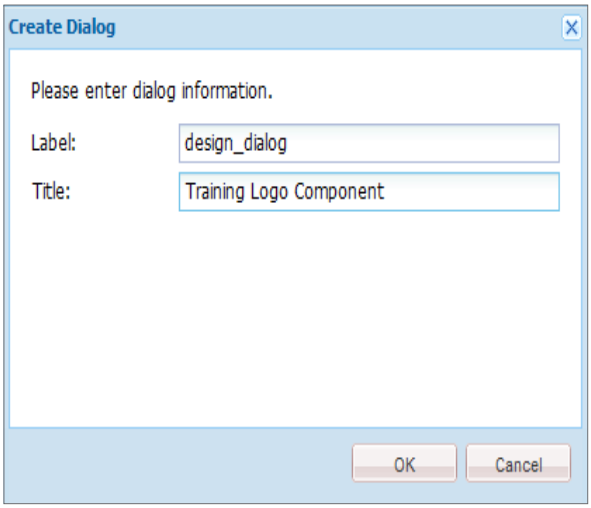
```
<%@include file="/libs/foundation/global.jsp"%><%
%><%@ page import="com.day.text.Text,
    com.day.cq.wcm.foundation.Image,
    com.day.cq.commons.Doctype" %><%
   String home = Text.getAbsoluteParent(currentPage.getPath(), 2);
   Resource res = currentStyle.getDefiningResource("fileReference");
   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>
```

4. Open the file *header.html* of *page-content* and replace the code that you used for logo.

header.html

```
<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>
```

5. Create a new design dialog for the logo Component.




6. 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 so that it is a peer of the tab1 node.
   - e.g. */apps/training/components/logo/design_dialog/items/items*

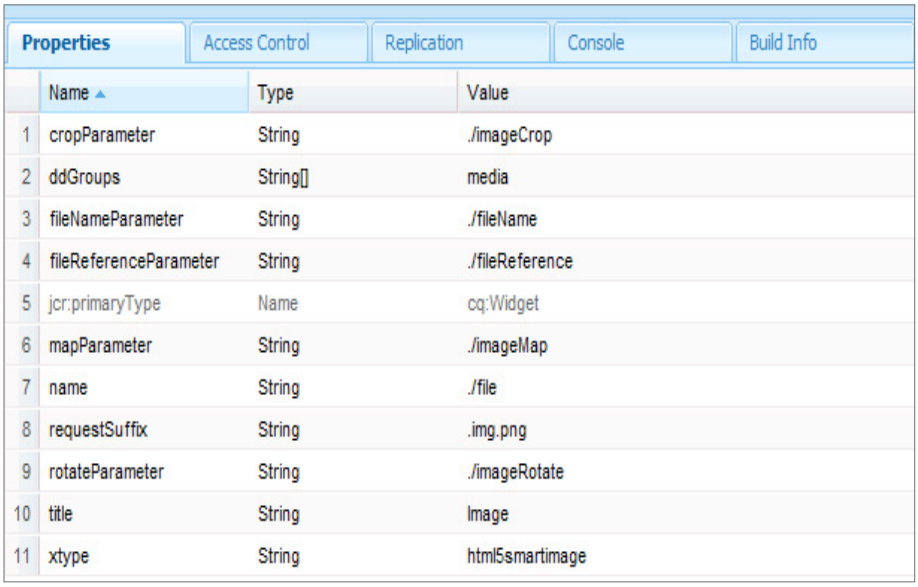
   In addition, you may have noticed, this Widget was pasted in the location where tabs are typically located. The smartimage xtype (along with few others) is unique in that it provides a better content author experience when **"**casted**"** as a tab.

---

NOTE: Instead of always reinventing the wheel, it is often more efficient to copy and paste existing Dialogs 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.

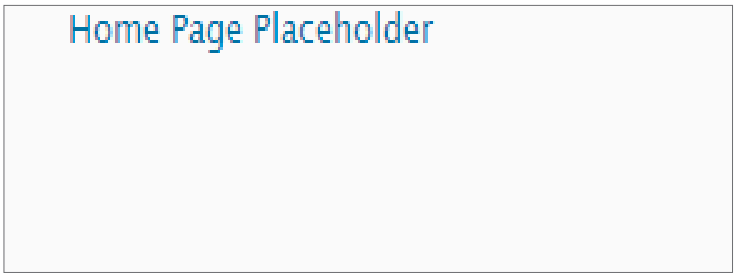
---

7. Review the properties associated with this **smartimage** widget.

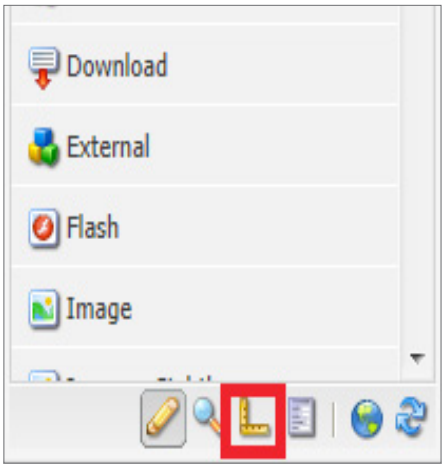



| | Name | Type | Value |
|---|---|---|---|
| 1 | cropParameter | String | ./imageCrop |
| 2 | ddGroups | String[] | media |
| 3 | fileNameParameter | String | ./fileName |
| 4 | fileReferenceParameter | String | ./fileReference |
| 5 | jcr:primaryType | Name | cq:Widget |
| 6 | mapParameter | String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

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




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




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


11. Drag and drop the image.
    › 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 that you need to obtain the following result: 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-line editing
- Refresh of a dialog box or page after an author action

To add inplace 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 from the Content Finder functionality, the following are required:

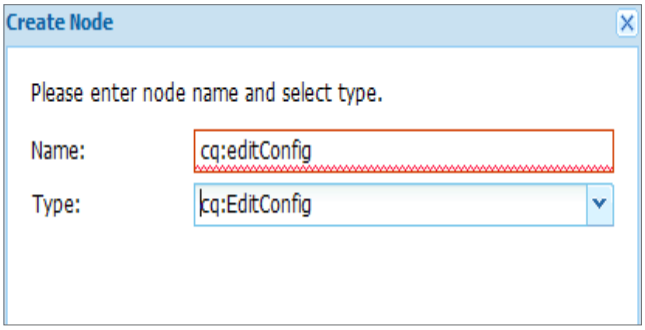
- Node of type cq:editConfig that is a child node to the cq:component node
- Configuration node that is a child of the cq:EditConfig node. This node defines what action you are configuring. Asset node that is a child of the cq:dropTargets node. This node defines what types of assets the paragraph will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

### EXERCISE 8.5 - Enable inplace editing in the Title component

In this exercise, we will enable inplace editing in the Title component. Inplace 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.

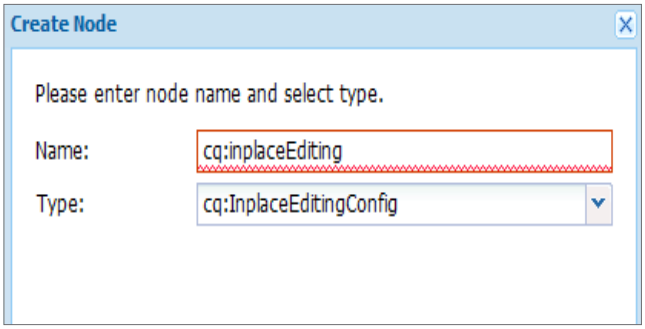
1. Locate the training-title component you created.
2. Right-click and select **Create** > **Create Node**.

3. Create a node with the following details:
   Name: cq:editConfig
   Type: cq:EditConfig




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

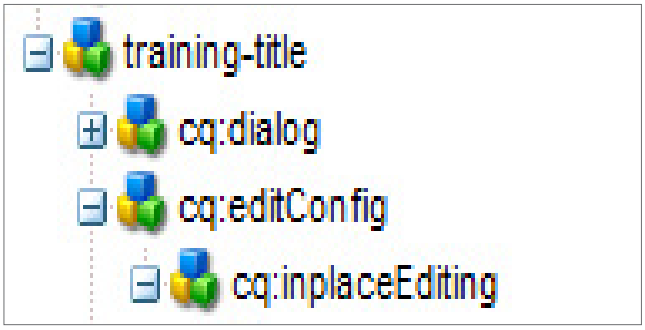
5. Create a node with the following properties:
   Name: cq:inplaceEditing
   Type: cq:inplaceEditingConfig




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. Open any of the page that you created.

9. Click the Title component.
Note that the cursor appears in the text box allowing you to edit the title without opening the dialog box.

# 09

# AEM Authoring Framework - Foundation Components, Internationalization, and Client Libraries

In this chapter, you will create a website using the authoring framework in AEM. 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 web site. They range anywhere 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 scratch. You can also copy the out-of-the box components to your apps folder. Then customize them to your requirement.

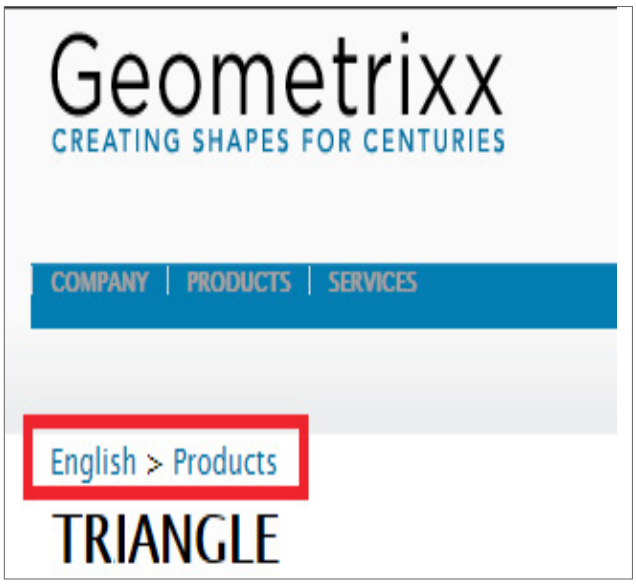
## EXERCISE 9.1 - Include a breadcrumb foundation component

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. 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**

```
<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 always - we never modify anything in `/libs` so: Create the foundation */components* structure in /apps and copy the Foundation breadcrumb component to the *.../components* folder.
2. Modify the code to allow the author to enter the start of the crumbs as a path, instead of as a level.
3. Modify the Design Dialog to allow the author to enter a path, instead of a level.
   If you have previously entered a number (level) with the same property name (e.g.;absParent), the breadcrumb component will throw an error, as it is looking for a string that is a path and encounters a number. To fix this, merely navigate to the design element (*/etc/designs/trainingDesign/jcr:content/breadcrumb* and delete the *absParent* property. Also remember that the Geometrixx application also uses the foundation breadcrumb component. So you will have to modify the geometrixx design in the same fashion to get the geometrixx pages to render properly.
4. Test your code.
   Examine the repository to see the values written into the design when author enters values into the design dialog box and saves them.

# Include the paragraph system

The paragraph system is the main content area of a web page. By adding the paragraph system 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 wanting 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 allows content authors the ability to dynamically add, delete, move, copy, and paste "paragraphs" on a Page, not to mention the ability to use the column control Component to structure your 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 that developers then have to maintain.

## EXERCISE 9.2 - Include the paragraph system component

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

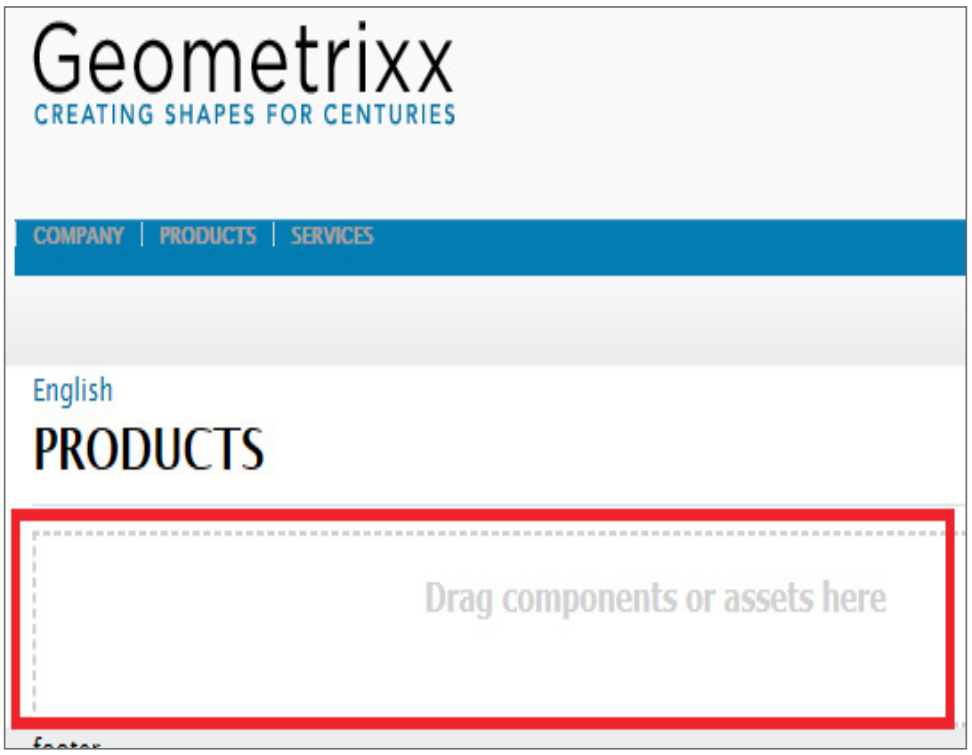
1. Open *body.html* by double-clicking it.
2. Select the following code:

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

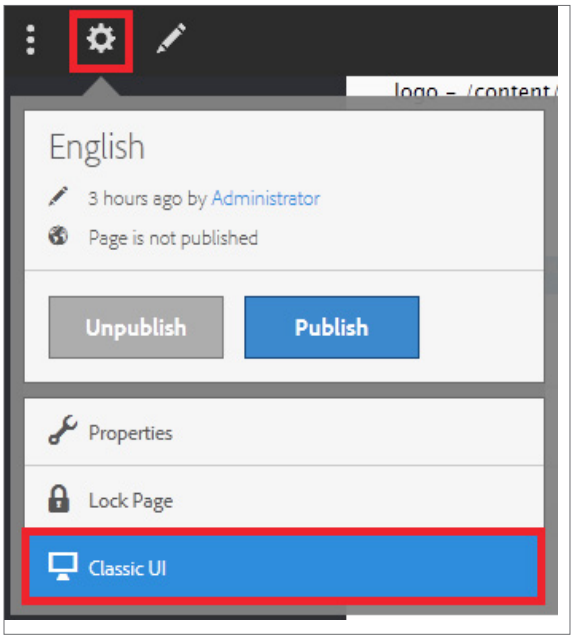
3. Replace it with the following code:

```
<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>
```

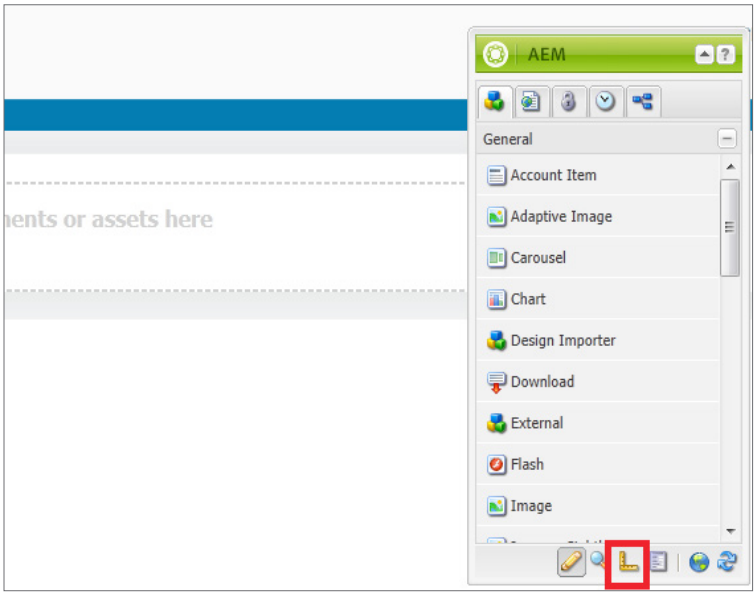
4. Open one of the pages you created. (For example, Products.)
   It displays as follows:
   Note that the Paragraph component is displayed, where you can drag and drop another out-of-the-box component.



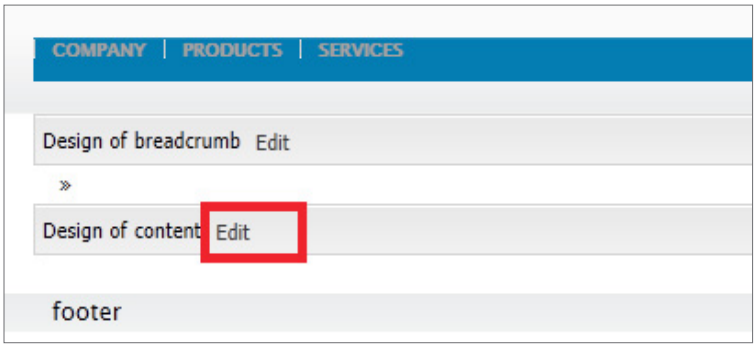
5. Click the Page Information icon and select Classic UI.
   The page appears in Classic UI.




6. Select **Design** view in the sidekick.

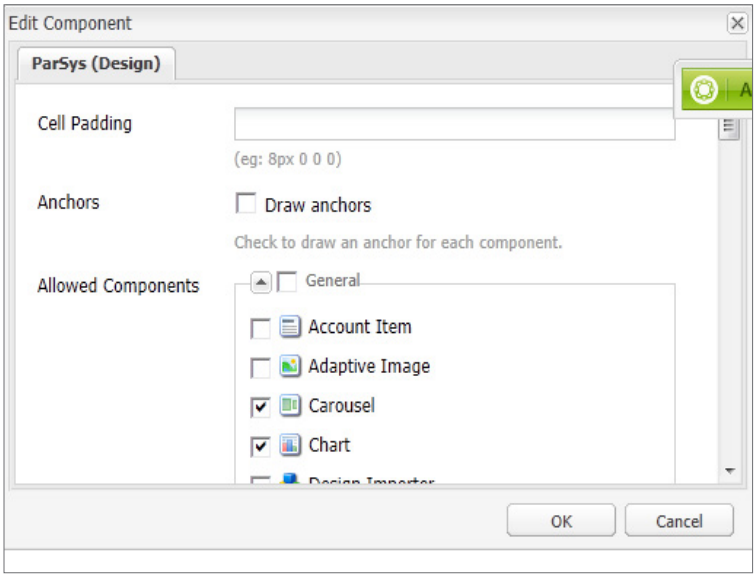


7. Click **Edit** in the Design of Content that appears.



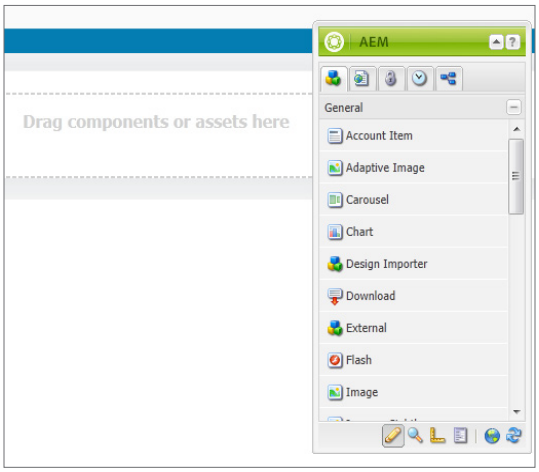

The **Edit Component** dialog box appears.

8. Expand the General section and select some of the components as shown below. (Ensure that you included the Image component.) Click **OK**.

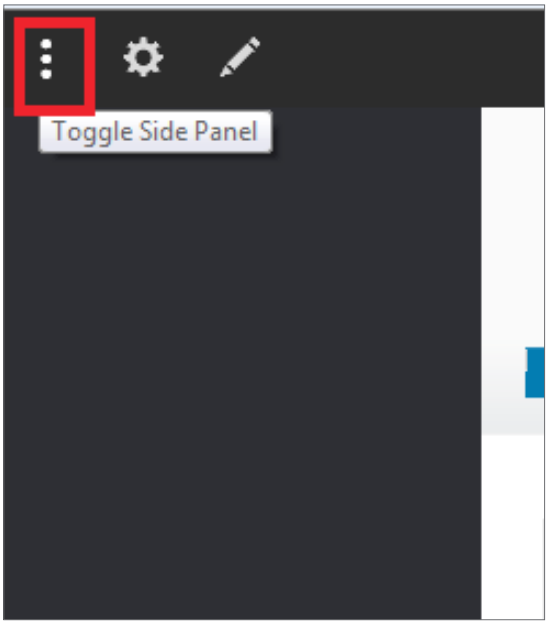



9. Observe the refreshed page.

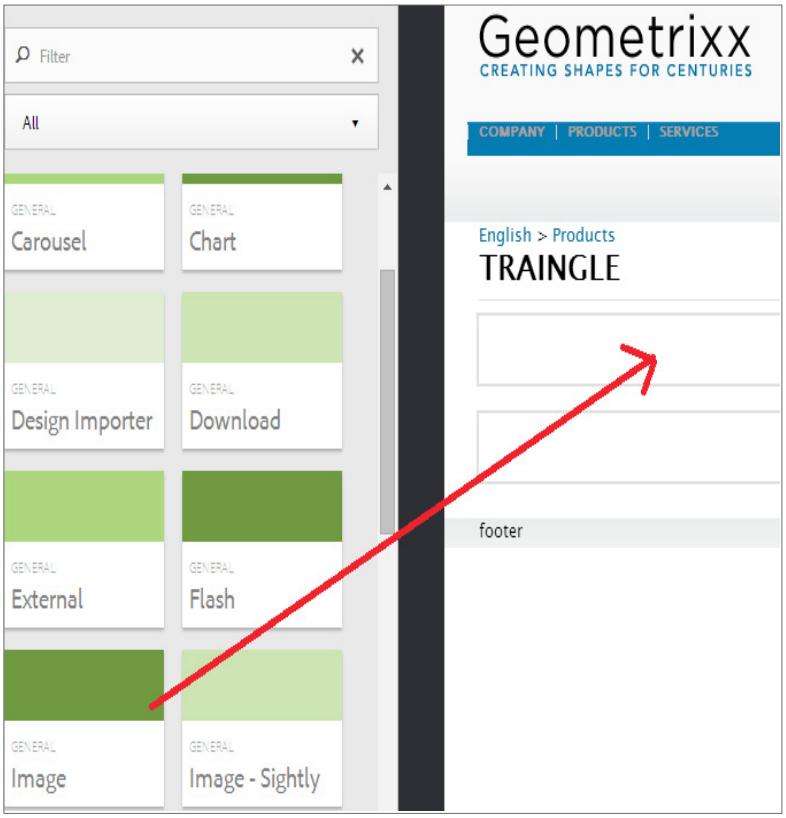
10. Click the Arrow in the collapsed sidekick to return to the Edit mode.
    The sidekick now displays the components you selected in the previous page.




11. Close the **English** page you opened. Open it again from the Site Admin, so that it appears in the Touch-UI mode.
12. Go to the **Edit** mode if you are not already in this mode.
13. Click the **Toggle Side Panel** button.

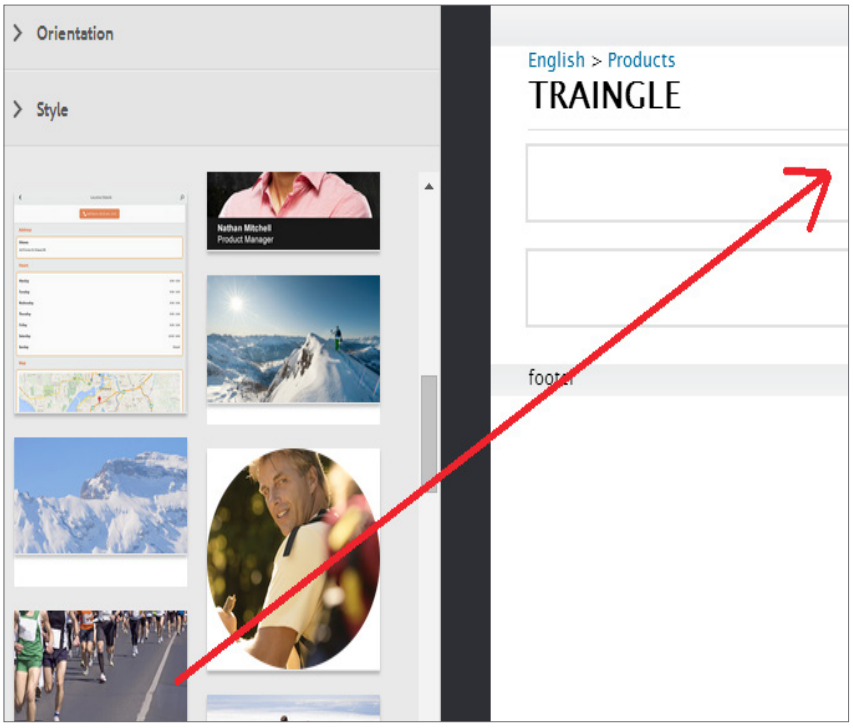


14. Go to the **Components** tab. Drag and drop the Image component to the Paragraph system.

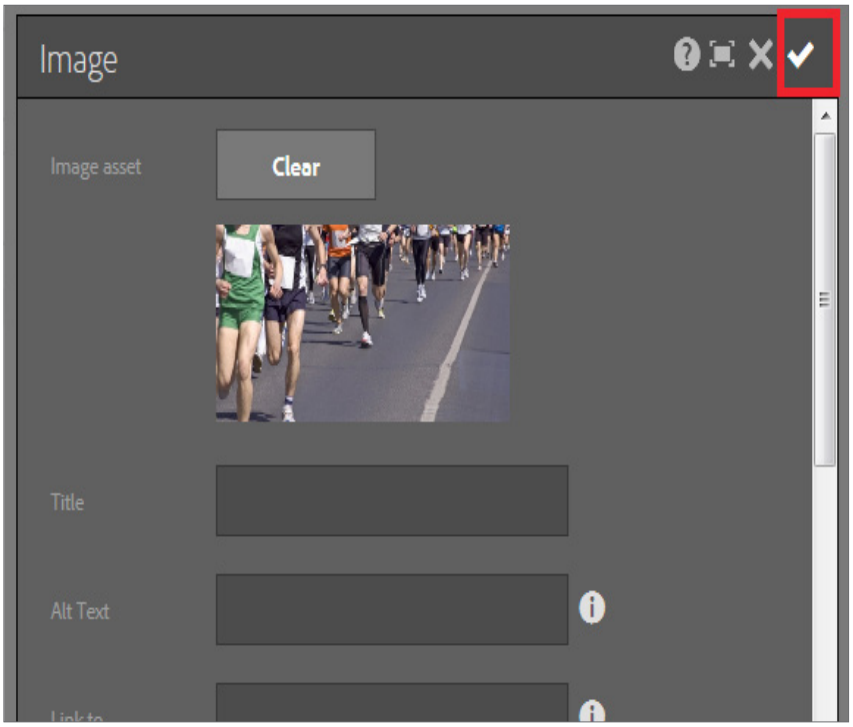



15. Go to the **Asset** tab.
    The Asset tab appears with digital assets.

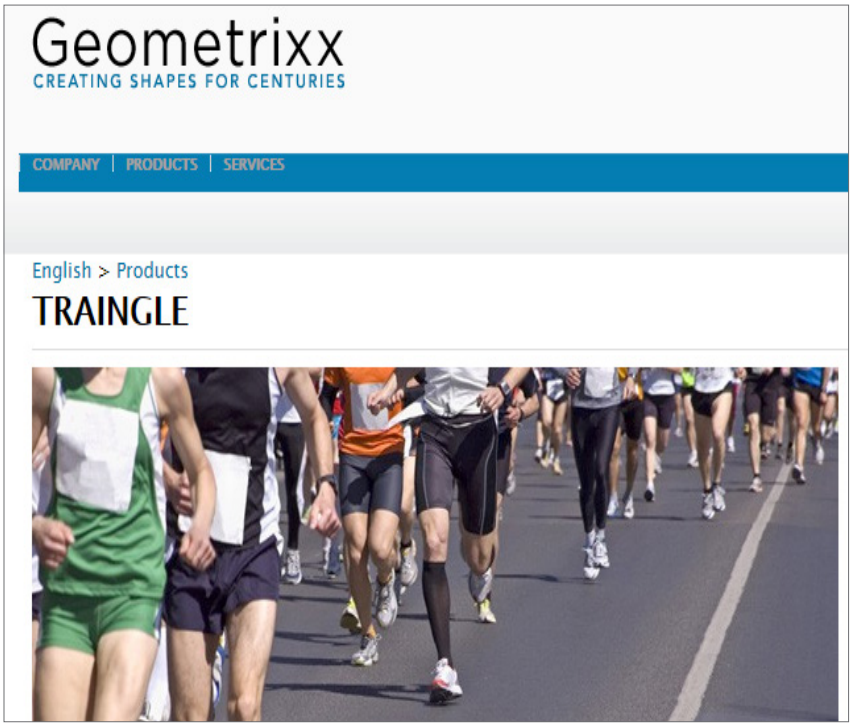
16. Drag an image from the Asset tab and drop it to the Image component.




17. Double-click the component to open it in the **Edit** mode and click **Done**.



18. Refresh the page to see the image displayed in the page.




## EXERCISE 9.3 - Use the Toolbar component

One of the requirements in the wireframe was 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, Toobar. You can also set a property named cq:toolbars and assign a value to filter the nodes based on your requirement.

1. Log in to CRXDE Lite. Locate the Training Site.
2. Go to the Components folder and open the *footer.html*.
3. Replace the placeholders for Tool Bar as shown below:

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

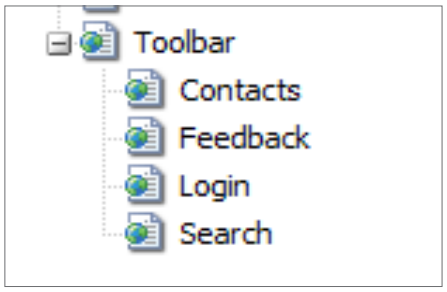
4. Open the *header.html* page.

5. Replace the code as shown below:

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

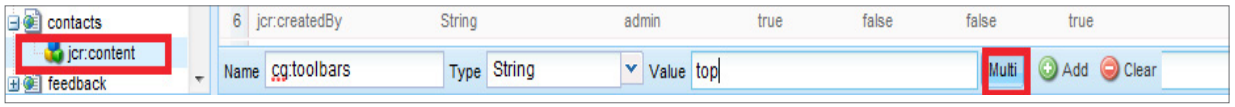
6. Open the Authoring environment and create a Toolbar page in the English folder. Ensure that the folder name is Toolbar.

7. Under Toolbar, create the following pages:
   - Contacts
   - Feedback
   - Login
   - Search



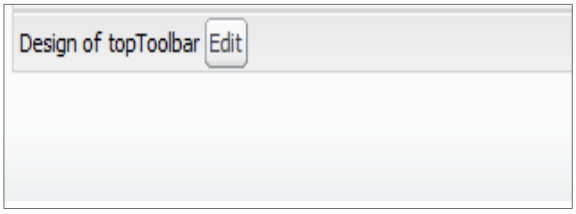

8. In CRXDE Lite, locate */content/TrainingSite/English/toolbar/Contact*.

9. Select the content node and add the following multi properties:
   - Name: cq:toolbars
   - Type: String
   - Value: top

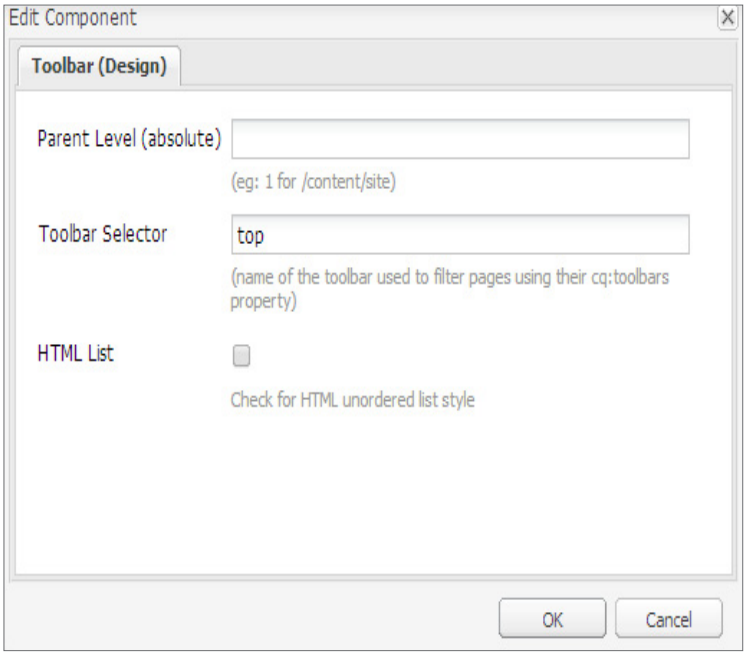



10. Select **Multi** and click **Add**.

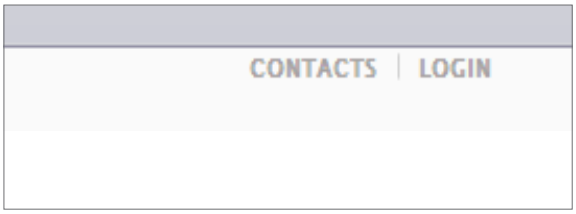
11. Select the Login page and add the following properties. Select **Multi** and click **Add.**
    - Name: cq:toolbars
    - Type: String
    - Value: top
12. Select the Feedback page and add the following properties. Select **Multi** and click **Add**.
    - Name: cq:toolbars
    - Type: String
    - Value: bottom
13. Select the Search page and add the following properties. Select **Multi** and click **Add**.
    - Name: cq:toolbars
    - Type: String
    - Value: bottom
14. Open the web site page you created. Using the sidekick, go to the Design mode.
15. Click **Edit** in the Design of topToolbar.



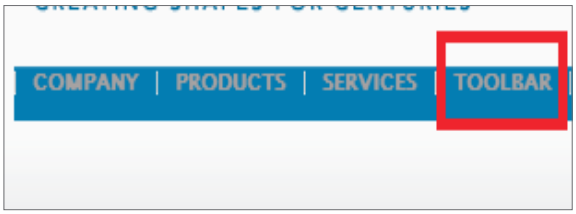

16. In the Toolbar Selector, enter *top*. Select HTML List check box.



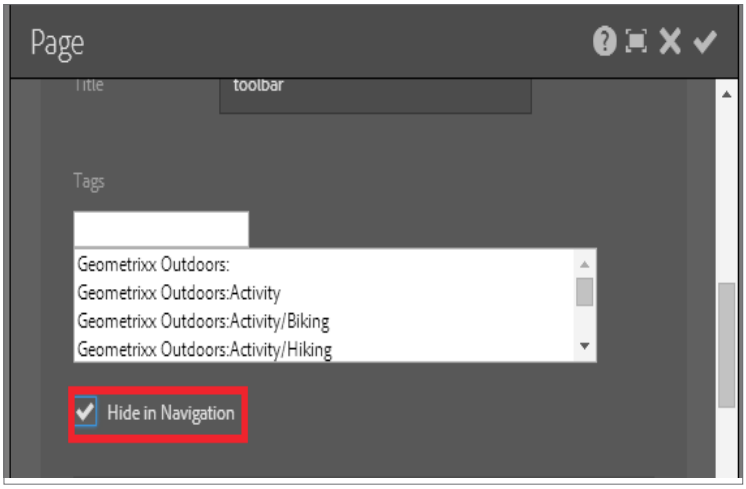
Observe that only the pages with *top* property appear in the top.




17. Click **Edit** in the Design of bottomToolbar.
18. In the Toolbar selector, enter *bottom*. Select HTML List check box.
    Observe that only the pages with *bottom* property appear in the bottom.
    Also, notice that the top navigation bar now displays a Toolbar link that is not needed.




19. To remove the Tool bar link from the top navigation bar, open the Toolbar web page. Go to the Page Properties and select the Hide in Navigation option. Then, click **Done**.




20. Observe that the refreshed page doesn't contain the Toolbar link.

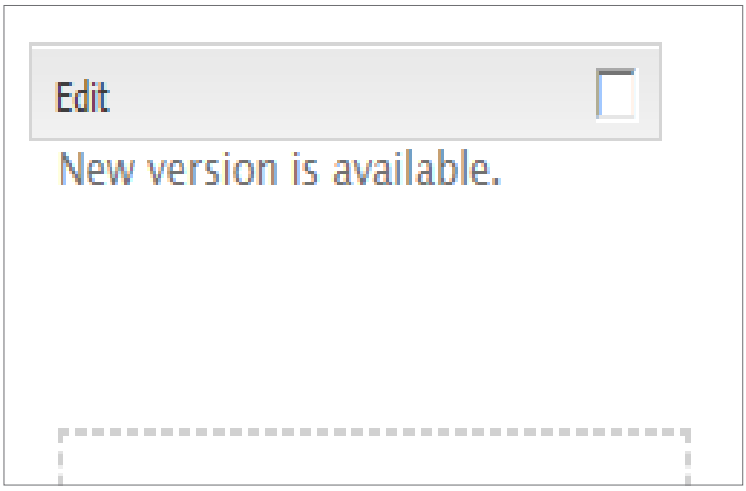
## EXERCISE 9.4 - Include iParsys component

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

1. 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. Refresh the page in the Classic UI.
4. Drag and drop the Text component to iParsys.




5. Added any text.
6. Create a child page to the current page and observe that the text you added appears in the child page.

# Internationalize the Authoring Interface

The AEM Authoring interface ships in 7 languages allowing authors to enter and manage content in their language of choice. When you create your own components and dialog boxes, as we have 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*.**

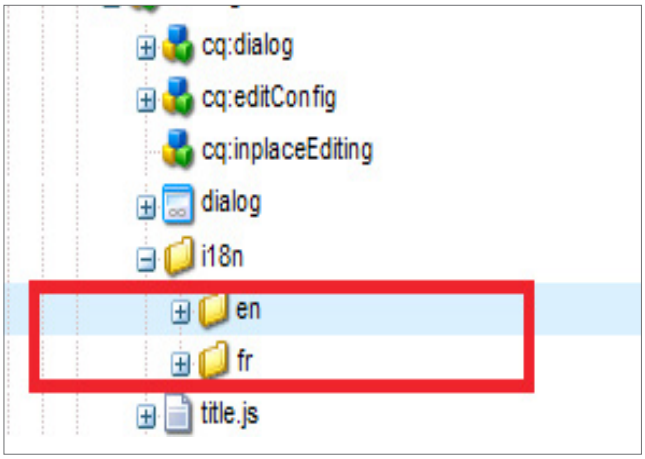
The children nodes (nodeType *sling:Folder + mixin mix:language*) of the *i18n* node represent languages and are named using the ISO code of the languages that are supported (e.g. en, de, fr, etc.). Below these language nodes are the message bundle nodes (nodeType sling:MessageEntry), which will contain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

The location of the i18n node within the repository determines the scope of the message bundles. If located in a project directory (e.g. /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.

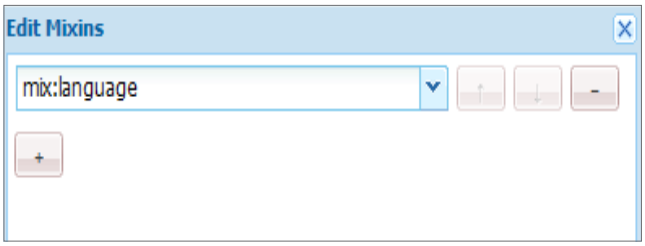
### EXERCISE 9.5 - Internationalizing the Title component's GUI

In this exercise, you will internationalize the GUI instruction provided in the Title component. You would 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**.
3. Right-click and select **Create** > **Create Folder**.
4. Create a folder named, *i18n*.
5. Create two folders underneath: *en* and *fr.*



6. Select en folder and click the Mixins button.
7. Click the + symbol in the Edit Mixins node and select mix:language.




8. Add the following property to the en node:

| Name | Type | Value |
|---|---|---|
| jcr:language | String | en |

9. Do step 6 through 8 for the fr node.
   Ensure that you add jcr:language property as fr.
10. Select en node. Right-click and create a node with the following details:
    Name: title
    Type: sling:MessageEntry
11. Add the following properties to the Title node:

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

12. Do step 10 through 11 to the fr node.
    Ensure that the value of the sling:message property is "*Enter French title here.*"
13. To internationalize the Touch-Optimized UI, go to the cq:dialog node. Navigate to **content** > **items** > **column** > **items** > **title**.
14. Change the value of the fieldLabel property to *i18n-title*.
15. To internationalize the Classic UI, go to the dialog node. Select items > title.
16. Change the value of the fieldLabel property to *i18n-title*.
17. Open the page you created.
18. Double-click the Title component.
19. Note that the label for the text box is changed.

To view the label in French, go to user admin. Select a user and change the language of your choice to French. Then login and double click the component.

# Add Client Libraries

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

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

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

## Client- or HTML Libraries

AEM has introduced a very interesting concept: Client Libraries, aka "clientlibs".
Client Libraries are "folders" (nodes of node-type cq:ClientLibraryFolder) that contain the client-side assets, CSS and JS files and required resources, e.g. images or fonts.

There can be unlimited client library folders. The folder content can be loaded individually and at any given time.

## Client Library Conventions

Create client libraries either under /etc/clientlibs or within the component folder.
A client-library "folder" is created as a node with 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

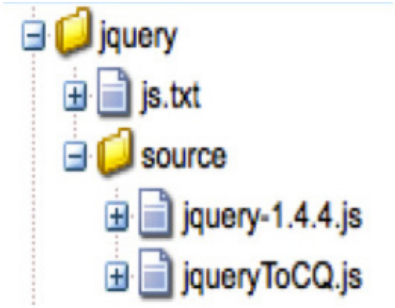
Create sub folders for the CSS and JS files, e.g. /scripts or /styles. Create a css.txt and/or a js.txt file and add the .css and .js files that will be minified and loaded by templates or components. If the assets are in a sub-folder, the .txt file must start with "#base=sub-folder". Resources, like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

Client libraries are loaded with the <cq:includeClientLib> tag. The <cq:includeClientLib/> 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 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, 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, else the current library will not be included.

**Embed**: The libraries of categories listed in "embed" will be included to 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 to 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.

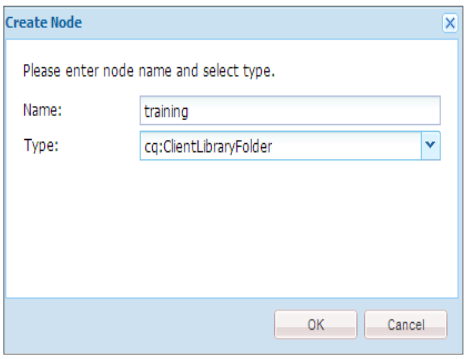
**css**: A list of comma-separated client library categories. This will include all css libraries from the listed categories.

**theme**: A list of comma-separated client library categories. This will include all theme related libraries (both 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. 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: categories
   Type: String[]
   Value: training.edit
3. Create a folder named, *scripts*.

4. Inside *scripts*, create JavaScript file named *trainingEdit.js*.

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

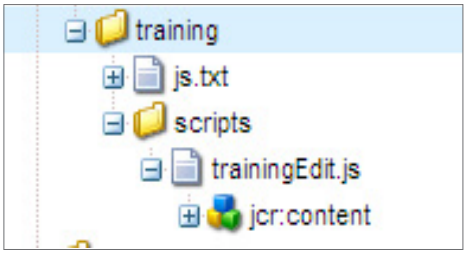
```
/*Training JavaScript*/
alert("This is from ClientLib");
```

6. The training folder you created in step 1, create a file, js.txt.

7. Add the following in js.txt:

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

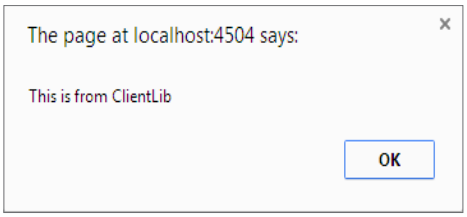
8. Following must be the directory structure:




9. The client libraries are now ready, to be included into 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 code that you added after you test this. It is provided for the demo purpose. Addition of this script makes navigation in the site difficult.

**Example: Adding 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>
```

# 10

# Mobile Websites

In this chapter, you will learn about the following:

- Responsive design
- Mobile components
- Creation of mobile websites using MSM
- Emulators

## Responsive design

The website you created is optimized to view in desktops and laptops. However, your users may want to view your website in hand-held devices, such as tablets and mobile phones. Therefore, you need to ensure that the website is optimized to view in all the devices to provide your users a better experience.

Responsive Design allows you to create sites that provide optimal viewing experiences 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 both 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 web design


Responsive web design is an approach to web design aimed at implementing sites that provide an optimal viewing experience for all web site 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 web 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 web design are:

- fluid grids
- flexible images
- CSS3 media inquiries to detect screen resolution

Responsive web 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 web 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 web design works well for sites where users consume content. However, it does not work well when you want your customer 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 web 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 can't fit into a grid layout.

## Pros and Cons of responsive web design

### Pros

- A single design for all devices ensures a consistent experience across devices
- A single URL for all devices
- A single code base for your website
- Less expensive to maintain

### Cons

- 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
- User journey for mobile users is typically different than 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 in various devices

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

- responsive-1200px.css: Styles for all media that are 1200 pixels wide or more.
- responsive-980px-1199px.css: Styles for media that are between 980 pixels and 1199 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.

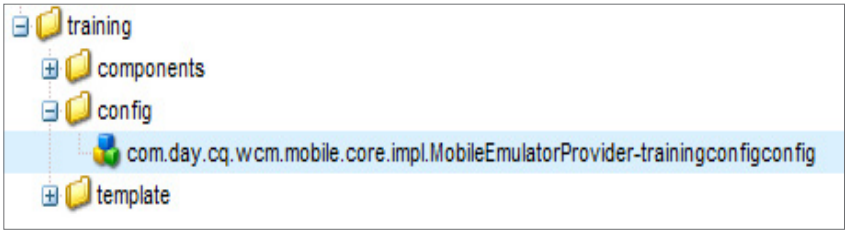
1. Include the client library folder to your template.
   Use the client library files created for Geometrixx Media. Open *header.html* and include the following code at the top of the page:

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

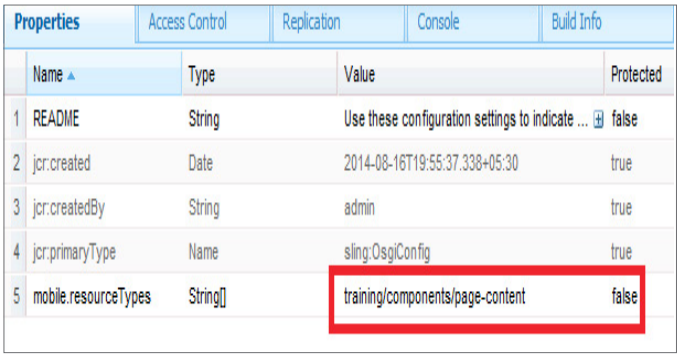
2. Include the device list in the sidekick. Open *header.html* and include the following code immediately after the code you added in step 1:

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

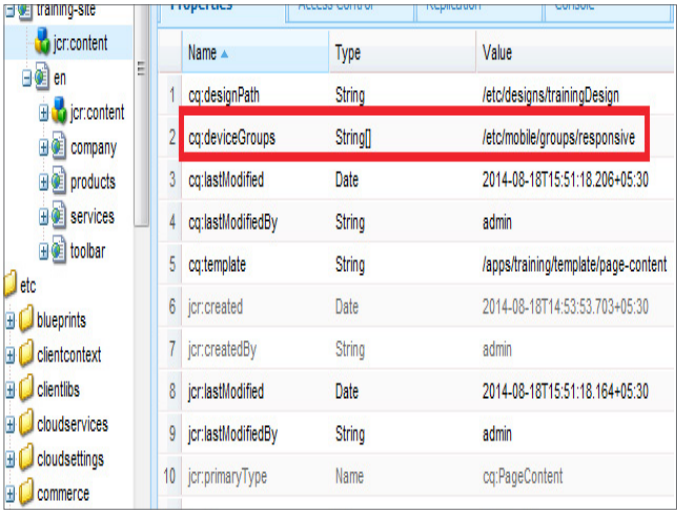
3. To enable the device simulator to support your pages, register your page components with the MobileEmulatorProvider factory service and define the mobile.resourceTypes property. In the Config folder, create a node with the following details:
   Name: *com.day.cq.wcm.mobile.core.impl.MobileEmulator-trainingConfig*
   Type: *sling:OsgiConfig*



4. Add the following to the node property:
   Name: *mobile.resourceTypes*
   Type: *String[]*
   Value: *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: *cq:deviceGroups*
   Type: *String[]*
   Value: */etc/mobile/groups/responsive*




6. In the Classic mode, open the one of the pages you created.

7. Go to the Preview mode and select the Device icon.


8. Switch between the devices to see the way your page appears.
   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 come quite handy every time that 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)<br>This method finds the device groups currently assigned to the given page or its closest ancestor. |
| static String | getDeviceGroupSelector(SlingHttpServletRequest request)<br>Returns the last selector found in the request URL. |
| static boolean | hasCapability(SlingHttpServletRequest request, DeviceCapability capability)<br>This method retrieves the DeviceGroup from the current request and checks the group whether it offers the given capability. |
| static boolean | isDeviceGroup(Page page)<br>Checks whether the given Page represents a WCM Mobile Device Group. |
| static boolean | isDeviceGroup(Resource resource)<br>Checks whether the given Resource represents a WCM Mobile Device Group. |
| static boolean | isMobileRequest(SlingHttpServletRequest r)<br>True if the request's user agent is a mobile device. |
| static boolean | isMobileResource(Resource r)<br>True if given Resource is to be handled as a mobile resource. |
| static boolean | isNoMatch(String[] selectors) |

The most important method for you is the method—hasCapability, which indicates if you should render or not, a piece of content depending on the capability of the emulator to handle the kind of content.

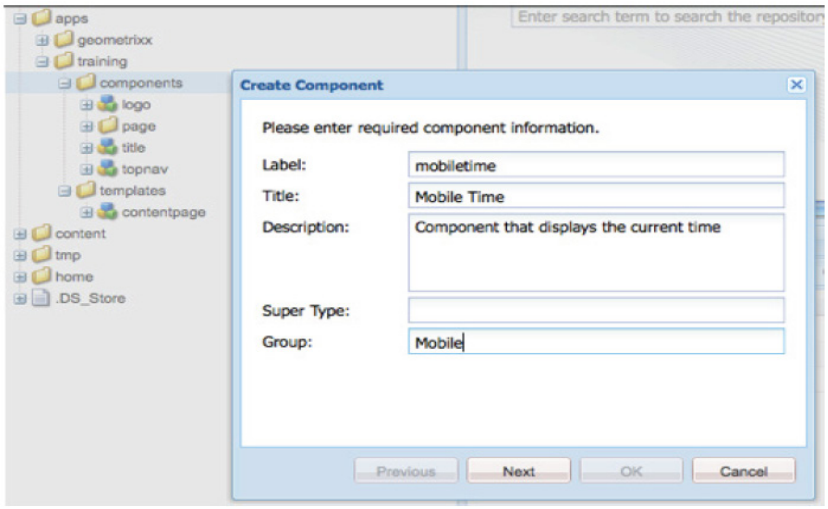
The interface DeviceCapability on the other hand has the following fields:

| Field Summary | |
|---|---|
| static DeviceCapability | CAPABILITY_CSS |
| static DeviceCapability | CAPABILITY_DEVICEROTATION |
| static DeviceCapability | CAPABILITY_IMAGES |
| static DeviceCapability | 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*. Select **Create > Create Component.**
2. Enter the desired Component Label, Title, Description—then click **Finish**.
   - Label = mobiletime
   - Title = Mobile Time
   - Description = Component that displays the current time
   - Group= Mobile




3. Click **Next** and then on next enter */*parsys for allowed parents.

4. Click **Next** again and your component will be created.

mobiletime.jsp

```
<%@ 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=setTimeout('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>
<% } %>
```

5. Select your component and create an empty dialog for it (You need this so that it appears in the list of components available to paragraph system).
6. Open the English page of the Geometrixx Mobile site. Go into Design mode and add your mobiletime component to the design. Return to Edit mode.
7. From the sidekick, drag the Mobile Time component into the paragraph system of the Geometrixx Mobile > 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
```

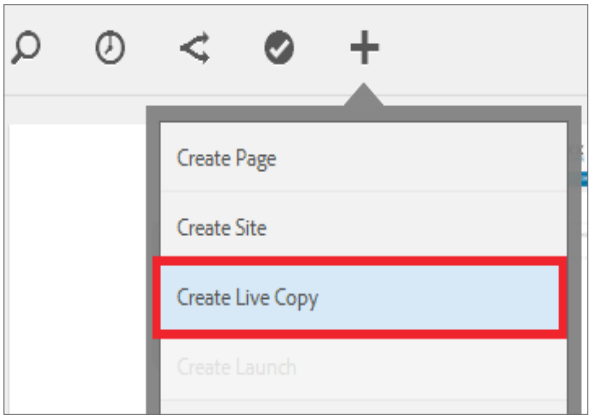
# Creation of Mobile website using MSM

The Geometrixx Outdoors Mobile  Site has a setup that enables to create a special live copy from the Geometrixx Outdoors Site (the standard, non-mobile one): when the live copy is created and when 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.

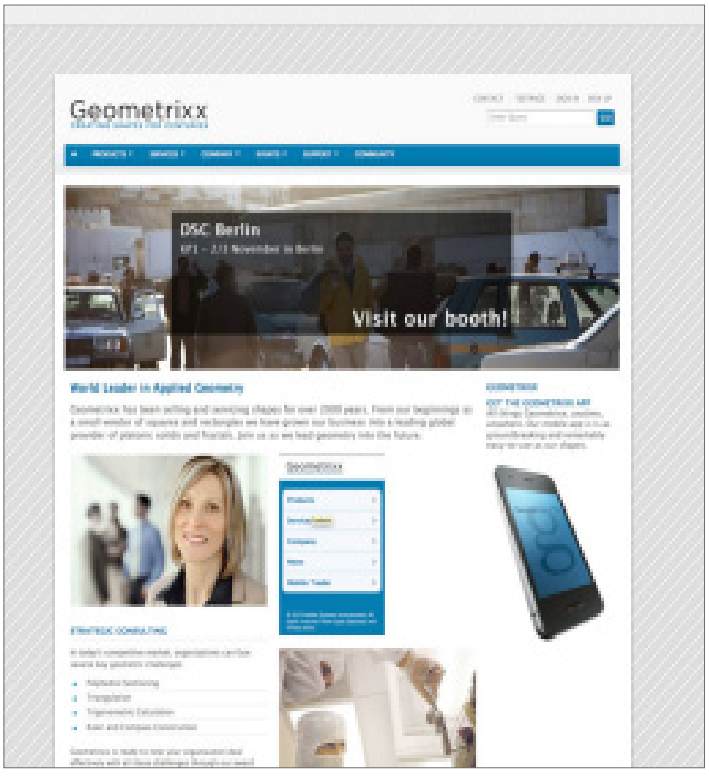
### EXERCISE 10.3 - Create a mobile website

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 (e.g. edition within an emulator) and can be managed in sync with the standard site.

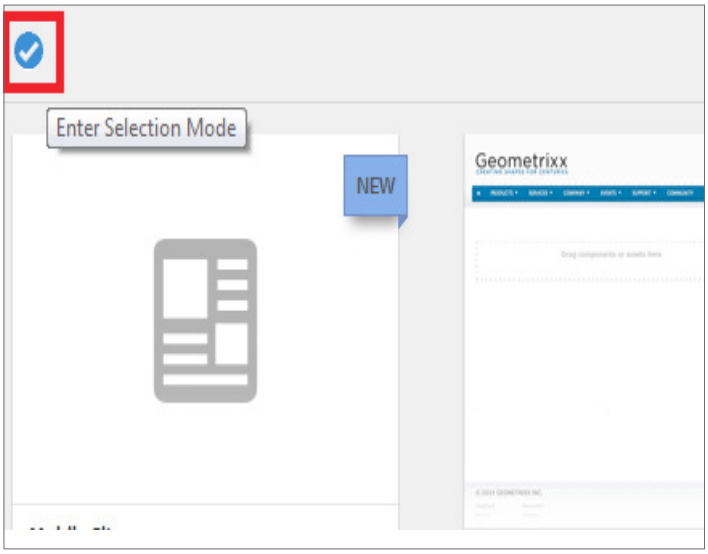
1. Log into Touch-Optimized UI. From the rail, select **Sites**.
   http://localhost:4502/
2. Click the + symbol and select **Create Page.**
3. Select the template you already created and click **Next**.
4. Provide a title (*mobile site*) and click **Create**.
5. Go to Site and click the + symbol. Select **Create Live Copy**.



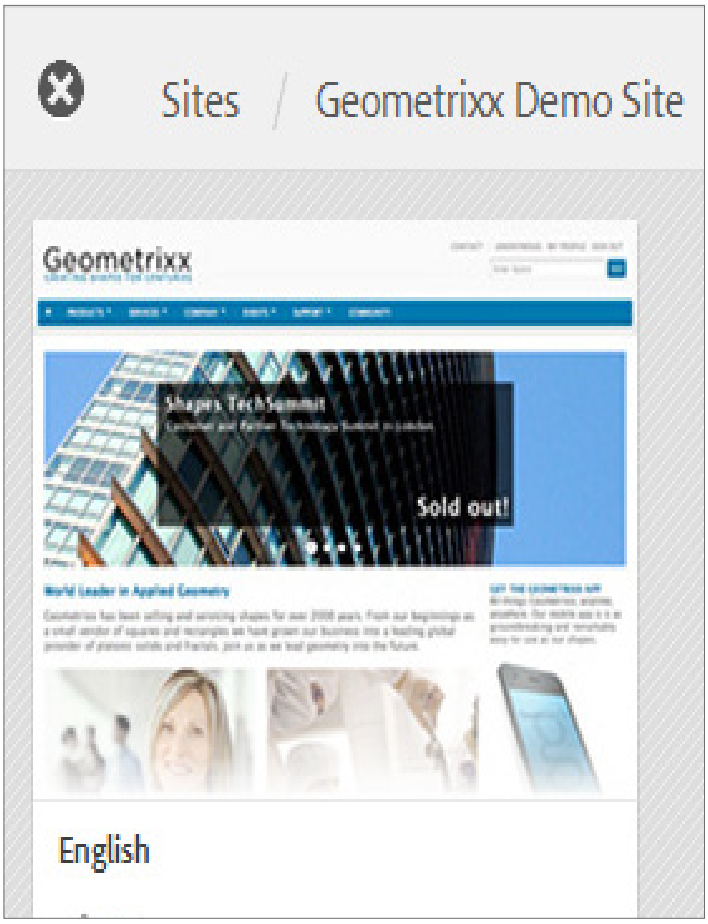

6. Double-click the Geometrixx Demo Site.



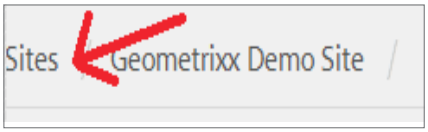
7. Select the tick symbol to enter into the Selection mode.



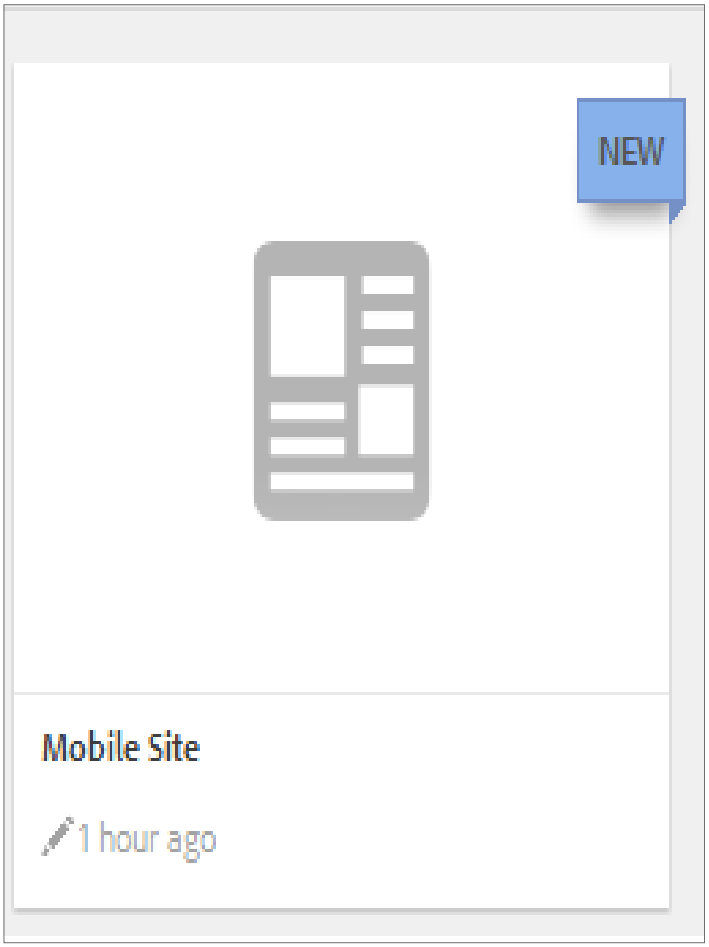

8. Select **English** and click **Next**.



9. From the top-left corner, select **Sites.**

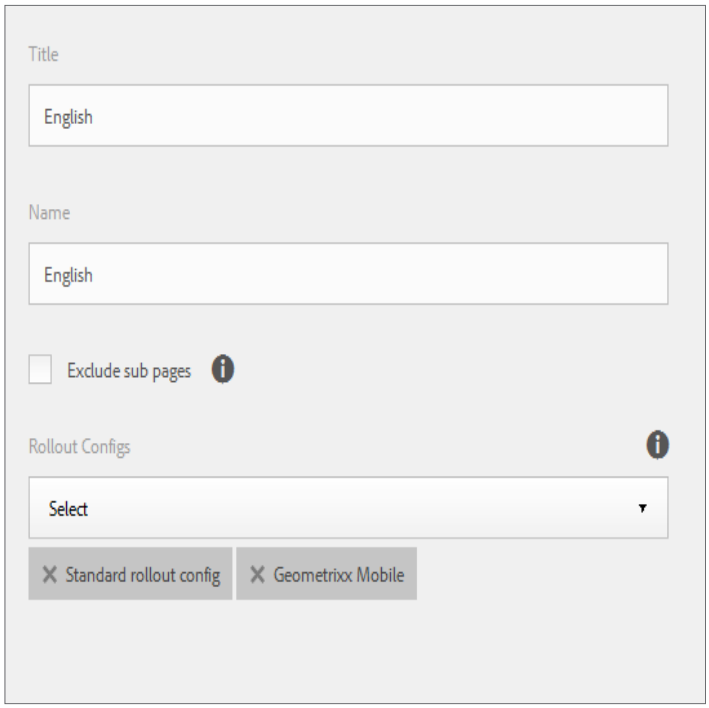



10. Select Mobile Site you created now and click **Next**.

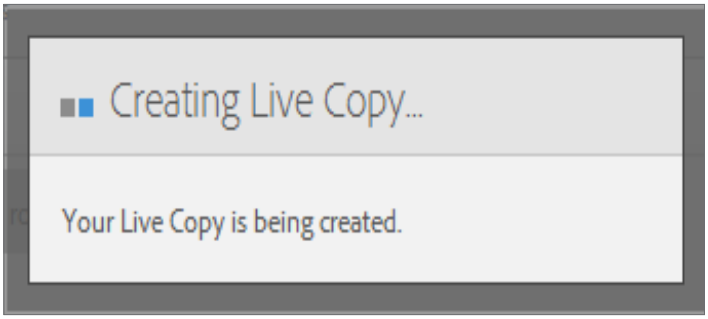



11. Enter the title as English.
12. From the drop-down list, select the following configurations:

› Standard rollout config
› Geometrixx Mobile

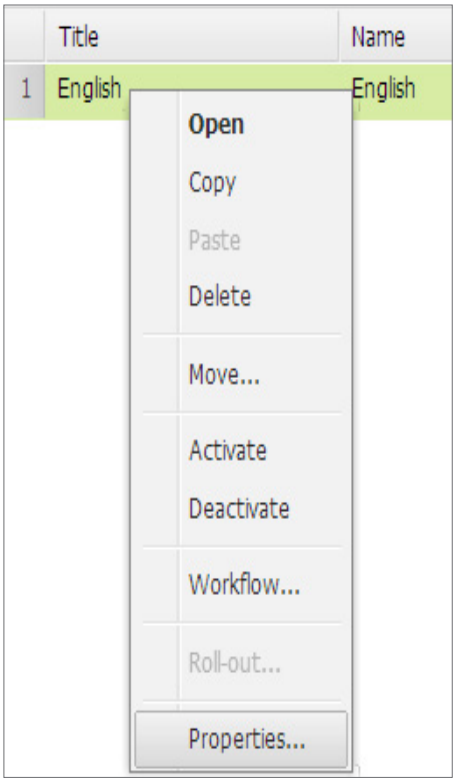



13. Click **Create** to create the Live Copy.

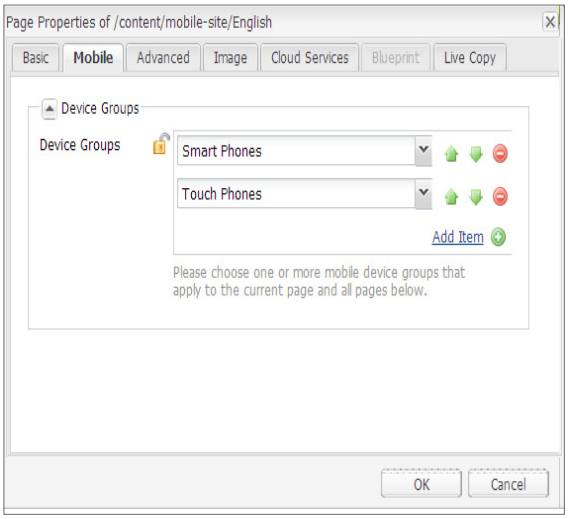



14. Wait for the Live Copy to be created.
15. After the Live Copy is created, go to the website console in the Classic UI.
16. Select Mobile Site.

17. Select and right-click English. Choose **Properties** from the context menu.




18. 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

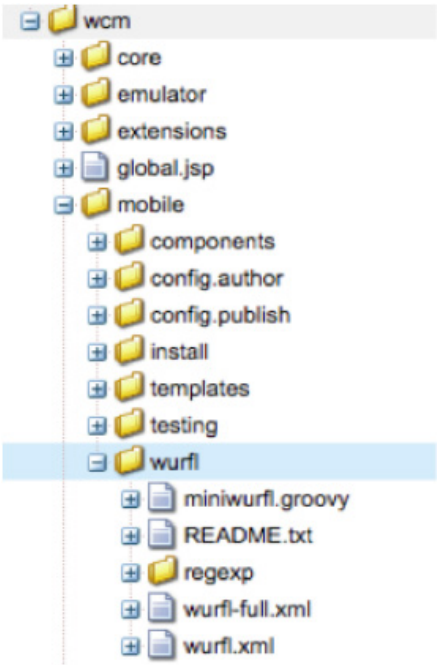
AEM provides several device emulators that allow you to see how your web content will appear on those mobile devices. The default emulators can be found in the following location in the repository:
*/libs/wcm/mobile/components/emulators*
Those emulators are grouped, based on sets of capabilities, e.g., "supports images" or "support 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/

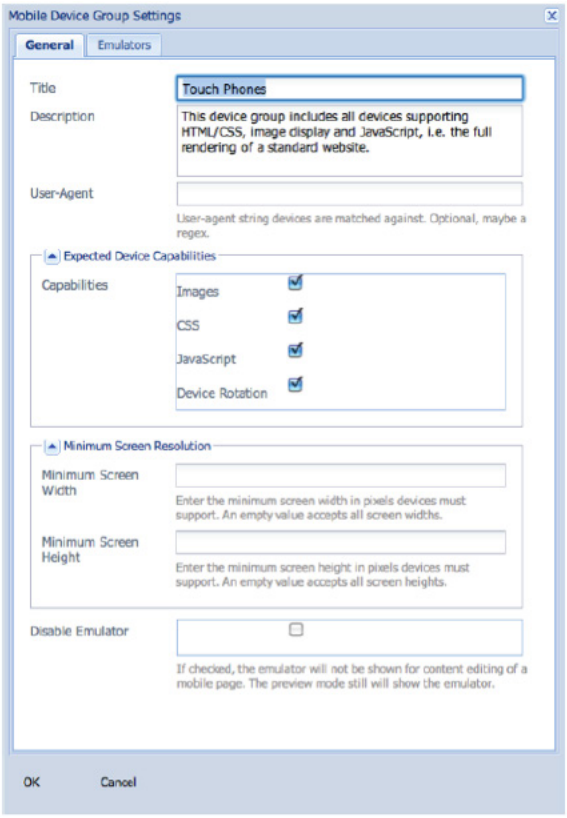
The AEM wurfl.xml file can be located at this URI:
*/libs/wcm/mobile/devicespecs/*




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

Emulator groups 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 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 User Interface (UI), e.g. 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 contents into emulator DIVs.
See the following example HTML code:

```
<body>
<div id=”wrapper” class=”page mobilecontentpage “>
    <div class=”topnav mobiletopnav”>
    ...
```

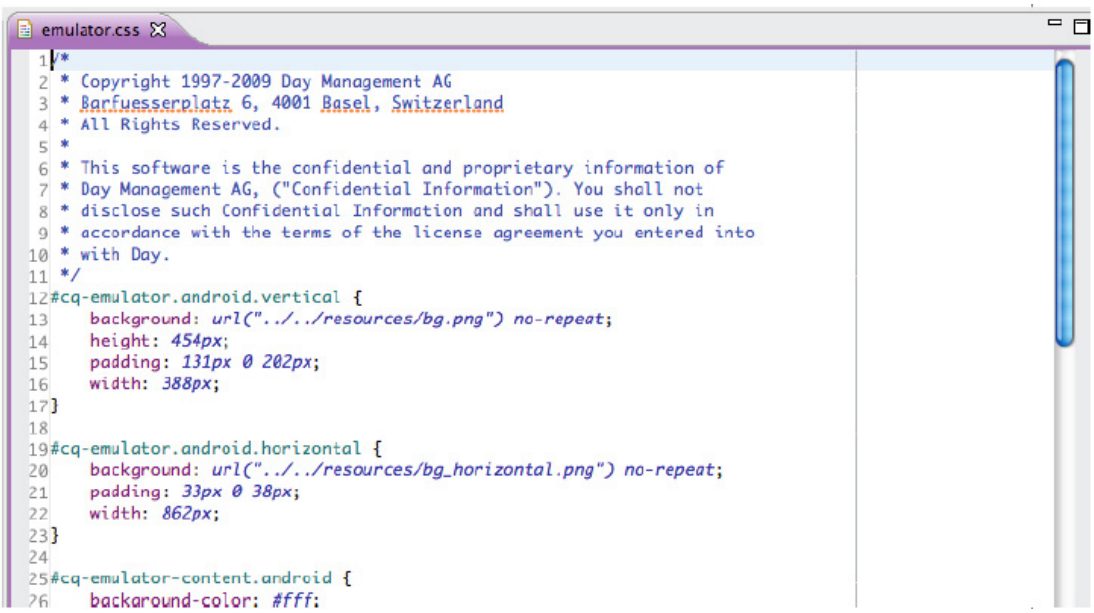
```
    </div>
    ...
</div>
</body>
```

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:

*/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 the 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.

# 11

# Complex Components Using JSP

So far each component that we have created has taught us one thing. As you know, that is not real life. The following section explores turning end user requirements into a development plan and then implementation. The requirements define a "complex" Component that manages both textual and binary (i.e. image) content, and has both a Dialog and Design Dialog. The requirements include allowing the Component to be used by the parsys Component, creating a Dialog 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? Since 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. Secondly, you will observe how a Day Solution Engineer 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 for 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 Web site structure “/content/trainingSite”.
3. Widget should have property regexText with an error message if regular expression fails.

SOLUTION ENGINEER’S TRANSLATION: A paragraph system Component that allows for the writing and displaying of three properties (2 paragraph properties, 1 design/style property), and has both a Dialog and Design Dialog.

The image:

1. Is a Dialog Widget, most likely an xtype of smartimage.
2. Must be configured in the Component’s cq:editConfig to allow for draggingand- dropping of images from the Content Finder.

The rich text:

1. Is a Dialog 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 Widget, most likely an xtype of pathcompletion.
2. Widget should have property regex with a regular expression validating the user’s input (e.g. “/^\\/content\\/training-site\\/(.)*$/”).
3. Widget should have property regexText with an error message if regular expression fails.

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

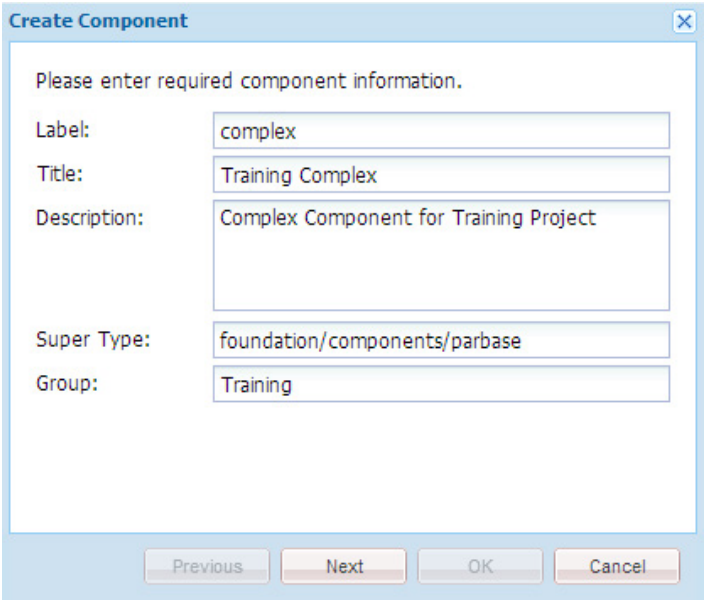
1. Create a Component, making sure to allow that any paragraph system Component can be this Component’s “parent.”
2. Edit the default JSP so it displays the content/properties in an appropriate manner, even without any written content.
3. Create a Dialog for the Component.
4. Create a Design Dialog for 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.
8. Add an image Widget to the Dialog.
9. Add a path completion Widget to the Design Dialog.

10. Perform necessary Component configurations.

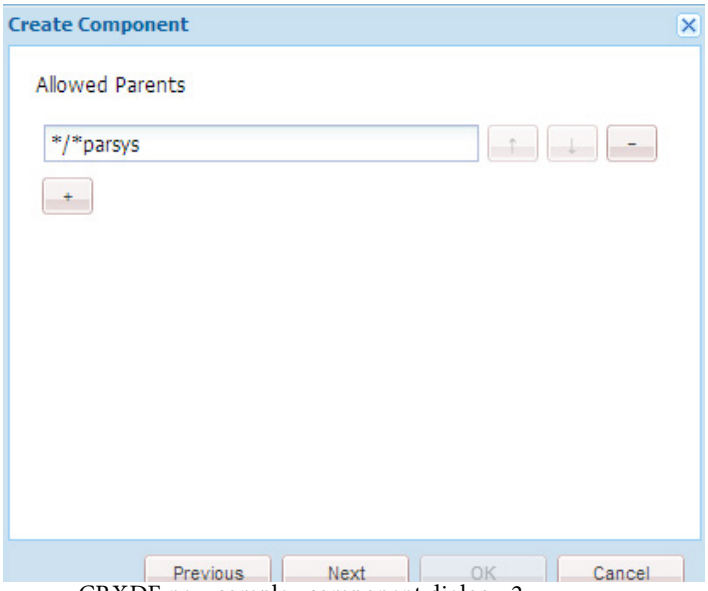
11. Test this Component by writing content using the newly created Dialog and Design Dialog.

## EXERCISE 11-1 Create a complex component

1. Create a new complex "content" Component.




CRXDE new complex component dialog - 1




CRXDE new complex component dialog - 2

2. Open the file **complex.jsp**, and enter some HTML and JSP code, similar to below – then **Save**.

complex.jsp

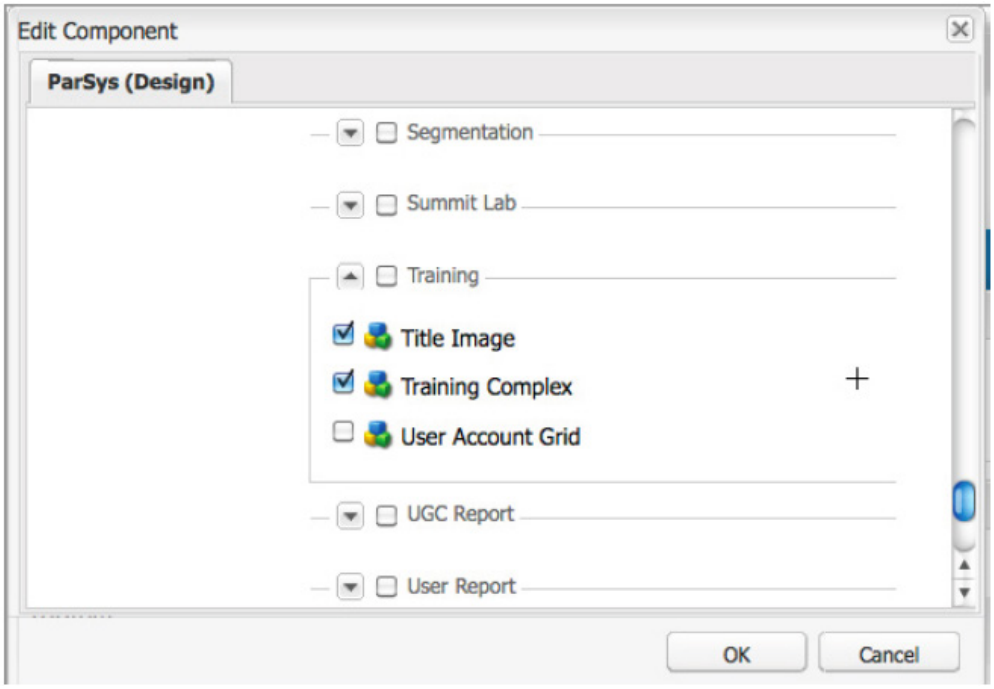
```
<%@include file="/libs/foundation/global.jsp"%>
<%@ page import="com.day.cq.wcm.foundation.Image" %>
<%
// getting the image from the Dialog/resouce
// notice the use of the second parameter "image" -- this signifies
that the
// image will live on a resource (called image) below the requested
resource
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);
%>
```

---

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

---

3. Create a Dialog and Design Dialog for the complex Component.
   - › You will worry about Widgets and configurations later - the focus now is to see your new Component in action.
4. Add your complex Component to the paragraph system Component in Design mode.




Design dialog of the paragraph system

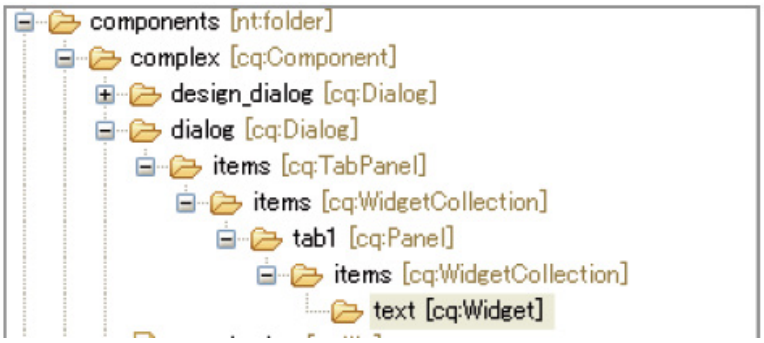
5. Test your script by adding an instance of this Component (i.e. paragraph) to the paragraph system Component of a Training Page.
   - › If successful, you should see the default complex Component output, similar to the image below.

Page preview of complex component - no content

**NOTE**: Notice how your complex Component is listed under the Training group. This is because you declared such during the creation of your Component.

6. Create an **items** node (nodeType cq:WidgetCollection) under the *tab1* node of your Component's Dialog.

7. Create a **text** node (nodeType cq:Widget) under the newly created *items* node.



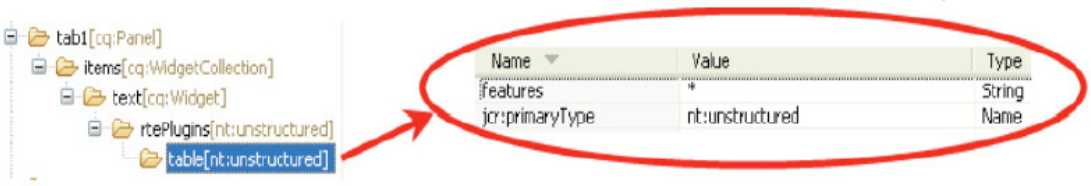
8. Assign the following properties to the newly created text node:

- The property that will define where content is stored
  - Name = name
  - Type = String
  - Value = ./text
- The property that will define the Widget type
  - Name = xtype
  - Type = String
  - Value = richtext
- The property that will define to hide the label of the Widget
  - Name = hideLabel
  - Type = Boolean
  - Value = true
- The property that will define the the default value of the Widget
  - Name = defaultValue
  - Type = String
  - Value = This is some text.

9. Create an **rtePlugins** node (nodeType nt:unstructured) under the newlycreated *text* node.

10. Create a **table** node (nodeType nt:unstructured) under the newly created *rtePlugins* node.

11. Assign the features property (Type = String, Value = *) to the newly created *table* node.




CRXDE rtePlugin table plugin structure and property

12. Copy the nodes tab2 and tab3 under the node /libs/foundation/components/textimage/dialog/items - then paste to the complex Component's Dialog so that they are a peer of tab1 (e.g. /apps/training/components/complex/dialog/ items/items).
  - tab2 = the smartimage tab
  - 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 have just copied to better understand the internal workings of AEM. If you examine closely enough, you will see the Widget is actually storing image related content at a level deeper than current resource (e.g. ./image/file, ./image/fileReference, etc.). This ties nicely with your previously written code (Image image = new Image(resource, "image");).

1. Now we build out the Design Dialog. Create an **items** node (nodeTypecq:WidgetCollection) under the tab1 node of your Component's Design Dialog.

2. Create a **path** node (nodeType cq:Widget) under the newly created *items* node.

3. Assign the following properties to the newly created *path* node:

- The property that will define where content is stored
  - Name = name
  - Type = String
  - Value = ./path
- The property that will define the Widget type
  - Name = xtype
  - Type = String
  - Value = pathfield
- The property that will define the label applied to the Widget
  - Name = fieldLabel
  - Type = String
  - Value = Enter a Path
- The property that will define the root path to display
  - Name = rootPath
  - Type = String
  - Value = /content/training-site
- The property that will define the regular expression used to evaluate user input
  - Name = regex
  - Type = String
  - Value = /^\/content\/training-site\/(.)*$/
- The property that will define the error message if a user's input fails the regular expression
  - Name = regexText
  - Type = String
  - Value = Please insert a Page that "lives" under /content/trainingSite

4. Copy the node /libs/foundation/components/textimage/cq:editConfig - then paste to the root node of your complex Component (e.g. /apps/training/components/complex).

- Enables drag-and-drop capabilities from the Content Finder In order 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.

5. Using CRXDE, navigate to:

   <path-to-complexcomponent>/cq:editConfig/cq:dropTargets/image

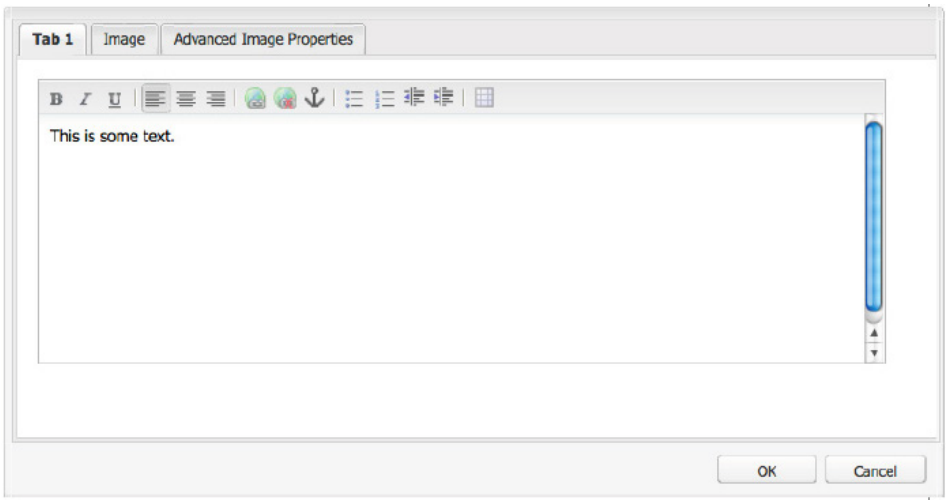
   Validate that the image node has the following properties:

   - Accept (Type = String) - the media types to be accepted (e.g. image/.*, etc.)
   - Groups (Type = String) - the groups in the Content Finder assets can be accepted from (e.g. media)
   - PropertyName (Type = String) - the property the reference should be stored (e.g. ./image/fileReference)

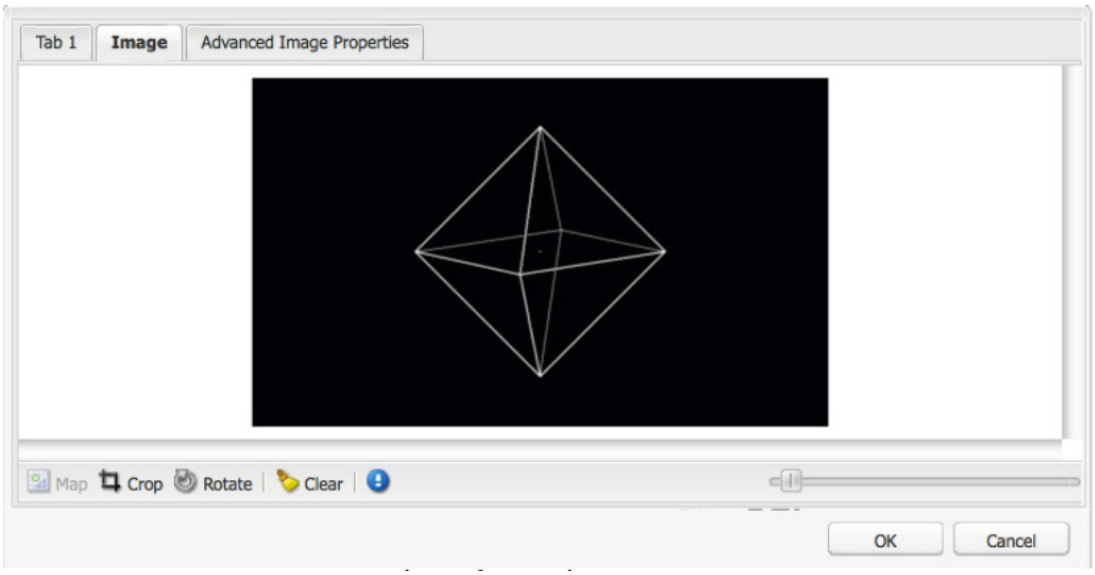
6. Navigate to the ***parameters*** node that is a child to the ***image*** node. Change the value of the *sling:resourceType* property to have the value: *training/components/complex*. Remember that the *sling:resourceType* property should reference the location to find a rendering script.

7. Test your script/Dialog by requesting a Page in AEM Siteadmin that implements this Training complex Component - then interact with the Dialog and Design Dialog.

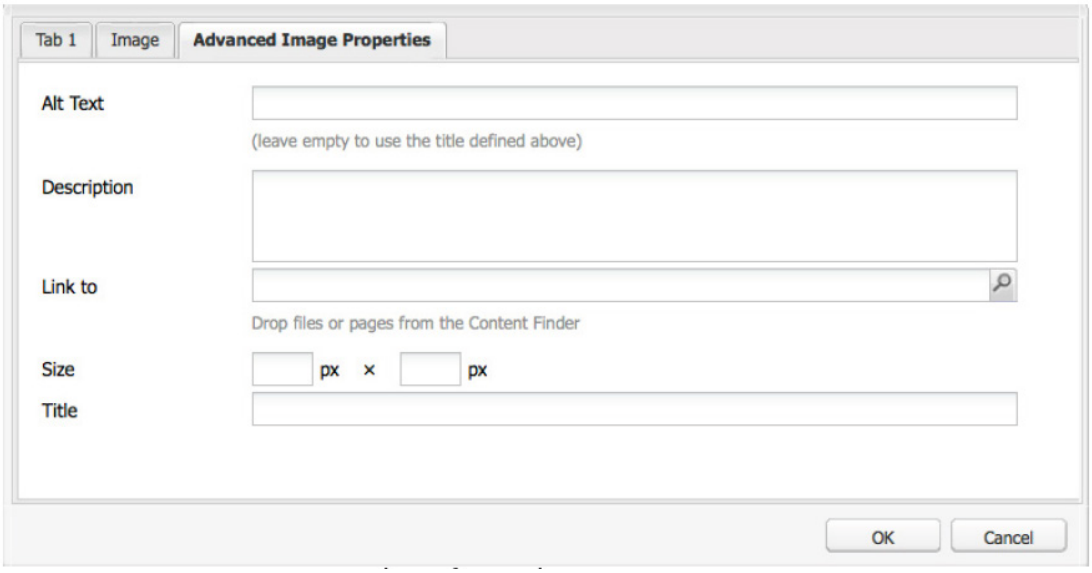
- If successful, you should see Page output, a Dialog and Design Dialog similar to the images below.



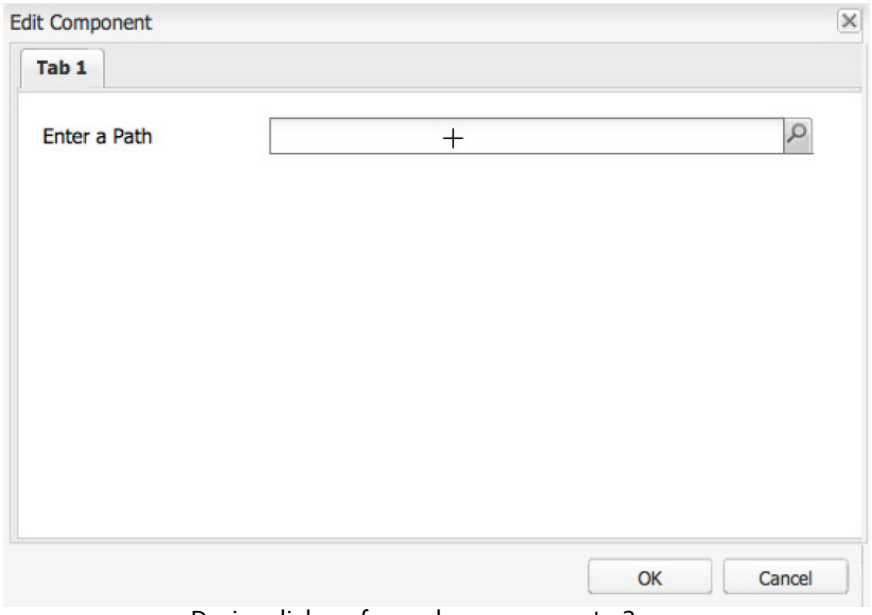

Dialog of complex component - 1



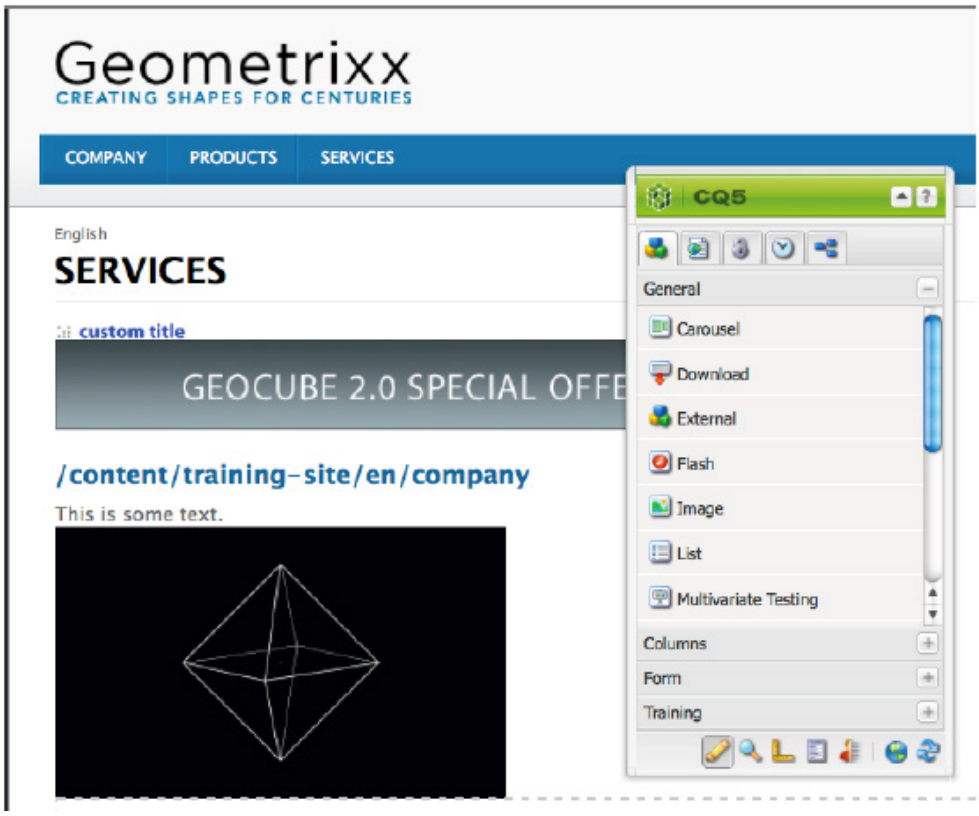

Dialog of complex component - 2




Dialog of complex component - 3




Design dialog of complex component - 3




Page preview of complex component - with content

***Congratulations!*** You have successfully created a complex Component that contains a Dialog, Design Dialog, default values, custom configuration, and validation of user input, in addition to enabling drag-and-drop functionality from the Content Finder. Give yourself a pat on the back - this was quite a challenge.

## End User Search

Searching for content in AEM is very similar to "traditional" searches you have created in the past.

1. An HTML form is needed to collect the user's input (search string).
2. Once 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 Java Content Repository (JCR) using the JCR API, some basic functionality (API calls) need to be implemented:

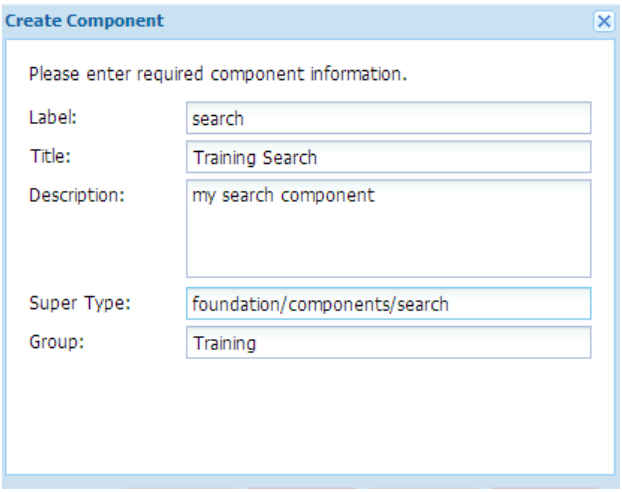
- javax.jcr.Session - JCR session, can be reached for example by Node. getSession()
- javax.jcr.Workspace - JCR workspace, can be reached for example by Session. getWorkspace()
- javax.jcr.query.QueryManager - QueryManager is used to create a Query object and can be reached via Workspace.getQueryManager()
- createQuery(String statement, String language) - creates the Query object for the provided statement in the provided language (e.g. 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

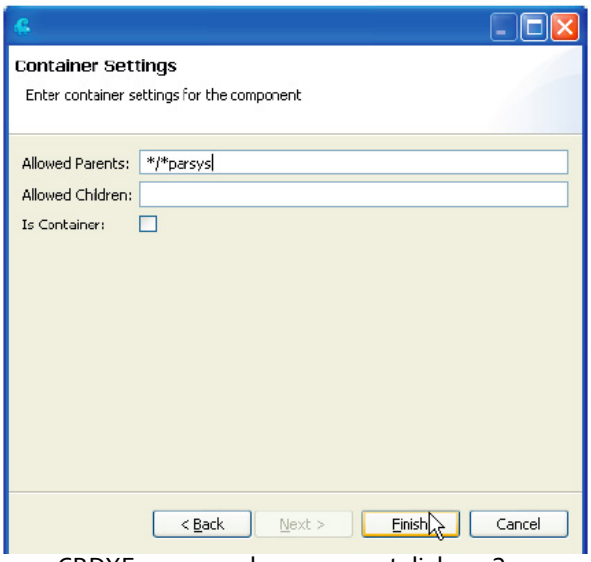
For more detailed information, please review the Javadocs for the JCR and CRX provided in the documentation.

### 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 Web site/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 Web site based on a query string provided in the request.

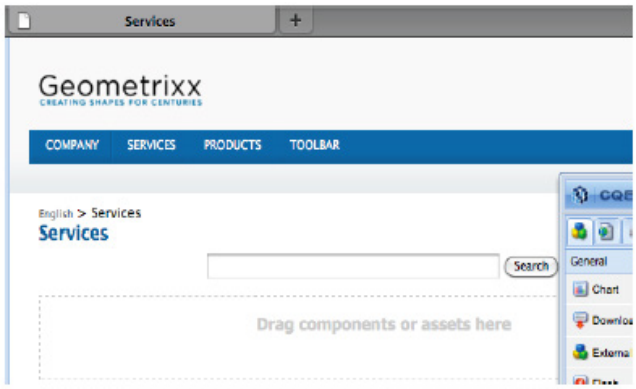
1. Create a new Search component.






CRDXE new search component dialog - 2

2. Replace *search.jsp* with the *search.jsp* from the USB drive.
3. Update search.jsp to ensure that the search path matches the path to your training site.
4. Add your search Component to the paragraph system Component in Design mode.
   - › Notice how the Component will be found in the "Training" group - this was defined during the creation of the Component
5. Test your script by adding an instance of this Component (i.e. paragraph) to the paragraph system Component of a Training Page.
   - › If successful, you should see the default complex Component output, similar to the image below.




Page preview of search component

6. Search for a word you know exists on a separate Page in your Training Web site structure.
   - › You may need to perform this search in Page preview mode, to ensure a "clean" request
7. Now replace the contents of *search.jsp* with the contents of *search.aemwcm.jsp*.
8. Examine the code to see the differences between the JCR search API and the WCM search APIs.

9. Try the search again.
10. Now replace the contents of *search.jsp* with the contents of *searchenhanced.aemwcm.jsp.*
11. 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 and the use of facets. The code also builds a search term tag cloud - called **"**Search Trends**"** .
12. Try the search again.
13. **(Optional steps: Follow these steps to index a custom property for faster search results)** Locate the Oak:index node in CRXDE Lite.
14. Create a node, trainingIndexNodes, of type "Oak:QueryIndexDefinition".

Add the following properties to the node:

Name = propertyNames
Type = Name[]
Value = custom-title

Name = reindex
Type = Boolean
Value = false

Name = type
Type = String
Value = proprty

| | | | |
|---|---|---|---|
| 1 | jcr:primaryType | Name | oak:QueryIndexDefinition |
| 2 | propertyNames | Name[] | custom-title |
| 3 | reindex | Boolean | false |
| 4 | type | String | property |

***Congratulations!*** You have successfully created a search Component that queries the content in your Training Web site structure. You can further enhance this Component by adding Widgets to the Dialog 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 its 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, ASP, etc.

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.

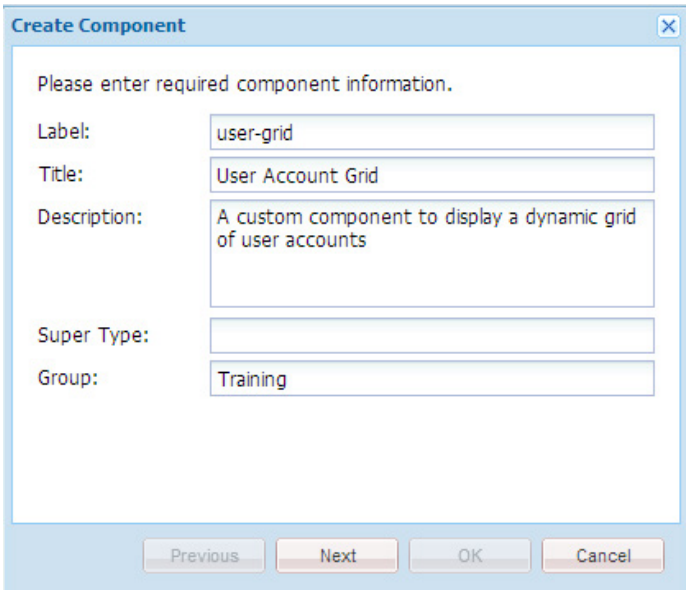
### EXERCISE 11-3 - Dynamic User Account Grid

Goal: 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

**NOTE:** The default Sling servlet in AEM can be used to retrieve content in JSON format for pretty much 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.

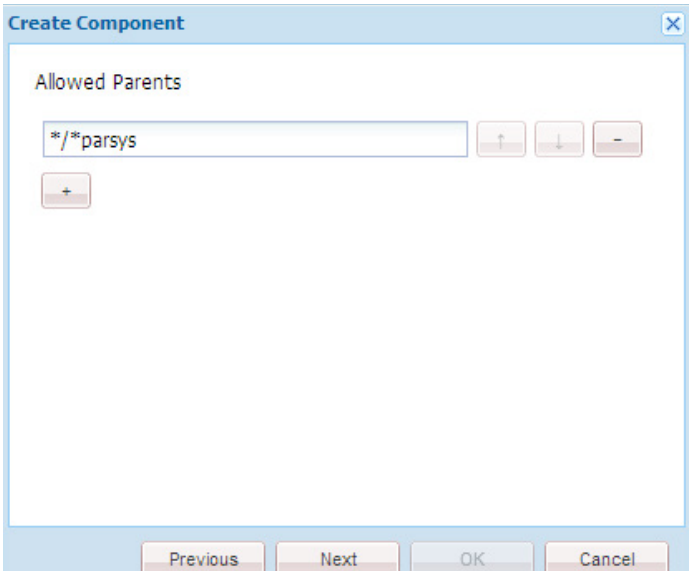
## Create the component

1. Right click the /apps/training/components folder and add the user-grid component.
   - Name = user-grid
   - Title = User Account Grid
   - Description = A custom component to display a dynamic grid of user accounts
   - Group = Training




2. Click **Next** and enter the following value:

› AllowedParents = */*parsys




3. Click **Finish**.

4. Add the following code to your user-grid component:

```
<%@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>
```

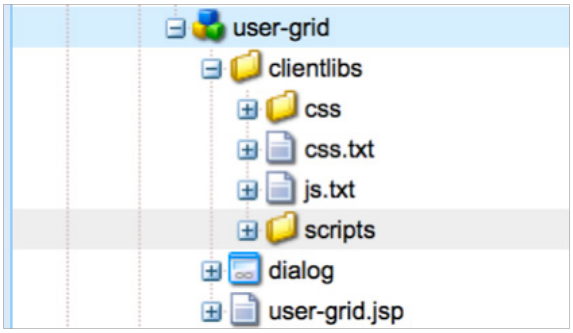
```
                <td>This is data 4</td>
            </tr>
            <tr>
                <td>This is data 1</td>
                <td>This is data 2</td>
                <td>This is data 3</td>
                <td>This is data 4</td>
            </tr>
            <tr>
                <td>This is 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>
```

NOTE: This is just placeholder code which we will make dynamic in a following section of this exercise.

5. Right click on the user-grid component and Select New.. Node.
   - Name = clientlibs
   - Type = cq:ClientLibraryFolder.
6. Add the following properties to the new client library folder:
   - Name = dependencies
   - Type = String[]
   - Value = cq.jquery
   - Name = categories

- Type = String[]
- Value = componentlab

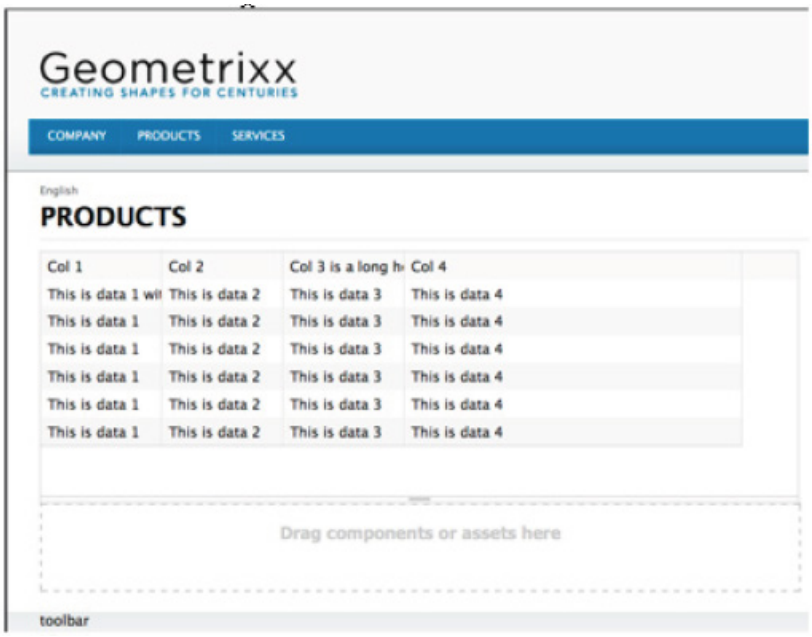
7. Install the user-grid-libs-1.0.zip from the respective folder in USB. (For more information on how to install a package, see the Package Manager section in the next chapter.)
   This package installs the assets you need to develop the User Grid component in a Custom folder of the Training project.
8. Refresh the Training project folder.
9. Copy the content of the custom folder to the clientlibs folder.
   The directory structure looks as shown below now:




10. Examine flexigrid.js and flexigrid.css files, so you understand what they do. (They are also provided in the Asset folder in USB.)
11. 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>
...
```

12. Using the procedures introduced previously, add your User Account Grid component to the Paragraph System design.
13. Test your component. It should look like the following:




**Make the component Interactive**

14. 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 () {}
```

15. 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 above 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.

---

16. 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.");
}
```

17. Delete the hardcoded text between the <table class="user-table"> and </ table> tags.

## Create the callback jsp file

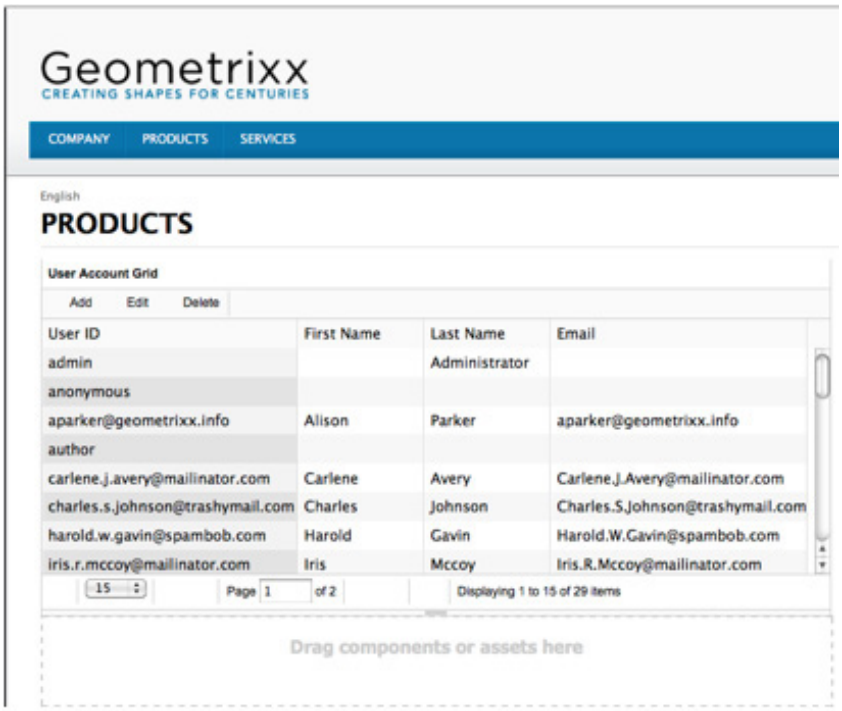
1. Create the user-grid.json.POST.jsp callback script. Right click on the user-grid component node and select New..JSP.
2. Enter the following code into your new user-grid.json.POST.jsp script:

```
<%@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();
```

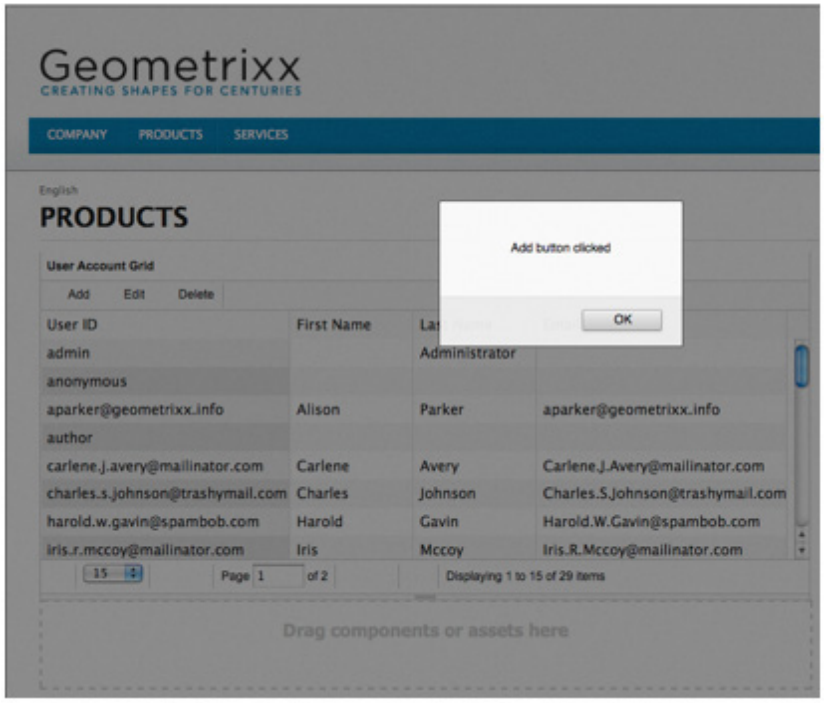
```
            users.add(tmpUser);
}

//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.

4. Notice the the interactive grid now appearing on the page.




5. Click on the action buttons (Add, Edit and Delete) to see the appropriate message box appear.
6. If needed, install the complete package from USB > 9-3 user-grid-component/crx-package/user-grid-step2.




NOTE: Not all of the grid features are functional. The search and pagination functions have not yet been implemented. This is left as an extra credit exercise.

# 12

# OSGi Bundles & Workflow

In this chapter, you will learn the following:

- OSGi bundles
- Basics of the Workflow Console
- Implementing 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 (are 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:

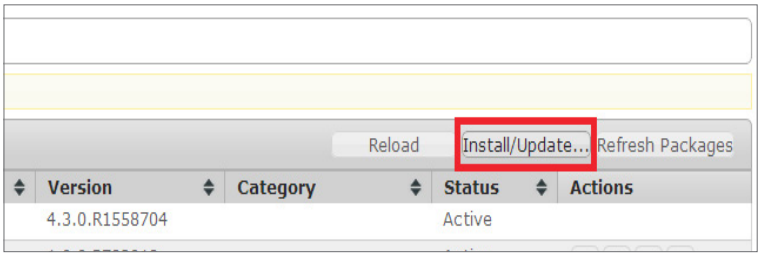
- install
- start
- stop
- update
- uninstall
- see the current status
- access more detailed information (e.g. symbolic name, version, location, etc.) about the specific bundles

## EXERCISE 12.1 - Consume an OSGi bundle

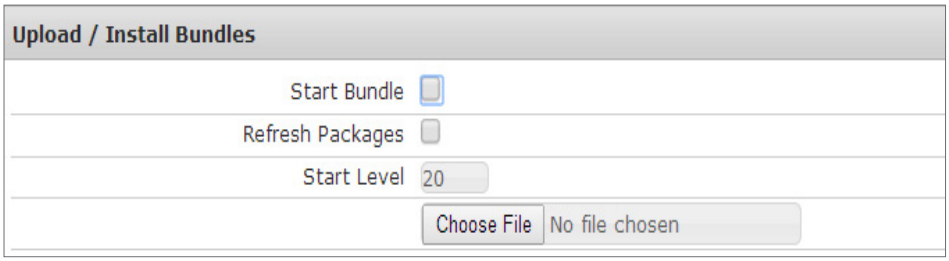
Creation of a bundle is beyond the scope of the course. So, you will use the bundle that is provided in the exercise folder.
Following is the Java class that is used for creating the bundle:

```
package org.training.test;
public class HelloWorld {
public String getMessage() {
    return “Hello World !!!”;
   }
}
```

1. Navigate to system console.
   *http://localhost:4502/system/console*
2. From the top-right corner, select **Install/Update**.



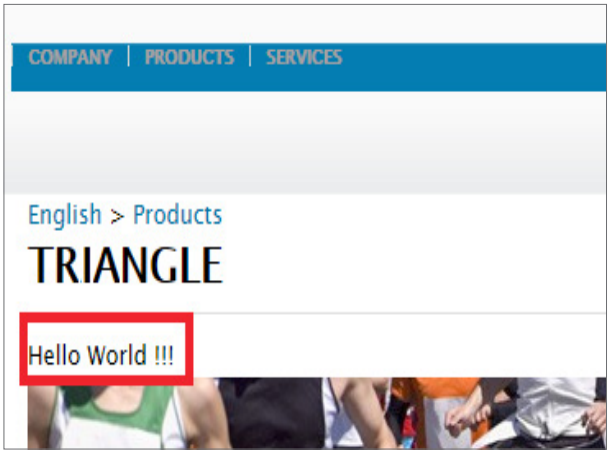
3. Select the Start Bundle check box and click **Choose File**.




4. Select the bundle from the exercise folder and click **Install or Update**.
5. In the browser, search for Company Portal. (Press CTRL+F or Command+F)
   Note that the bundle is installed.
6. Navigate to the Components tab (**Main** > **Components**). In the browser, search for *HelloWorld* component.
   Note that the component is installed.
7. In CRXDE Lite, open the default script of the Title component you developed. (*components/training-title/training-title.html*)
8. 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>
```

9. 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 in tight conjunction 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

Model is made of 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

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

References the resource that has to be advanced through a workflow. The payload implementation references a resource in the repository (by either 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 in conjunction with sling—productive; for example, the referenced node could be rendered as a form.

### Lifecycle

Lifecycle 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 either to a specifc 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 via the Workflows button on the Welcome page or through the Workflows button in 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
- SiteAdmin Console (Websites tab)
- Right Context Menu
- Workflow item in Toolbar
- Sidekick

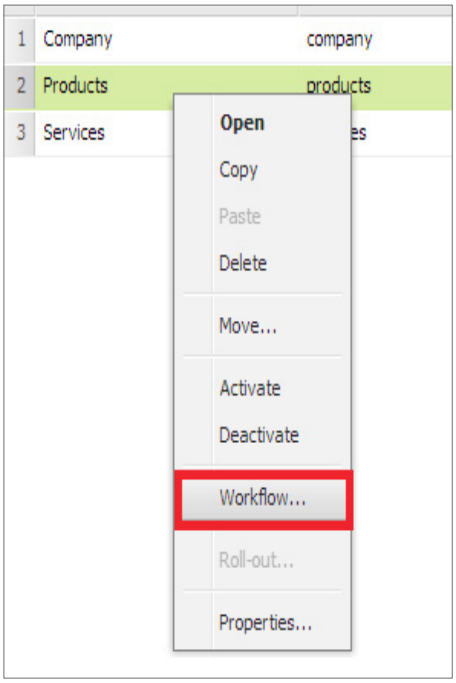
NOTE: The payload is assigned to the current version of the workflow; if the main copy of the workflow is updated later, the changes will have no impact on the currently running instance.

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. Both AEM's native functionality and the functionality of any website built with AEM are delivered through this framework.

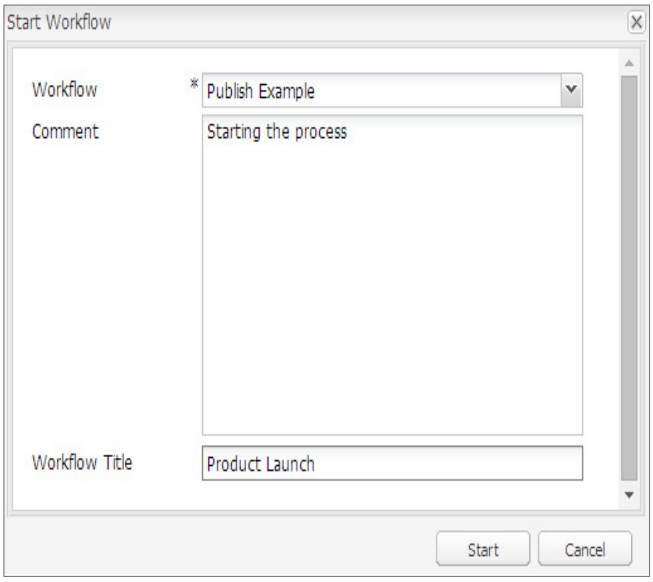
### EXERCISE 12.2 - Explore basic workflow

You will be starting the workflow from the SiteAdmin console.

1. Open the SiteAdmin console, either by clicking on the Websites button from the Welcome page or by clicking on the **Globe** icon at the bottom of your sidekick.
2. Right-click on one of your pages to open the context menu. Select **Workflow.**

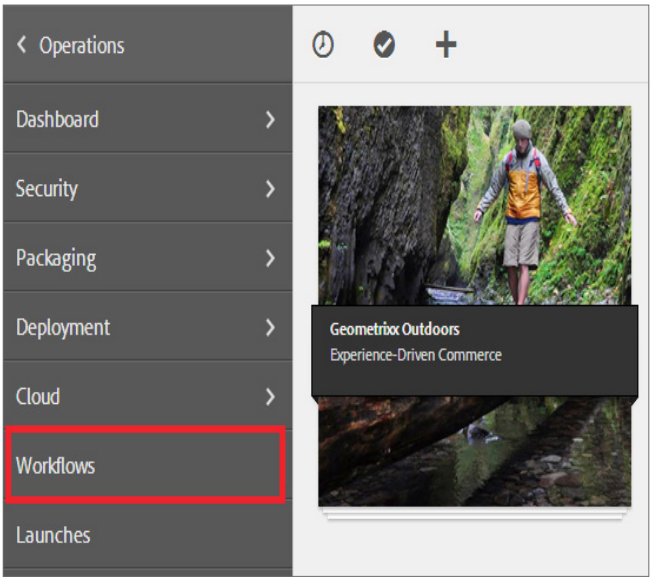


3. Select the **Publish Example** workflow, fill in the desired optional information, and click **Start.**



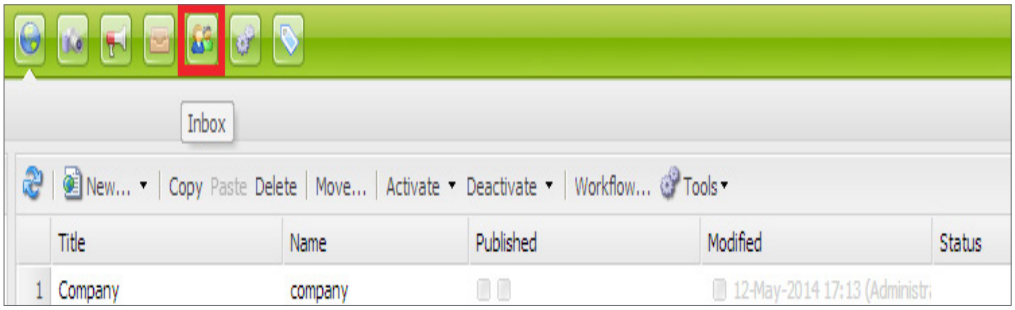

4. You will notice that the workflow icon shows up in the SiteAdmin console next to the page you chose.


5. Change to the Workflow Console by returning to the Welcome Screen and selecting Operations > Workflows.

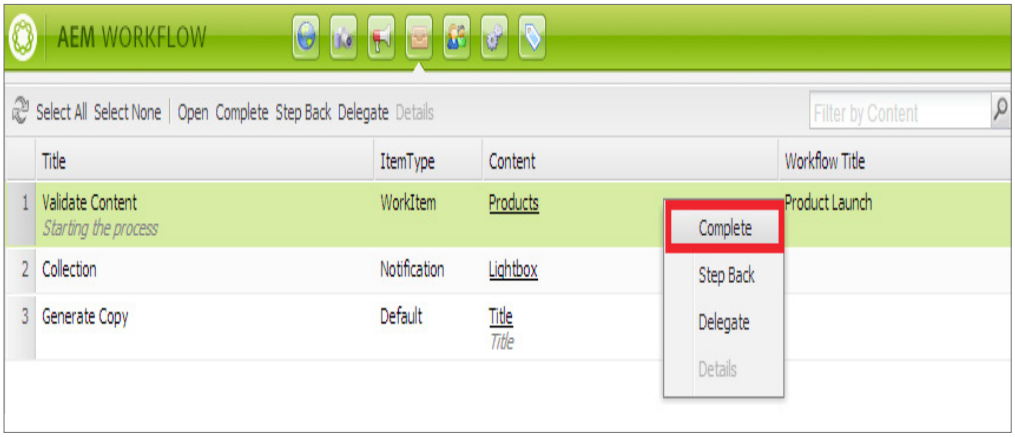


6. Search for **Publish Example**. After locating Publish Example, double-click it.
   The **Publish Example** workflow appears.
   You will notice that the **Publish Example** workflow has two steps—**Validate Content** and **Publish Content**. The **Validate Content** step is a participant step assigned 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 on the step to open the dialog box.

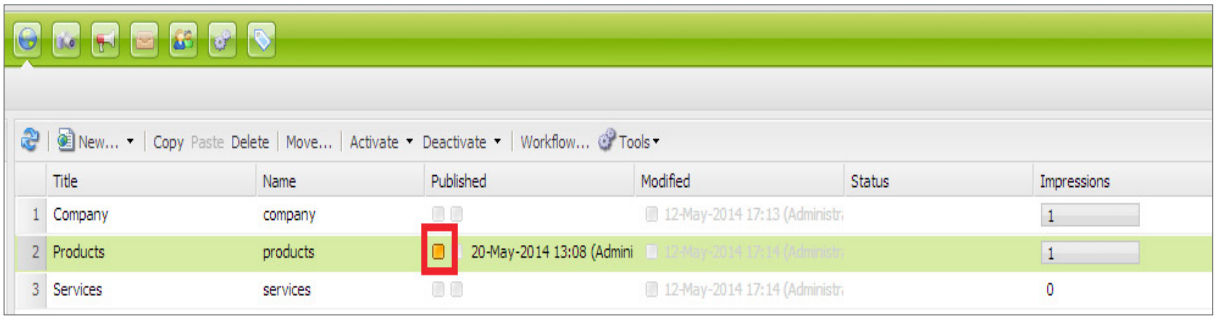

7. Switch to the **Inbox** by changing to either the Welcome Screen or the SiteAdmin Screen and clicking on the **Inbox** tab to change context. The Inbox will show the WorkItem entry for the page you put into workflow.



8. Select the WorkItem Validate Content and click Complete. Notice that the WorkItem disappears from the Inbox.




9. You will also note 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 a AEM Publish instance running).



# Create a Workflow implementation step

A workflow is made of steps. The steps that define a workflow are either participant steps or process steps. Participant steps require manual intervention by a person to advance the workflow. Process steps, on the other hand, 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 administrators 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 either an ECMA script or a service (a Java class in a budle). 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 a Java

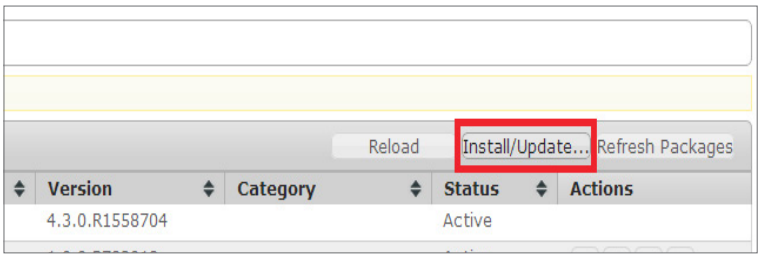
In this exercise, you will install a bundle that create a process step. Creation of a bundle is beyond the scope of the course. You will use the bundle that is provided in the exercise folder.
Following is the Java class that is used to create the bundle.

```
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”;
```

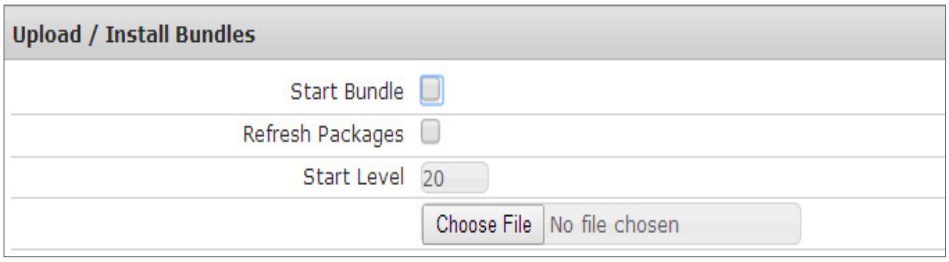
```
    public void execute(WorkItem item, WorkflowSession session, MetaDataMap
args) throws WorkflowException {
        WorkflowData workflowData = item.getWorkflowData();
        if (workflowData.getPayloadType().equals(TYPE_JCR_PATH)) {
          String path = workflowData.getPayload().toString() + "/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 system console.
   *http://localhost:4502/system/console*

2. From the top-right corner, select **Install/Update**.




3. Select the Start Bundle check box and click **Choose File**.

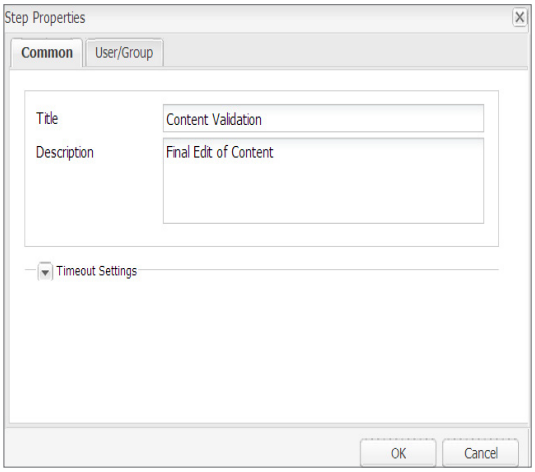


4. Select the bundle from the exercise folder and click **Install or Update**.
5. In the browser, search for Company Portal, and select the Company Portal bundle that is already installed.
6. Navigate to the Components tab (**Main** > **Components**). In the browser search for *My* Process. Note that the component, My Process, is installed.

## EXERCISE 12.4 - Implement a process step

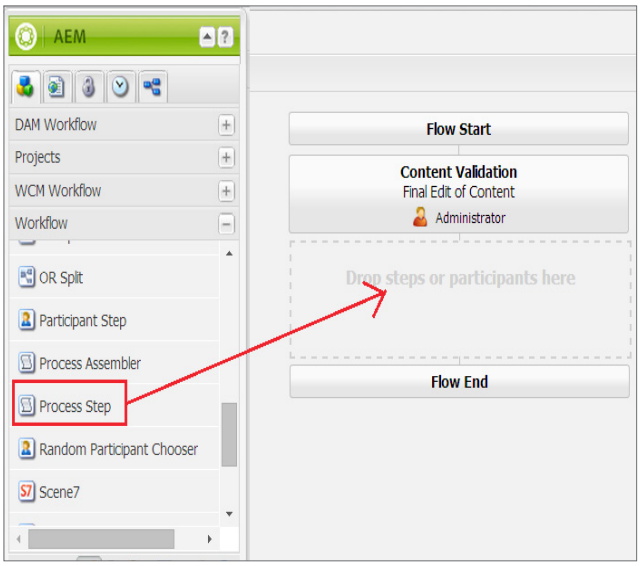
In this exercise, you will use the process step you created in the previous exercise in a workflow.

1. Navigate to the Workflow console.
   http://localhost:4502/libs/cq/workflow/content/console.html
2. In the Models tab, select **New**. Enter the title as **Approval**.
3. After workflow is created, double-click the workflow.
   The new workflow model has a Start and End steps.
4. Double-click Step 1. Add the following details:
   Title: Content Validation
   Description: Final edit of content




5. Go to the User/Group tab and select User/Group as admin.

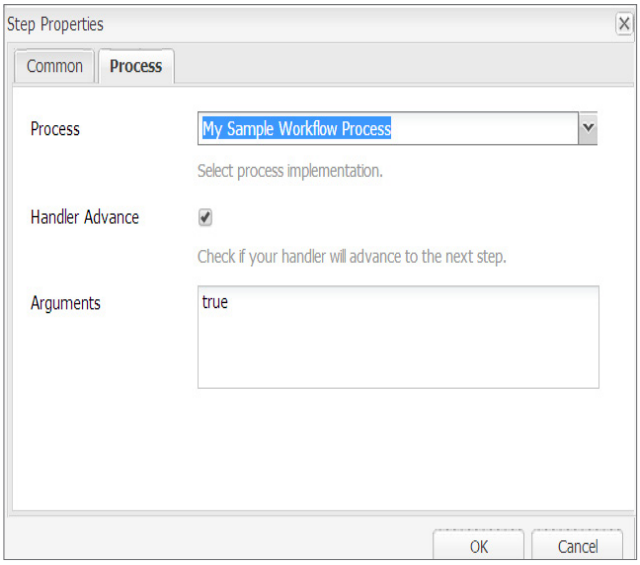
6. Add a process step by dragging and dropping a process step from the sidekick.



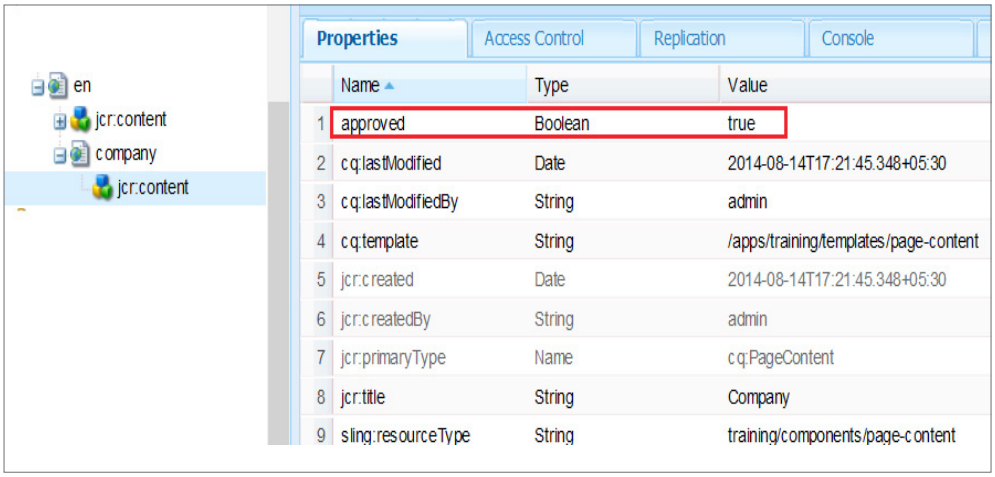

7. Double-click the process step.

8. Enter title as Final Approval.
   Description: Set the approved property

9. Enter the following details in the Process tab.
   Process: My Sample Workflow Process
   Arguments: true



10. Select the Handler Advance check box and select **OK**.
11. Ensure that you saved the workflow by clicking the **Save** button on the top-left corner.
12. Go to SiteAdmin and select any of the pages created.
    *http://localhost:4502/steadmin/English/*
13. Click **Workflow,** select Approval in the Workflow drop-down list, and select Start.
14. Click the Inbox icon at the top.
15. Select the workflow you created. Right-click and select **Complete**.
16. Go to CRXDE Lite and navigate to the page.
17. Note that the Approved property is added to the page.



# 13

# AEM Environment

In this chapter, you will learn the following:

- Package Manager
- AEM Deployment

## Package Manager

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

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 for informational purposes 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

There are multiple ways to manage content packages:

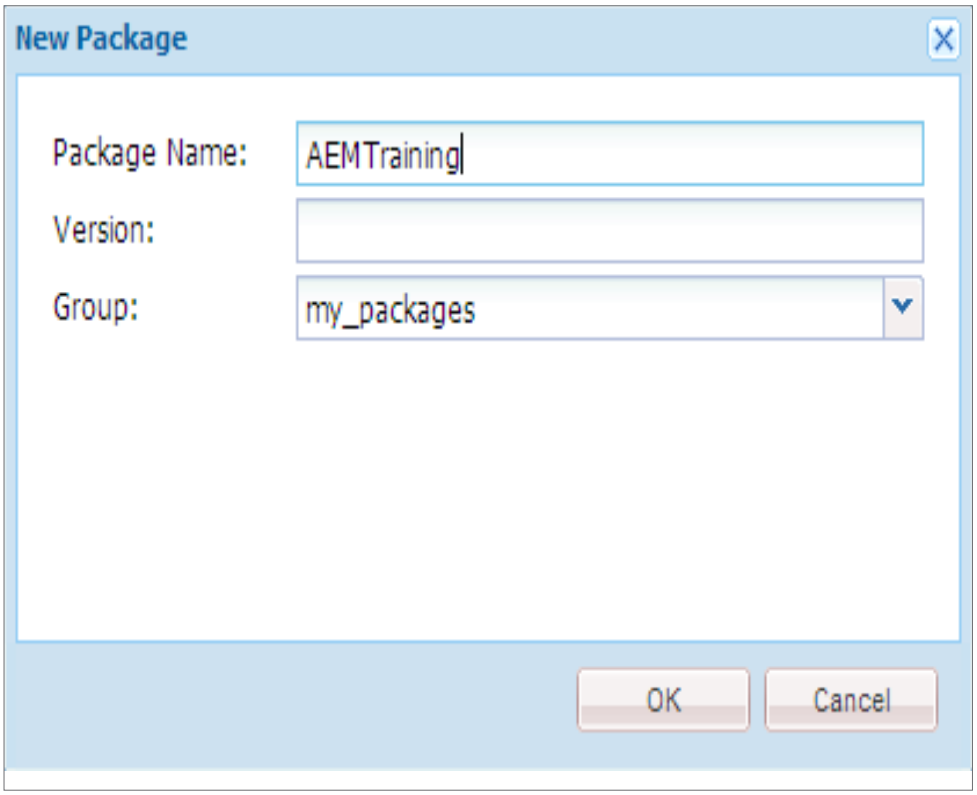
- CRX Package Manager—using the direct CRX interface
- AEM Package Manager
- cURL actions—command line option that allows package actions to be automated

In the following exercise, you will be using the CRX Package Manager. You should take some time to explore the documentation for the AEM and cURL methods of managing packages.

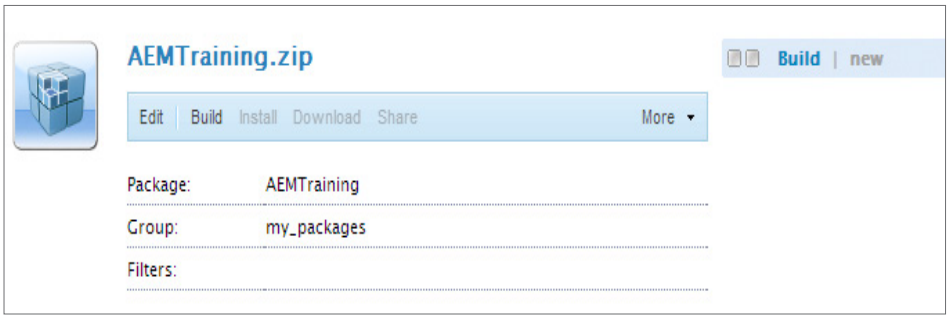
### EXERCISE 13.1 - Create a Content Package of everything we have done

The following instructions explain how to create a AEM package that will combine all elements of the training project, minus all jpegs. This is a good example of packaging application content, which you could then distribute to team members for review.

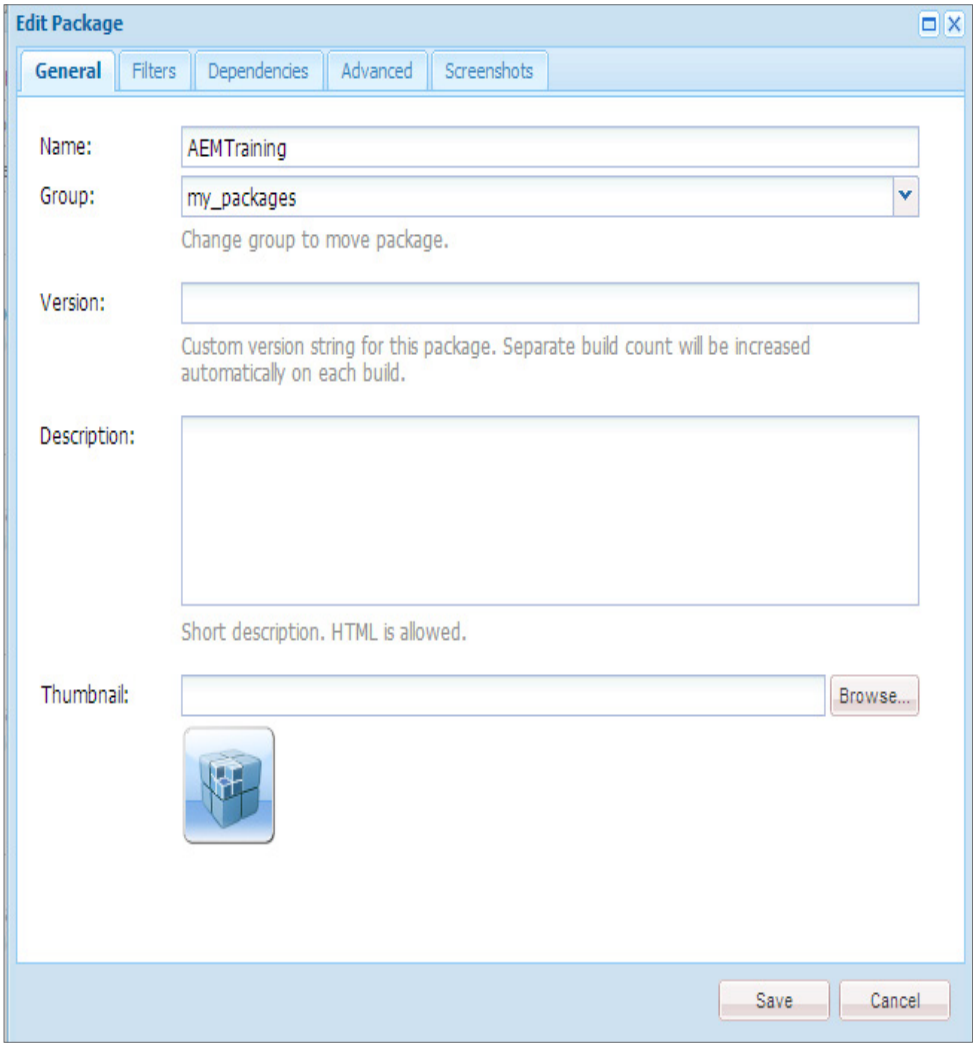
1. Navigate to projects.
   *http://localhost:4502*
2. In the left pane, select **Tools** > **Operations** > **Packaging** > **Package**.
3. Select **Create Package**.
4. Enter the package name as AEMTraining and click **OK**.



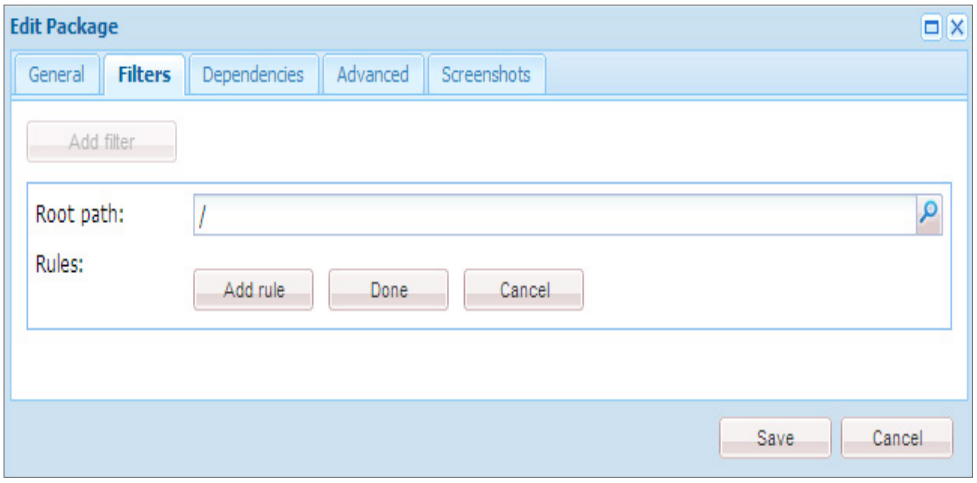
5. The result is an empty package definition.



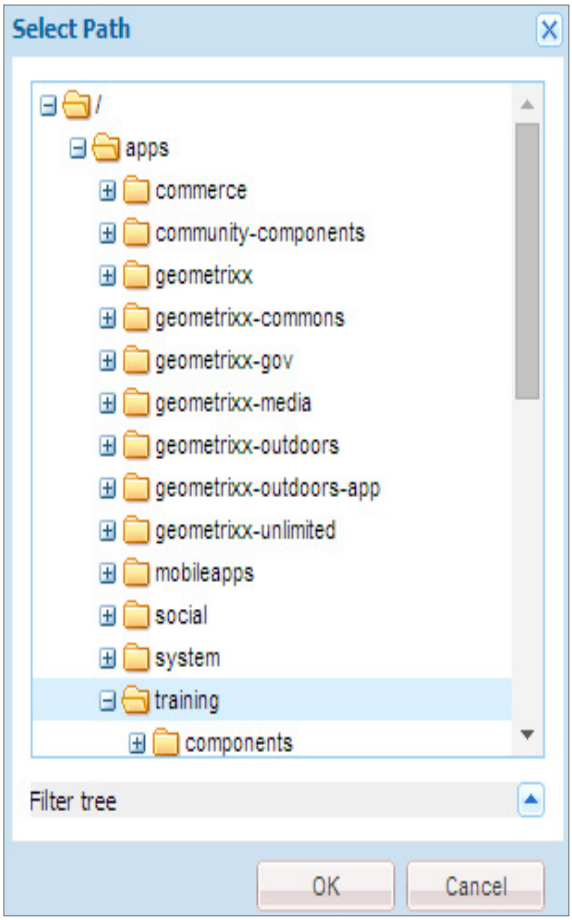

6. Click the **Edit** button to open the Edit dialog.



7. Select the **Filters** tab and Click **Add Filter.**




8. Click the browser icon to select the paths to be included in the package. Choose */apps/training.*



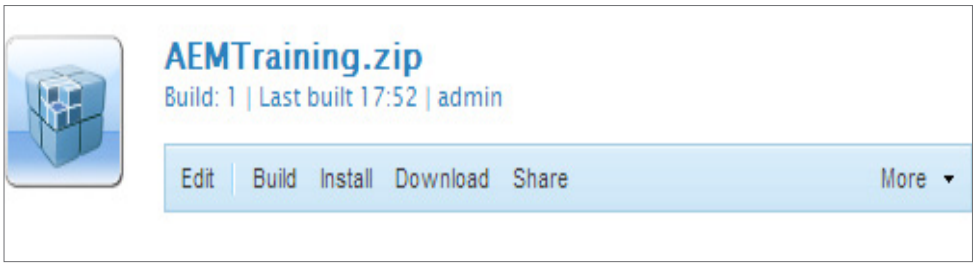
Package Filter Paths

9. Click **OK,** and then click **Done**.

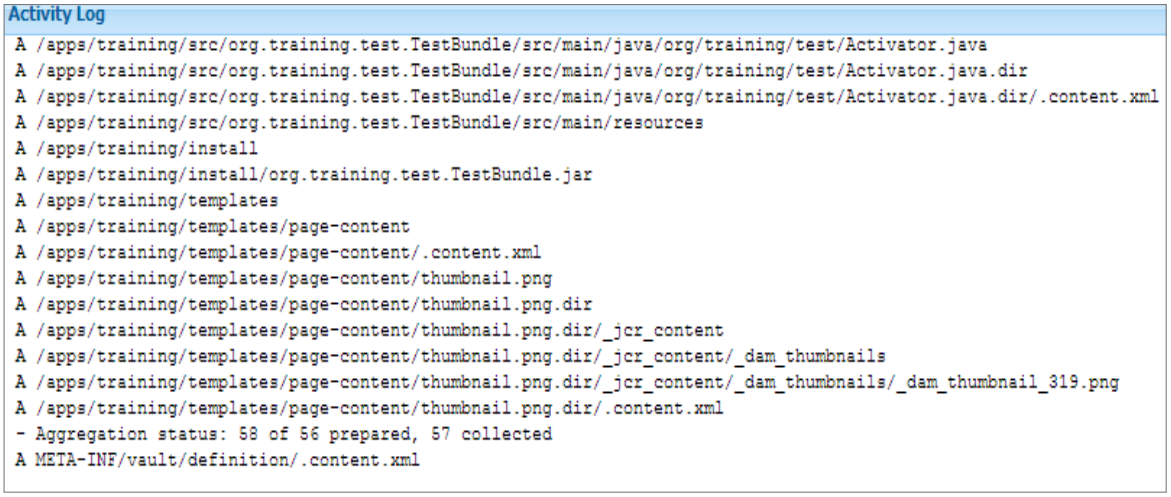
10. Repeat steps 7-8 for the following paths:
    - */apps/foundation/components/breadcrumb* (if you put the breadcrumb in /apps/foundation)
    - */etc/designs/trainingDesign*
    - */apps/training*
    - */etc/clientlibs/training*
    - */content/training-site*

    This creates a package definition to save all elements of the training project into the content package. You can then upload all these structures into any instance of AEM.

11. Click **Save**. Click **Build** to build the package.




12. Wait for the package to be built.



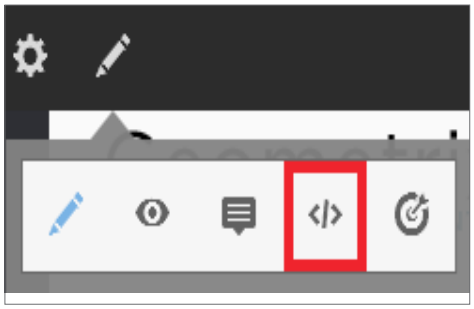

13. Click **Download** to download the package.

# Useful Tools

The following parameters will help you monitor what is being rendered:

## Developer mode in Touch-Optimized UI

You can access the Developer, in the Touch-Optimized UI as shown below:

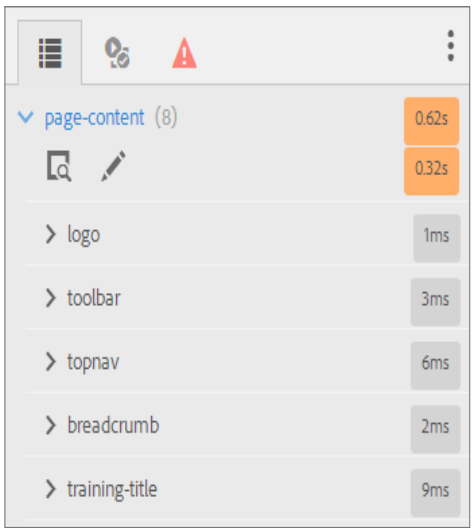


The Developer mode is provided you with the following options:

- Components
- Test
- Errors

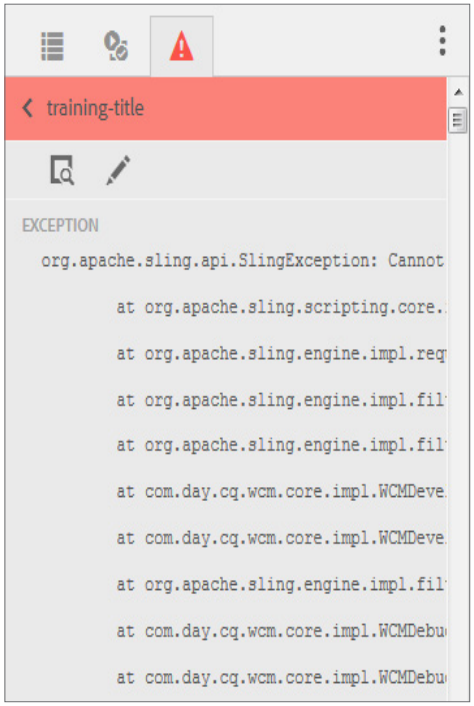
### Components

The Components tab provides you with a components tree of used components in the page. For each component, it provides you 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.



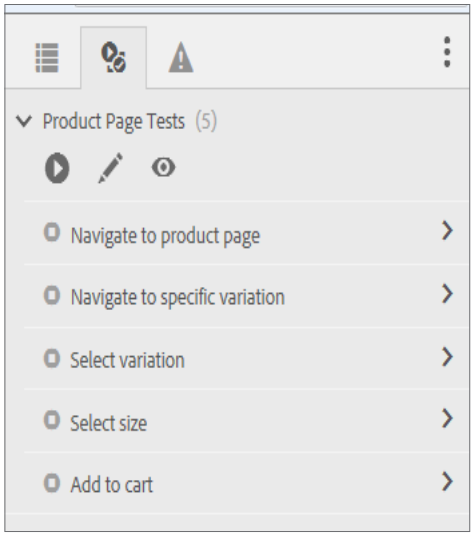
### Errors

The Error tab displays the error details of the broken components. You can see the required details without inspecting the log files.




### Tests

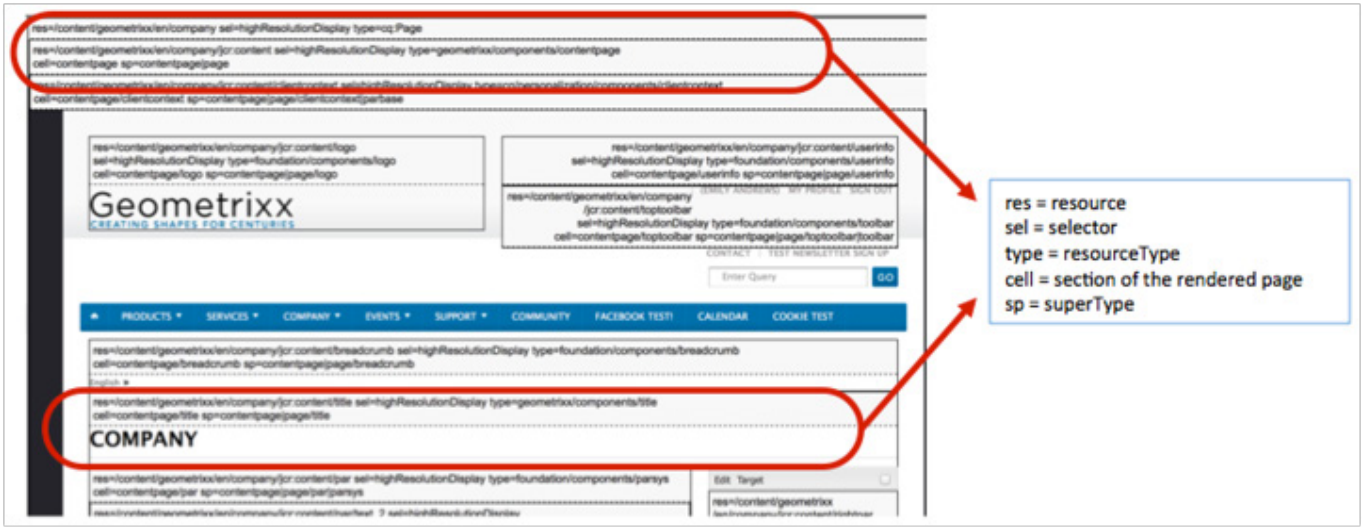
The Test tab allows you to write and execute tests directly in AEM. You can use *Hobbes.js* as the functional UI Testing framework. You can create various test suites and add test cases. Test cases can also be run in the Demo mode. The Demo mode is slow compared to the actual Run mode. You can also click the Edit button to navigate to the scripts that you wrote for the tests. The tab also displays the test results in the same page.



## debug=layout

Use this parameter to get information about renderers, selectors, resources, etc.

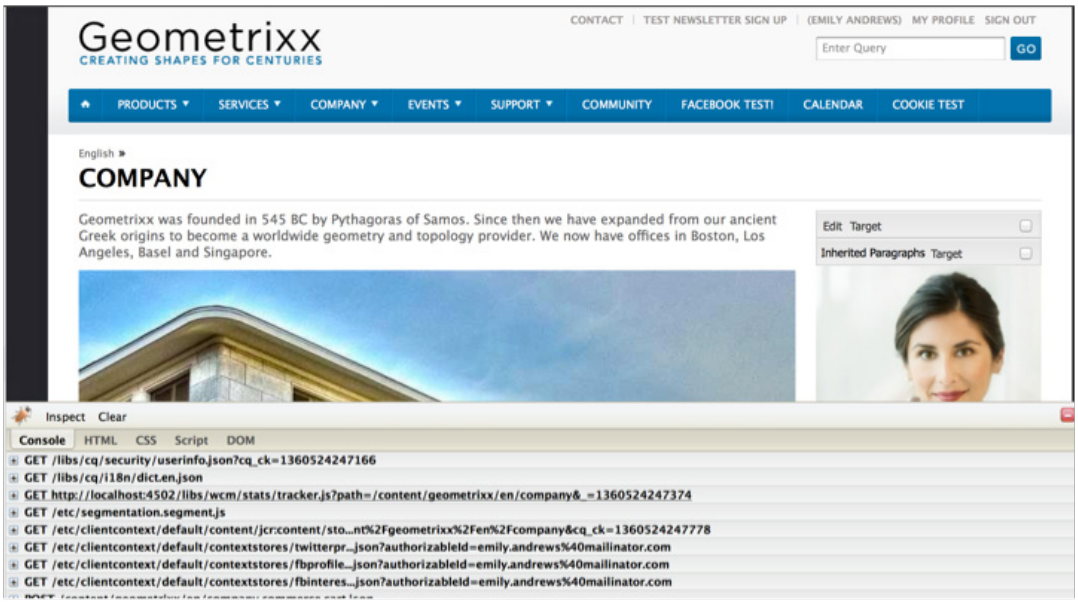
For example: http://localhost:4502/geometrixx/en/company.html?debug=layout




## 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 into the page shown in the source.

For example:
http://localhost:4502/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 web site takes to respond to visitor requests. Although this value will vary for each request, an average target value can be defined. Once this value is proven to be both achievable and maintainable, it can be used to Authors, who add and update the content, use this environment. monitor the performance of the web site and indicate the development of potential problems.

The response times you will be aiming for will be different on the author and publish environments, reflecting the different characteristics of the target audience:

### Author Environment

Authors, who add and update the content, use this environment. It must cater for small number of users—who generate a high number of performance intensive requests—when updating content pages and individual elements on those pages.

### Publish Environment

This environment contains content that you make available to your users, where the number of requests is even greater and the speed is just as vital; since, 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 summed up to five simple rules that can be followed 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.

### Planning 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 the level of complexity of a project and the experience of the development team. While your project may ultimately not require all of 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 in order to gather real-life experience and perform further optimizations, without the additional pressure that follows a full announcement. Once 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.

Since 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 web site and you 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, once people are focused on specific performance goals, it is very hard to change these goals afterwards, 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 web site (for example, the new web site'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. In order to consider this aspect properly, implement an agile validation process in the optimization phase rather than a more heavy-weight 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-1000ms.
- No pages should respond slower than 1 second.

The above numbers assume the following conditions:

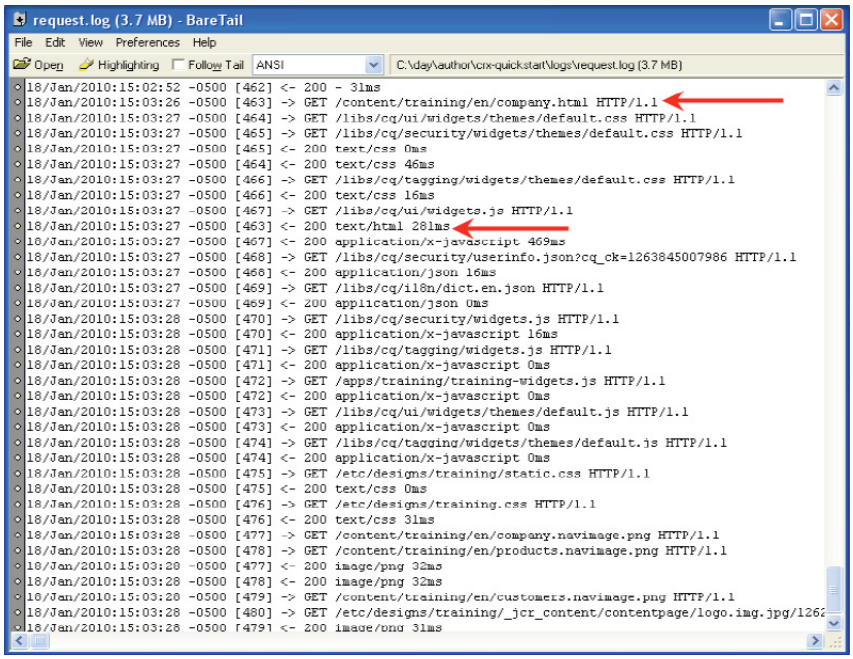
- measured on publish (no authoring environment and/or CFC overhead)

- 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, ...)
- no other load on the system

There are a certain number of issues that frequently contribute to performance issues, which mainly revolve around (a) dispatcher caching inefficiency; (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 13.2 - 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.
   - e.g. */content/trainingSite/en/company*
3. Review the response times directly related to the previous step's request.
   - A Page request of /content/trainingSite/en/company




Request.log response time

---

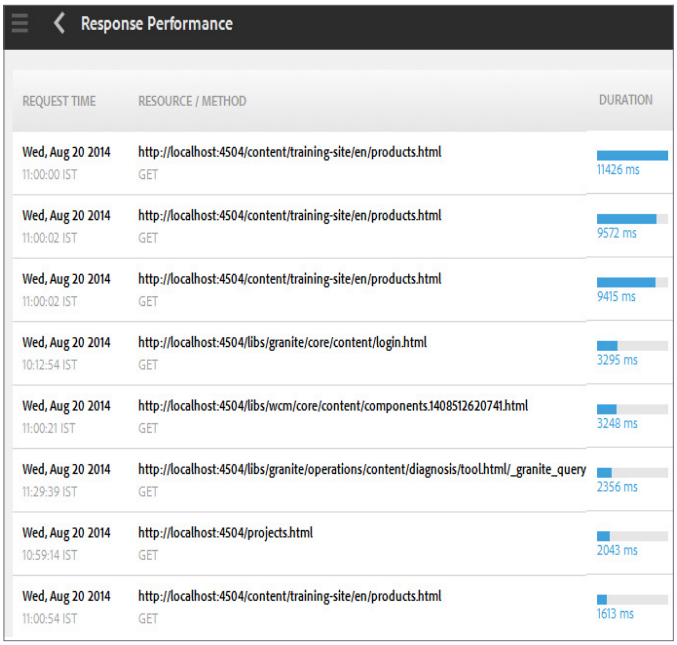
NOTE: Though this is clearly the response time of a page while in author, it is a good practice to be able to identify where response times are located (request.log) and how to identify a specific page request /response in the log files itself. Ideally, this test would occur on the publish instance to get a better idea of its expected behavior in production.

---

You have now 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 13.3 - Find the response performance

1. Log into 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.

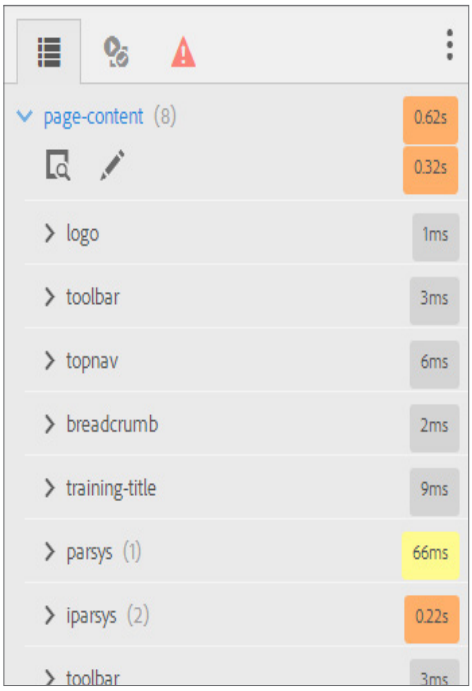



You can use this information to further investigate the cause of the long response.
You have now successfully found and displayed long lasting requests/responses in AEM. Again, this is just one of many tools to help you meet your project's performance goals.

## EXERCISE 13.4 - How to monitor Component based timing

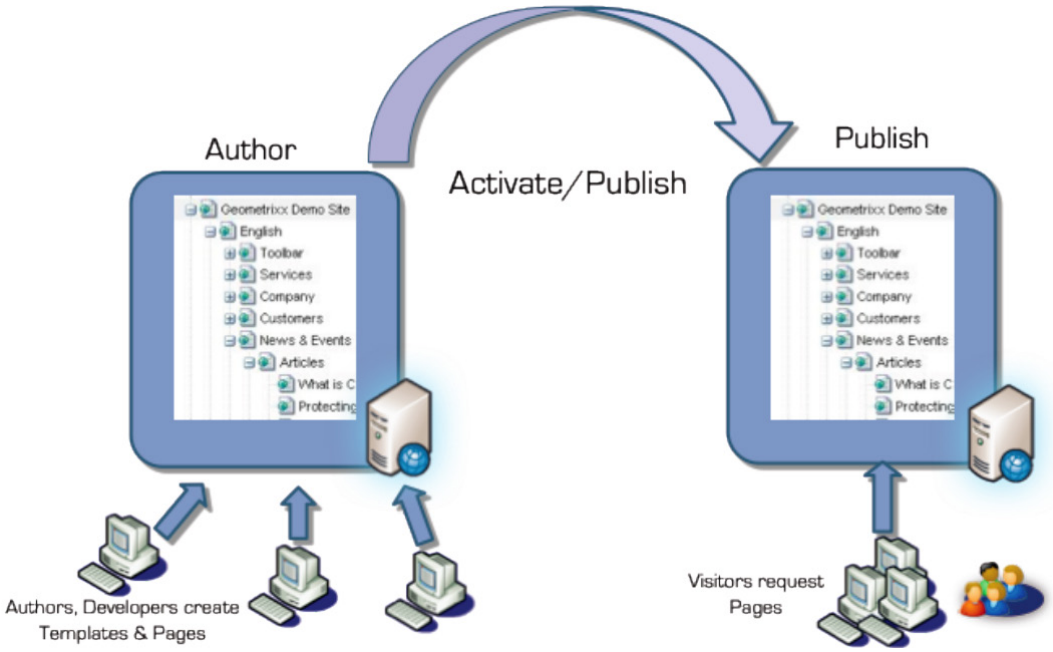
In the Developer mode of the page, you can view the time taken by the components to load.

1. Open the web page and move to the Developer mode.
2. Go to the Components tab.
   The Components tab displays a component tree that provides you with the server-side computation time for each component.



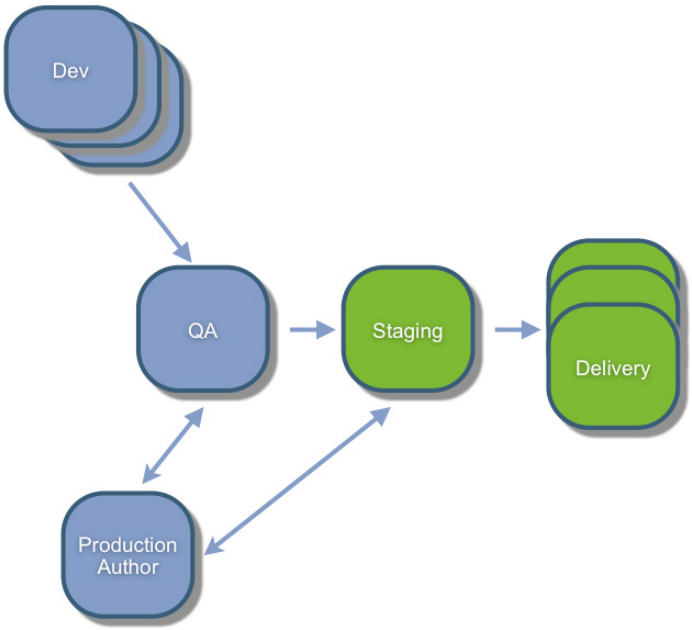
# AEM Deployment

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.



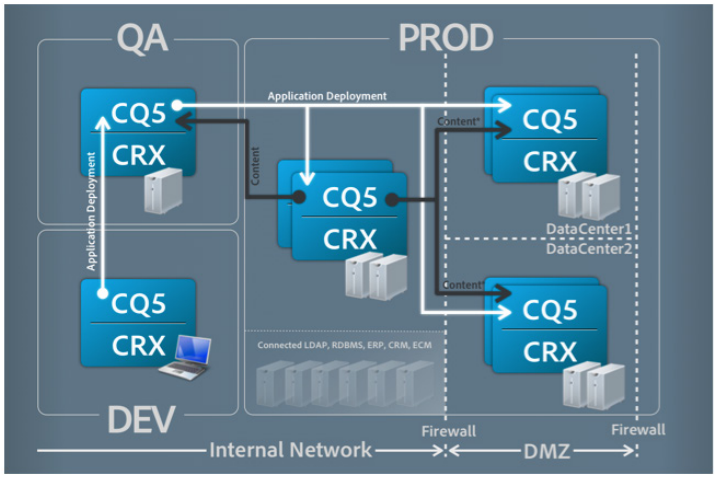

Depending on the scale of a project, a production environment may consist of several author and/or publish instances, and at a lower level, the CRX repository may be clustered among several instances as well. Additionally, separate development and test environment levels may also consist of both author 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.

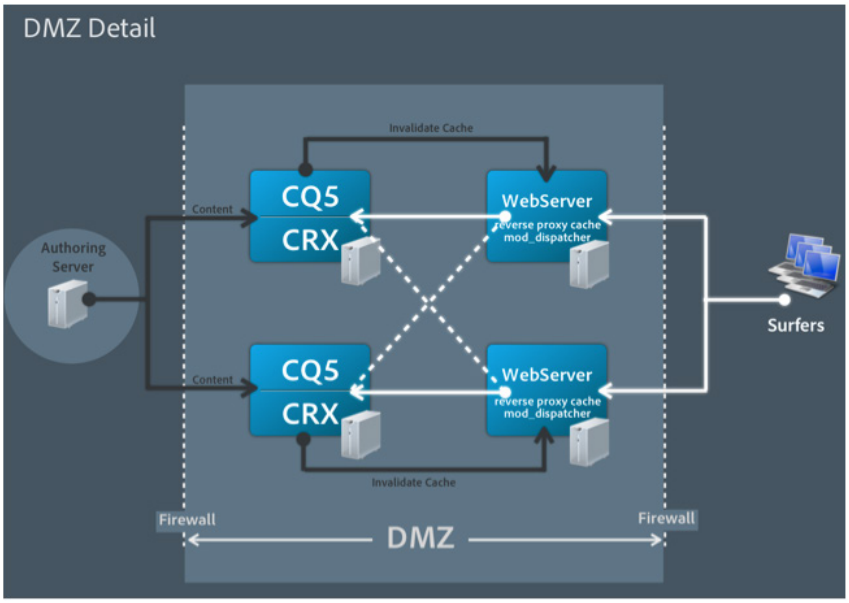



Content and code often move in different directions and get to their destination through different mechanisms. Content typically is moved by use of the replication agents. Content will move forward from Author to Publish, but also may move back to QA. This allows testing to be done on real data.

Deployment of code and configuration information is typically done through automated processes that use content packages, replication agents, and/or filevault.

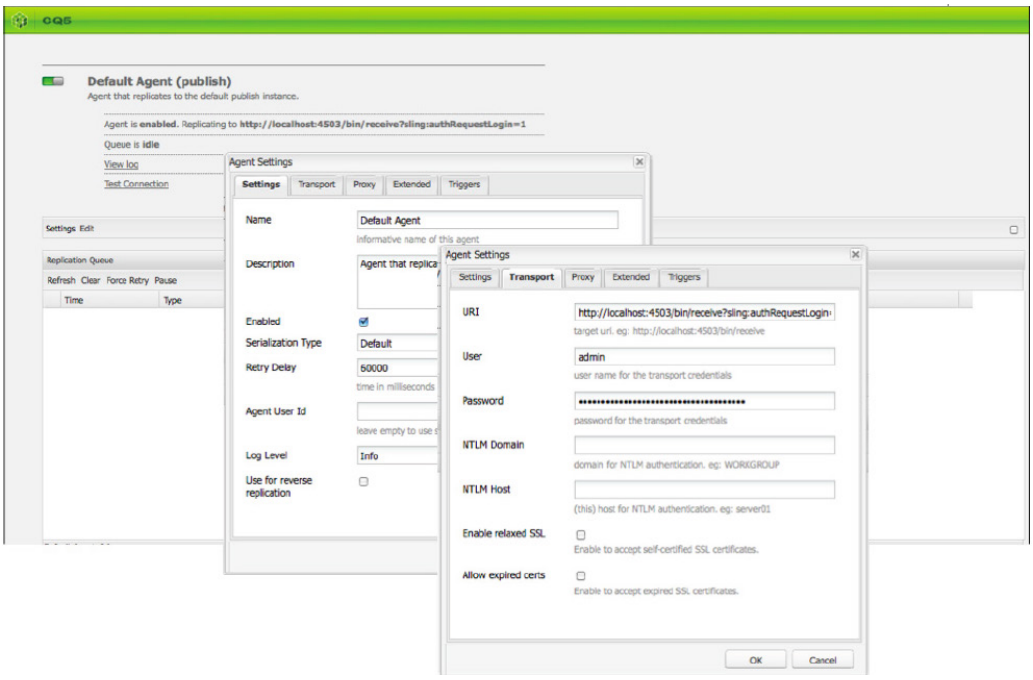



There are many designs for deployment of the AEM application itself. One of them is pictured below:




## 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 4 publish instances, you need 4 Replication Agents.

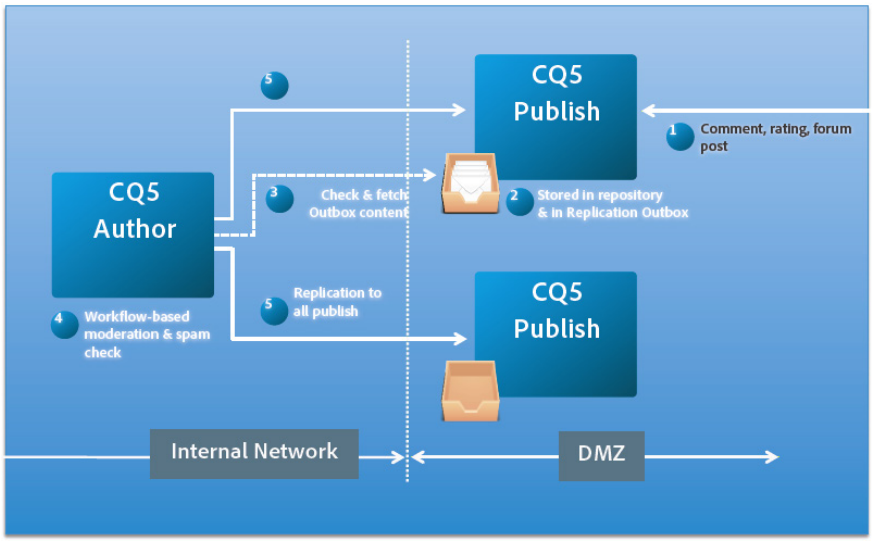



Details on creation and management of Replication Agents can be found in the documentation at:

http://dev.day.com/docs/en/cq/current/deploying/configuring_cq.html

## Reverse Replication

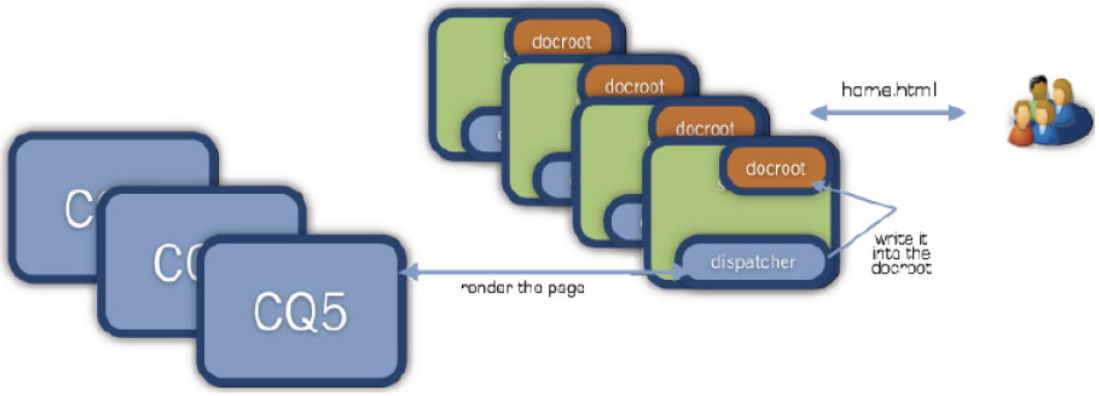
Reverse Replication allows the transfer of content from the web site visitors (the publish instance) to the author instance for review, moderation, or spam check without violating any firewall rules.



The available Replication Agents

## Dispatcher

In the production delivery environment, the publish instance is joined by 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 both 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 web site
- Accessing the layout engine as little as possible.

Which means that:

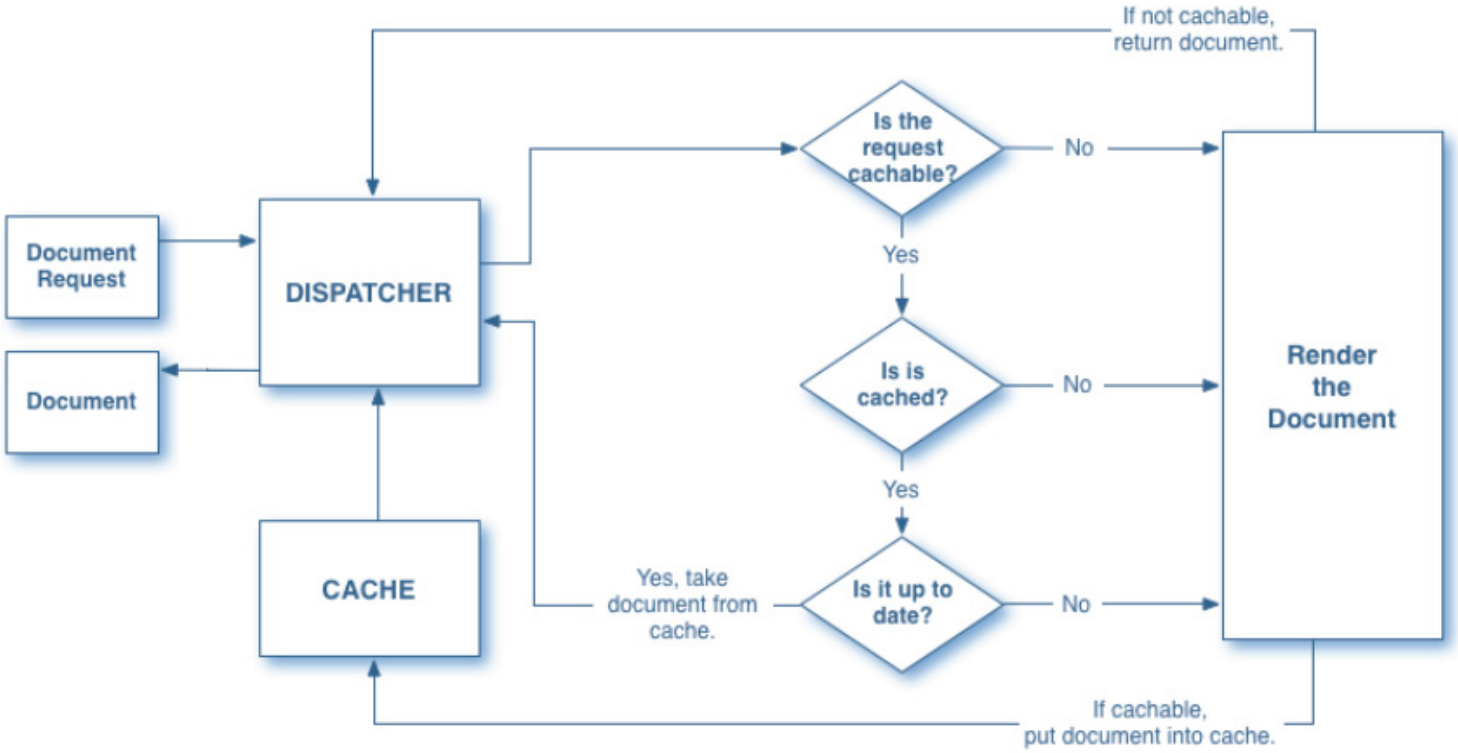
- Static content is handled with exactly the same speed and ease as on a static web server; additionally, you can use the administration and security tools available for your static web server(s).
- Dynamic content is generated as needed, without slowing the system down any more than absolutely necessary.

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 Author instance for the related Replication Agent log files using the configuration panel you used, as you edited the settings.

---




Details regarding the management and configuration of the Dispatcher can be found at:

http://dev.day.com/docs/en/cq/current/deploying/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 cachable.