Contents

[AEM (ADOBE EXPERIENCE MANAGER) 3](#_Toc80259576)

[AEM TECHNOLOGY STACK 3](#_Toc80259577)

[STARTING AEM 3](#_Toc80259578)

[AEM PROJECT – MAVEN ARCHETYPE- 23 4](#_Toc80259579)

[SYNCING AEM WITH FILESYSTEM 5](#_Toc80259580)

[RUN MODES 5](#_Toc80259581)

[DEBUGING BUNDLE CODE 7](#_Toc80259582)

[CREATING TEMPLATES 8](#_Toc80259583)

[CREATING COMPONENTS 12](#_Toc80259584)

[INITIALIZING WCM 21](#_Toc80259585)

[DIALOG & DESIGN DIALOGS 22](#_Toc80259586)

[CLIENTLIBRARY 25](#_Toc80259587)

[CLIENT-LIBRARY PROPERTIES 26](#_Toc80259588)

[OSGI 28](#_Toc80259589)

[OSGI CONFIGURATION – USING R7 ANNOTATIONS 32](#_Toc80259590)

[SLING SERVLET 39](#_Toc80259591)

[EVENT LISTENER IN AEM 41](#_Toc80259592)

[SCHEDULERS 44](#_Toc80259593)

[HTML TEMPLATE LANGUAGE (HTL) – SIGHTLY 46](#_Toc80259594)

[TAGGING 49](#_Toc80259595)

[WORKFLOW 53](#_Toc80259596)

[REPLICATION 59](#_Toc80259597)

[VLT (VAULT) TOOL 59](#_Toc80259598)

[AEM RESPONSIVE GRID SYSTEM 61](#_Toc80259599)

[SITEADMIN – IN DETAIL 62](#_Toc80259600)

[SLING SCRIPT RESOLUTION 65](#_Toc80259601)

[SECURITY 66](#_Toc80259602)

[QUERYBUILDER API 69](#_Toc80259603)

[MULTI SITE MANAGER 70](#_Toc80259604)

[INTERNATIONALIZATION 85](#_Toc80259605)

[PERSONALIZATION 87](#_Toc80259606)

[SOME USEFUL CODE 100](#_Toc80259607)

[STEPS SETTING UP MONGO DB WITH AEM 6.0 101](#_Toc80259608)

[ENABLING MOBILE EMULATOR 102](#_Toc80259609)

[STEP 3 : MOBILE EMULATOR IN ACTION 102](#_Toc80259610)

[DISPATCHER 103](#_Toc80259611)

[CURL 114](#_Toc80259612)

[AEM LOGS 117](#_Toc80259613)

[SLING RESOURCE MERGER 119](#_Toc80259614)

[CORE COMPONENTS 125](#_Toc80259615)

[CONTENT FRAGMENTS 129](#_Toc80259616)

[EXPERIENCE FRAGMENTS 134](#_Toc80259617)

[VERSIONING 134](#_Toc80259618)

[VERSION MANAGER 134](#_Toc80259619)

[CREATE MULTIPLE BUNDLES IN AN AEM MAVEN PROJECT 135](#_Toc80259620)

[FREQUENTLY USED CQ URLS 137](#_Toc80259621)

[ACS COMMONS 139](#_Toc80259622)

# AEM (ADOBE EXPERIENCE MANAGER)

## AEM TECHNOLOGY STACK

|  |  |
| --- | --- |
|  |  |

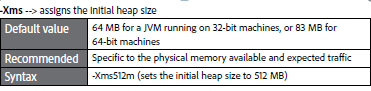
* AEM comes with in-built OSGI Container, which manages the bundles
* MANIFEST file takes care of the visibility of the bundle.
* Apache Felix an implemtation of OSGI framework which can be accessed using /system/console.

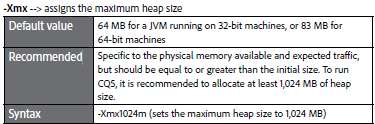
### STARTING AEM

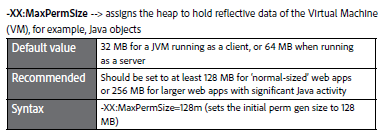
**Install or Start AEM Using a Command Line**

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

Typical command line start 🡪 $ **java -Xmx1024m -jar CQ-author-p4502 –gui**







1. The CQ Server runs on an application server called Jetty server.

**What is PERM Size:** The permanent generation is special because it holds meta-data describing user classes (classes that are not part of the Java language). Examples of such meta-data are objects describing classes and methods and they are stored in the Permanent Generation. Applications with large code-base can quickly fill up this segment of the heap which will cause java.lang.OutOfMemoryError

1. Suppose if you create a class name A, its instance variable will be stored in heap memory and class A along with static classloaders will be stored in permanent generation.
2. Garbage collectors will find it difficult to clear or free the memory space stored in permanent generation memory. Hence it is always recommended to keep the permgent memory settings to the advisable limit.
3. JAVA8 has introduced the concept called meta-space generation; hence permgen is no longer needed when you use jdk 1.8 versions.

### AEM PROJECT – MAVEN ARCHETYPE- 23

**MAVEN COMMAND:**

**mvn -B archetype:generate -DarchetypeGroupId=com.adobe.granite.archetypes -DarchetypeArtifactId=aem-project-archetype -DarchetypeVersion=23 -DaemVersion=6.5.0 -DappTitle="AEM Geeks" -DappId="aemgeeks" -DgroupId="com.aemgeeks" -DfrontendModule=none -DincludeDispatcherConfig=n -DlanguageCountry="en\_us" -DsingleCountry=y -DincludeExamples=n**

| **Name** | **Default** | **Description** |
| --- | --- | --- |
| **appTitle** |  | For website title and component groups |
| **appId** |  | For app, conf and content folder names; clientlib names |
| **artifactId** | *${appId}* | Base Maven artifact ID |
| **groupId** |  | Base Maven group ID |
| **package** | *${groupId}* | Java Source Package |
| **version** | 1.0-SNAPSHOT | Project version |
| **aemVersion** | 6.5.0 | Target AEM version (can be cloud for [AEM as a Cloud Service](https://docs.adobe.com/content/help/en/experience-manager-cloud-service/landing/home.html); or 6.5.0, 6.4.4, or 6.3.3 for [Adobe Managed Services](https://github.com/adobe/aem-project-archetype/tree/master/src/main/archetype/dispatcher.ams) or on-premise) |
| **sdkVersion** | latest | When aemVersion=cloud an [SDK](https://docs.adobe.com/content/help/en/experience-manager-cloud-service/implementing/developing/aem-as-a-cloud-service-sdk.html) version can be specified |
| **includeDispatcherConfig** | y | Includes a dispatcher configuration either for cloud or for AMS/on-premise, depending of the value of aemVersion (y/n) |
| **frontendModule** | none | Includes a Webpack frontend build module that generates the clientlibs (general or none; angular or react for SPA) |
| **languageCountry** | en\_us | Language and country code to create the content structure from (e.g. en\_us) |
| **singleCountry** | y | Includes a language-master content structure (y/n) |
| **includeExamples** | y | Includes a [Component Library](https://www.aemcomponents.dev/) example site (y/n) |
| **includeErrorHandler** | n | Includes a custom 404 response page that will be global to the entire instance (y/n) |

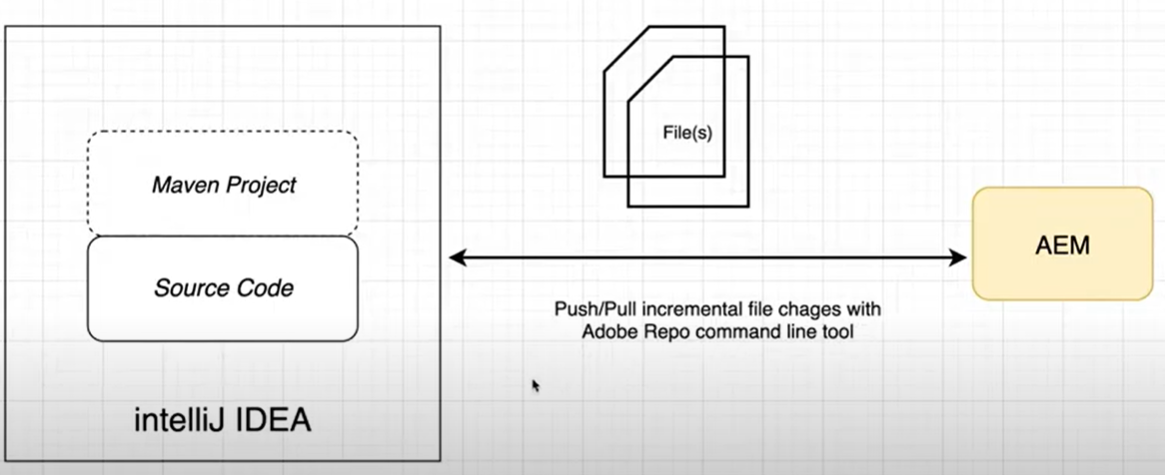
#### PROJECT STRUCTURE

|  |  |
| --- | --- |
|  | **The project created by this archetype creates a multi-module project. After the all the module at bundled in the target folder inside the “all” folder as a “.zip” file**   * **core**: Java bundle containing all core functionality like OSGi services, listeners, or schedulers, as well as component-related Java code such as servlets or request filters. * **ui.apps**: contains the /apps (and /etc) parts of the project, ie JS & CSS clientlibs, components, templates, runmode specific configs as well as Hobbes-tests * **ui.content**: contains sample content using the components from the ui.apps * **ui.tests**: Java bundle containing JUnit tests that are executed server-side. This bundle is not to be deployed onto production. * **ui.launcher**: contains glue code that deploys the ui.tests bundle (and dependent bundles) to the server and triggers the remote JUnit execution * **ui.frontend**: an optional dedicated front-end build mechanism (Angular, React or general Webpack project) |

#### ADDING CORE COMPONENTS AND ACS COMMONS DEPENDECIES

### SYNCING AEM WITH FILESYSTEM

The AEM Repo Tool is a simple solution to transfer JCR content between your local filesystem and the AEM server via the command line comparable to FTP. The AEM Repo Tool is similar to the [Jackrabbit FileVault tool](https://experienceleague.adobe.com/docs/experience-manager-64/developing/devtools/ht-vlttool.html?lang=en), but is faster, has minimal dependencies, and is a simple bash script.



### RUN MODES

Run modes allow us to tune the AEM instance for a specific purpose; for example author or publish, test, development, intranet or others.

**WHY RUN MODES?**

1. Uniquely identify an environment and instances
2. Unique configurations based on environment
3. OSGI Component Creation for a specific environment
4. Bundle Creation for a specific environment

**TYPES OF RUN MODES**

1. Primary Run Mode
2. Secondary Run Mode

#### PRIMARY RUN MODES

1. **Author**: This instance is used for the complete development and authoring purpose.
2. **Publish**: This is the actual environment which can be accessed by end users.
3. **nosamplecontent**: This instance is having no sample content(like geometrixx,we-retail not available).It is highly recommended in production environment because it is very secure and it provides no sample configurations.lt makes your instance production ready, by disabling CRXDE lite, webdav etc
4. **samplecontent**: having sample content like geometrixx-all package.This is just for the help of developers,not required on any server.

Note: Primary Run Modes can’t be changing once the aem jar is started. So at the time of AEM instance startup only, we need to finalize which primary run mode is required.

#### SECONDARY RUN MODES

1. Dev Server
2. QA Server
3. UAT Server
4. Prod Server

**CUSTOMIZED RUN MODES: You** can also create your own, customized, run modes. These can be combined to cover scenarios such as: author + development; publish + test; publish + test + golive; publish + intranet;. Customized run modes can also be selected at each startup.

**DEFINING CONFIGURATION PROPERTIES FOR A RUN MODE**

A collection of values for configuration properties, used for a particular run mode, can be saved in the repository.

The run mode is indicated by a suffix on the folder name. This allows you to store all configurations in one repository as. For example:

* Config 🡪Applicable for all run modes
* config.author🡪Used for author run mode
* config.publish 🡪Used for publish run mode

**STARTING CQ WITH A SPECIFIC RUN MODE**

If you have defined configurations for multiple run modes then you need to define which is to be used upon startup. There are several methods for specifying which run mode to use; the order of resolution is:

* sling.properties file
* -r option
* system properties (-D)

**USING THE SLING.PROPERTIES FILE:** The sling.properties file can be used to define the **required run mode:**

**Step 1: Edit the configuration file:** <cq-installation-dir>/crx-quickstart/conf/sling.properties

**Step 2:** Add the following properties; the following example is for author: sling.run.modes=author

**USING THE -R OPTION: A** custom run mode can be activated by using the -r option when launching the quickstart. For example, use the following command to launch an AEM instance with run mode set to dev. - java -jar cq-56-p4545.jar **-r** dev

**USING A SYSTEM PROPERTY IN THE START SCRIPT:** A system property in the start script can be used to specify the run mode.

For example use the following to launch an instance as a production publish instance located in the US:

-Dsling.run.modes=publish,prod,us

**FILENAME DETECTION - RENAMING THE JAR FILE:** Two installation run modes can be activated by renaming the installation jar file before installation:publish and author.

The jar file must use the naming convention: cq5-<run-mode>-p<port-number> mple, set the publish run mode by naming the jar file: cq5-publish-p4503

**ADVANTAGE /PURPOSE OF RUN MODES**

* OSGi Component Creation for a specific environment.
* Bundle Creation for a specific environment.

|  |  |
| --- | --- |
|  | **Example-** Let see how can load configurations based on run modes. In author mode we log the error in **“error.log”** file**.** We can check the configuration  Now we want to change this configuration value - say in author dev run mode it should log the errors in **error-dev.log** file.  **STEPS**   * Create a node of type sling folder with the name “**config.author.dev**” in apps folder * Copy the PID of the “Apache Sling Logging Configuration” and create a node of type “**sling:OsgiConfig**” with the same name * Create a property in the OSGI confuguration with the same name as the property name of Log file property name * Start the aem from command line   **java –jar <aem\_jar\_file\_name> -r author,dev** |
|  |  |
|  | |

**HOW TO CHECK THE RUN MODE:** Felix Console 🡪 Status TAB 🡪 Sling Setting



### DEBUGING BUNDLE CODE

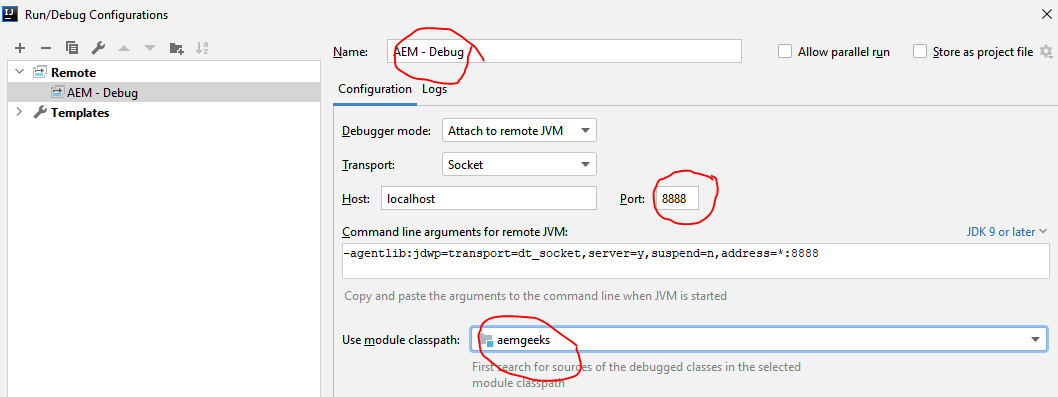
* Start AEM in Debug Mode using command

**java -Xdebug -agentlib:jdwp=transport=dt\_socket,address=8888,server=y,suspend=n -jar aem-author-p4502.jar**

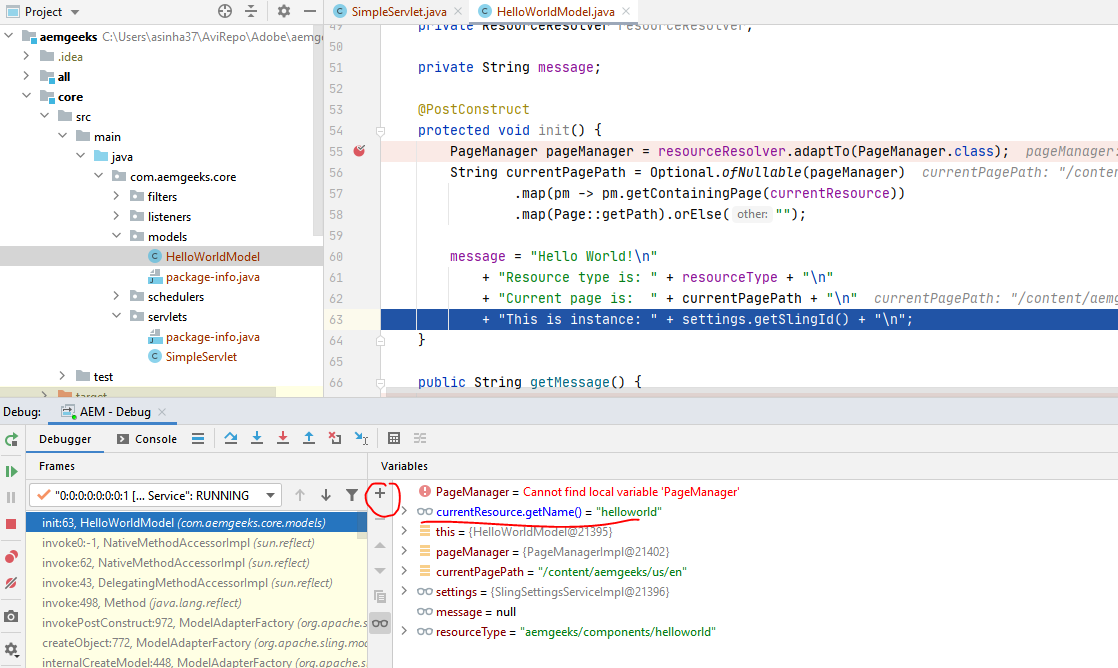
AEM will still run on 4502 port but allow another debugger port 8888 to debug

#### INTELLIJ CONFIGURATION

* Add a new configuration from Run 🡪 Edit Configuration.
* Create a new remote configuration and Configure the name, debug port of AEM and Module to debug



* Start the debug using debug icon: 
* Add the breakpoint
* We can add variables to debug the varible in the variable window.



### CREATING TEMPLATES

Template Types

1. Page Templates - Editable
2. Page Templates - Static
3. Content Fragment Templates
4. Adaptive Template Rendering

#### STATIC TEMPLATES

|  |  |
| --- | --- |
| **STEP 1**   1. Resource Type (sling: resourceType) 🡪 Path of page cpmponent 2. Ranking🡪 the order of the template appearance while creating a page. 3. Allowed Paths will define the paths where template may be used to create pages.   Regex in Allowed path🡪/content(/.\*)? 🡪 This means the pages from this template can be created at any place inside the content folder | |
| **STEP 2: ALLOWED PARENTS**   * It takes the template/templates name as an input. If the input is **templates/home-page** * The pages, which can be created using template which we are going to create now will have the parent pages that are created by using home-page template. | |
| **STEP 3: ALLOWED CHILDREN**   * It takes the template name as an input. * It will list out those templates which can be created underneath the given template | |
| **ALLOWED TEMPLATES**  **You have a page and want to create a child page. Which property has the highest priority to determine which templates can be used?-->** cq:allowedTemplates  **You have created two templates, tempA and tempB. In the property allowedChildren of tempA you include tempB. You create a pageA based on tempA and add a property cq:allowedTemplates with a list of templates, but excluding tempB. Can you select tempB to create a page as child of pageA?**  No, tempB needs to be added to the property cq:allowedTemplates of pageA to accomplish that.  **Precedence** [Refer below]: cq:allowedTemplates 🡪 cq:allowedPath🡪cq:allowedParents🡪cq:allowedChilderen | template-resolution-order |
| **ADDING THUMBNAIL TO TEMPLATE**:  Name of thumbnail image : thumbnail.png  Path of thumbnail image: /apps/<project\_name>/<template\_folder>  /<template\_name>/thumbnail.png |  |

**Note🡪** While creating a template, if a custom property is added in the jcr node of the template, the same property will be added to the jcr node of page, created by using that template.

#### EDITABLE TEMPLATE

Introduced in AEM 6.2

|  |  |
| --- | --- |
| **STATIC TEMPLATE** | **EDITABLE TEMPLATE** |
| Defined and configured by developer | Created and edited by template editor/author |
| Structure of the page will be same | Can define structure, content and content policies |
| No dynamic connection between content & page | Maintains dynamic condition between page and content |

**STEPS TO CREATE EDITABLE TEMPLATES**

|  |  |
| --- | --- |
| * Go to Tools 🡪 General 🡪 Configuration Browser * Create Configuration, Title should be your project name(my-project) and check on **Editable templates**. * It will create the basic hierarchy of templates in **/conf** directory. |  |
|  | **View the created Template**  Navigate to Tools 🡪General 🡪 Templates |

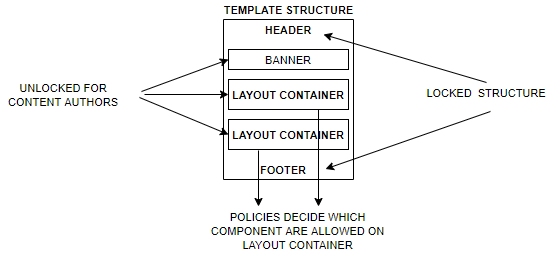
**EDITABLE TEMPLATE HIREARCHY**

* **Templates** - Here all the editable templates are contained which are created by template authors.
* **policies**: It defines which components are allowed within the layout container (something like design view)
* **Template-types**: This is a base template based on which template author creates all editable templates at the run time.

**STEPS TO CREATE EDITABLE TEMPLATE**

|  |  |
| --- | --- |
| 1. Open the template and select the Template Types 2. Select “**HTML5 Page**” Template🡪 Next 3. Enter the template details (Template Title & Description) and Click Create 4. Open the newly created editable template to define the page structure. |  |
|  | **MODE IN TEMPLATE EDITOR** |

**THE EDITABLE TEMPLAE CAN BE EDITED IN 3 MODES**



1. **STRUCTURE:** In this mode we define the basic structure of the editable template.
2. **INITIAL CONTENT:** 
   1. Initial Content defines content that will appear when a page is first created based on the template.
   2. The initial content is locked for the content authors e.g. headers and footers can be considered as initial content.
3. **LAYOUT**
   * In this mode – different container component like “content fragment” or simple component like text component are resized in the layout container.
   * **Once the placeholders are set-up. The component can be dropped – depending upon policies of layout container.** (Same as we do in design mode of the template)

|  |  |
| --- | --- |
| **IMPLEMENTING POLICIES FOR LAYOUT CONTAINER**   1. **Switch to structure mode** and Click of Policy (as highlighted below) 2. Either select a predefined policy from Drop Down or create a **New Policy (Need to give policy title for every new policy)** 3. Select the component which will bind with policy | |
|  |  |

### CREATING COMPONENTS

**IMPORTANT PROPERTIES AND CHILD NODES OF A COMPONENT**

|  |  |  |
| --- | --- | --- |
| **NAME** | **TYPE** | **DESCRIPTION** |
| **cq:isContainer** | **String** | Indicates whether the component is a container component and therefore can contain other components such as a paragraph system. |
| **dialogPath** | **String** | Path to a dialog to cover the case when the component does not have a dialog node. |
| **cq:cellName** | **String** | [If set, this property is taken as Cell ID.](https://helpx.adobe.com/experience-manager/kb/DesigneCellId.html) |
| **cq:childEditConfig** | **cq:EditConfig** | When the component is a container, as for example a paragraph system, this drives the edit configuration of the child nodes. |
| **cq:htmlTag** | **nt:unstructured** | Returns additional tag attributes that are added to the surrounding html tag. Enables addition of attributes to the automatically generated divs. |
| **cq:noDecoration** | **Boolean** | If true, the component is not rendered with automatically generated div and css classes. |
| **cq:template** | **nt:unstructured** | If found, this node will be used as a content template when the component is added from the Components Browser or Sidekick. |
| **cq:templatePath** | **String** | Path to a node to use as a content template when the component is added from the Components browser or Sidekick. This must be an absolute path, not relative to the component node. |
| Unless you want to reuse content already available elsewhere, this is not required and **cq:template** is sufficient. |
| **virtual** | **sling:Folder** | Enables creation of virtual components. To see an example, please look at the contact component at: |
| **/libs/foundation/components/profile/form/contact** |
| **icon.png** | **nt:file** | Icon of the component appears next to the Title in Sidekick. |
| **thumbnail.png** | **nt:file** | Optional thumbnail that is shown while the component is dragged into place from Sidekick. |

**cq:template & cq:templatePath**

Both these node[**cq:template**] and property[**cq:teamplatePath**] in used to give a default value to a field in the dialog. The Component will take the default value- whenever it is dragged on the page

**cq:template**

|  |  |
| --- | --- |
| Let’s say we have a component having text field- for which we want to set a default value. |  |
| 1. Create a nt:unstructured node below the component 2. Create a node below it and same property name as in dialog |  |
| Drag the component on the page – the component will be pre-populated with default value set in cq:template |  |

**cq:templatePath**

|  |  |
| --- | --- |
| Let’s say we have a component having text field- for which we want to set a default value. |  |
| Create a sling folder and add a child node (nt:unstructured)  Sling Folder Path : [/apps/company/components/templateConfig]  Child Node path : /apps/company/components/templateConfig/**text** |  |
| 1. Add a property cq:templatePath on the component node 2. cq:templatePath: /apps/company/components/temple?eConfig/text 3. Drag the component on the page – the component will be pre-populated with default value set in cq:template |  |
| 1. When an author drops the component, the component fetches all the values from its templatePath or cq:template node and store them in the /content hierarchy of that component.Author can change the default values from dialog 2. Use Case: The use case of these properties can be if you have one style tab in a component where you decide font size, color, font-family ( something like theme), you can set the initial default theme of a component. 3. Advantage of templatePath is that it can be used in multiple components | |

**cq:noDecoration**

|  |  |
| --- | --- |
| **When cq:Decoration is false or not present** | **WHEN cq:Decoration = true** |

**cq:htmlTag**

|  |  |
| --- | --- |
| Returns additional tag attributes that are added to the surrounding html tag. Enables addition of attributes to the automatically generated divs. |  |

|  |  |
| --- | --- |
| **IMPLICT OBJ** | **DESCRIPTION** |
| slingRequest | The wrapped Request Object (SlingHttpServletRequest). |
| slingResponse | The wrapped Response Object (SlingHttpServletResponse). |
| resource | The Sling Resource Object (slingRequest.getResource();). |
| resourceResolver | The Sling Resource Resolver Object (slingRequest.getResoucreResolver();). |
| currentStyle | The style of the addressed resource |
| properties | The properties of the addressed resource (resource.adaptTo(ValueMap.class);). |
| designer | The designer object for retrieving design information (resourceResolver.adaptTo(Designer.class);). |
| currentDesign | The design of the addressed resource |
| component | The component object of the current AEM component.. |
| pageProperties | The properties of the page of the addressed resource. |
| pageManager | The page manager for accessing AEM content pages (resourceResolver.adaptTo(PageManager.class);) |
| sling | The Sling script helper |
| log | Default Logger |
| currentNode | The resolved JCR node for the request |

#### CREATING COMPONENTS USING - SLING MODELS

In simple terms Sling Models are simple POJO classes which are mapped automatically with Sling Objects (resource, request objects etc.) and allow us to access jcr node property values directly into java classes.

#### WRITING A SLING MODEL



STEPS TO CREATE SLING MODEL

**STEP 1: CREATE AN INTERFACE (OPTIONAL)**

**STEP 2: CREATE AN IMPLEMENTATION CLASS**

**All Sling Models are anootated with @Model and an adaptable class. The Sling model can be validated in OSGI console in** Sling🡪Sling Adapter

ATTRIBUTES OF MODEL ANNOTATION

|  |  |
| --- | --- |
| **adaptable** | It tells which source object can be adapted to Sling model. The source object can be   * Resource * SlingHttpServletRequest |
| **adapters** | Interface of Sling Model |
| **defaultInjectionStrategy** | **DefaultInjectionStrategy.*OPTIONAL****: All properties of Sling model are optional*  **DefaultInjectionStrategy.*REQUIRED****: All properties of Sling model are required* |

|  |  |
| --- | --- |
| INTERFACE | **IMPL CLASS** |
| public interface Employee {  String getFirstName();  String getLastName();  boolean getIsPermanent();  } | @Model(adaptables = Resource.class,  adapters = Employee.class,  defaultInjectionStrategy = DefaultInjectionStrategy.**OPTIONAL**)  public class EmployeeImpl implements Employee {  @Inject  @Default(values = "AEM")  String fname;  @Inject  @Required  @Default(values = "Geeks")  String lname;  @Inject  boolean permanent;  @Override  public String getFirstName() {  return fname;  }  @Override  public String getLastName() {  return lname;  }  @Override  public boolean getIsPermanent() {  return permanent;  }  } |
| * The property names (fname, lname) in the sling model should match with the name of the dialog field * The Source object a **resource** which has been adapted to Sling model. * @**Inject** annotation inject the properties from AEM node to Sling properties. * @Required - Mark the property as required * @Default – Assign a default value to the sling model property if no value is set in AEM node.   Note: This value will appear while authoring as well – when the component is dropped for the first time on the page. |
| **HTML CODE: ACCESSING SLING MODEL PROPERTIES**  <div **data-sly-use**.**employee**="com.aem.geeks.core.models.Employee"></div> <p>**${**employee.firstName**}**</p> <p>**${**employee.lastName**}**</p> <p>**${**employee.IsPermanent**}**</p> | |

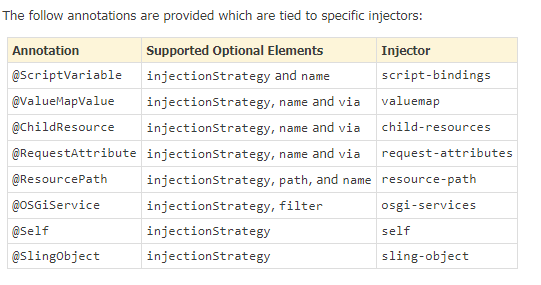
##### POM.XML

|  |  |
| --- | --- |
| Add dependency in bundle pom.xml  <dependency>  <groupId>org.apache.sling</groupId>  <artifactId>org.apache.sling.models.api</artifactId>  <version>1.3.2</version>  <scope>provided</scope>  </dependency>  Note : **Sling-Model-Packages : This contain the package name where all the Sling Models are created in the bundle.** | <plugin>  <groupId>org.apache.felix</groupId>  <artifactId>maven-bundle-plugin</artifactId>  <extensions>true</extensions>  <configuration>  <instructions>  <Bundle-SymbolicName>com.adobe.training.company-training-project-bundle</Bundle-SymbolicName>  **<Sling-Model-Packages>**  **com.adobe.training.models**  **</Sling-Model-Packages>**  </instructions>  </configuration>  </plugin> |

#### SLING MODEL ANNOTATIONS

##### INJECTOR SPECIFIC ANNOTATION

* @Inject is a generic injector in **javax.inject.Inject** package. We can use this annotation anywhere, but @Inject queries all Injector implementations “First Match” wins based on the service ranking.
* Rather than using a generic injector we can use specific injector annotation as shown below.
* Note almost all the injector annotation has property - **injectionStrategy**: The Strategy can be Optional, Required or Default.



###### **@ChildResouce**

|  |  |
| --- | --- |
| **TYPE 1: ADAPTABLE TO RESOURCE** | |
| @Model(adaptables = Resource.**class**)  **public** **class** TestSlingModel {  @ChildResource(name="content")  Resource childResource;  **private** String quotes;  **public** String getQuotes() {  Resource childResource = **this**.childResource.getChild("title");  **if**(**null** != childResource){  quotes = childResource.getValueMap().get("jcr:title","");  }  **return** quotes;  } | |
| Content Structure | <sly data-sly-use.model="com.aem.community.core.models.TestSlingModel">  <sly data-sly-test="${model.quotes}">  ${model.quotes}  </sly>  </sly> |
| **TYPE 2: ADAPTABLE TO REQUEST:** If the sling model is adabtable to SlingHTTPServletRequest, then the child resource must be injected via “resource” | |
| @Model(adaptables = SlingHttpServletRequest.**class**)  **public** **class** TestSlingModel {  **ChildResource(name="content",via="resource")**  Resource childResource;  **private** String quotes;  **public** String getQuotes() {  Resource childResource = **this**.childResource.getChild("title");  **if**(**null** != childResource){  quotes = childResource.getValueMap().get("jcr:title","");  }  **return** quotes;  } | |

###### **@ResourcePath**

|  |  |
| --- | --- |
| @Model(adaptables = Resource.class )  public class TestModel {  //directly inject a path as a resource  @ResourcePath(path = ”/etc/social”)  Resource pathResource;  @ResourcePath(name = "path")  Resource resourcePath;  @ResourcePath(paths = {"/etc/social","/etc/tags"})  Resource[] paths;  } | * This inject a path as a resource * If a resource is having a property whose value is a path, you can directly use that property as a resource. * The attributes of the ResourcePath annotation are: * name * injectionStrategy * path * paths[] |
| resourcePath.PNG | |

###### **@ScriptVariable**

|  |  |
| --- | --- |
| @Model(adaptables = SlingHttpServletRequest.**class**)  **public** **class** TestInjectAnnotation {  @ScriptVariable  Page currentPage;  String pageName;  **public** String getPageName() {  pageName = currentPage.getName();  **return** pageName;  }  } | * This Injector is used to inject the **currentPage, PageManager, Design, PageProperties, Node, Resource** etc. * This injector is adaptable to request.This injector is adaptable to SlingHttpServletRequest. * ScriptVariable annotation has these attributes:   1. name   2. injectionStrategy |

###### **@SlingObject**

|  |  |
| --- | --- |
| @Model(adaptables = SlingHttpServletRequest.**class**)  **public** **class** TestInjectAnnotation {  @SlingObject  Resource resource;  String resourceName;  **public** String getResourceName() {  resourceName = resource.getName();  **return** resourceName;  }  } | Supports sling-based Objects **like request, response, ResourceResolver, Resource and Sling ScriptHelper**.  **HTL**  <sly data-sly-use.testInjectAnnotation="com.aem.community.core.models.TestInjectAnnotation">  ${testInjectAnnotation.resourceName}  </sly> |

###### **@OSGIServices**

|  |  |
| --- | --- |
| @Model(adaptables = SlingHttpServletRequest.**class**)  **public** **class** TestInjectAnnotation {  @OSGiService  SlingSettingsService settingsService;    String runMode;  **public** String getRunMode() {  runMode = settingsService.getRunModes().toString();  **return** runMode;  }  } | **Inject OSGi Service in Sling Models**  **HTL**  <sly data-sly-use.testInjectAnnotation="com.aem.community.core.models.TestInjectAnnotation">  ${testInjectAnnotation.runMode}  </sly> |

###### **@Self**

|  |  |
| --- | --- |
| @Model(adaptables = Resource.class,  adapters = ResourceInterf.class,  defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL)  public class ResourceModel implements ResourceInterf {  @Self  Resource resource;  @Override  public String getResourcePath() {  return resource.getPath();  }  } | * To inject the object itself in the property of the Sling . * In the example – Resource has been injected – although the Sling Sling Model is adabtable to Resource. |

###### **@ValueMapValue**

|  |  |  |
| --- | --- | --- |
| **SLING MODEL: ADABTABLE TO RESOURCE**  @Model(adaptables = Resource.**class**)  **public** **class** TestInjectAnnotation {  @ValueMapValue  String firstName;  @PostConstruct  **public** String getFirstName() {  **return** firstName;  }  } | **SLING MODEL: ADABTABLE TO REQUEST**  @Model(adaptables = SlingHttpServletRequest.**class**)  **public** **class** TestInjectAnnotation {    @ValueMapValue(via="resource)  String firstName;  @PostConstruct  **public** String getFirstName() {  **return** firstName;  }  } | |
| **HTL**  <sly data-sly-use.testInjectAnnotation="com.aem.community.core.models.TestInjectAnnotation">  <div data-sly-test="${testInjectAnnotation.firstName}">  ${testInjectAnnotation.firstName}  </div> | | |
| **Content Structure** | | The attributes of the ValueMapValue is:   * name * injectionStrategy * via |

##### **OTHER SLING MODEL ANNOTATION**

###### **@PostConstruct**

The method maked @PostConstruct annotation are invoked after completion of all injections(@Inject)

|  |  |
| --- | --- |
| **Sling Model** | **HTL Code** |
| @Model(adaptables = Resource.**class**)  **public** **class** TestSlingModel {  @Inject  **private** String firstName;  @Inject  **private** String lastName;  @PostConstruct  **public** String getFullName() {  **return** **this**.firstName + " " + **this**.lastName;  }  **public** String getFirstName() {  **return** firstName;  }  **public** String getLastName() {  **return** lastName;  }  } | <sly data-sly-use.model="com.aem.community.core.models.TestSlingModel">  <div data-sly-test="${model.firstName && model.lastName}">  ${model.fullName}  </div>  </sly>  <div data-sly-test="${!(model.firstName && model.lastName)}">  Enter your Name  </div> |

###### **@Named**

|  |  |
| --- | --- |
| @Model(adaptables = Resource.**class**)  **public** **class** TestInjectAnnotation {  @Inject @Named("sling:resourceType")  String slingResourceType;  **public** String getSlingResourceType() {  **return** slingResourceType;  }  } | If there is a need to change the getter of any attribute like (sling:resourceType, jcr:primaryType) @Named annotation helps to achieve this.  **HTL**  <sly data-sly-use.testInjectAnnotation="com.aem.community.core.models.TestInjectAnnotation">  ${testInjectAnnotation.slingResourceType}  </sly> |

###### **@Via**

|  |  |
| --- | --- |
| @Model(adaptables = SlingHttpServletRequest.**class**)  **public** **class** TestInjectAnnotation {    @ResourcePath(path="/content/AEMMaven12/en/jcr:content")  @Via("resource")  Resource resource;    @SlingObject  ResourceResolver resourceResolver;  String pageTitle;  **public** String getPageTitle() {  Resource resource = **this**.resourceResolver.getResource(**this**.resource.getPath());  pageTitle = resource.getValueMap().get("jcr:title","");  **return** pageTitle;  }  } | SlingHttpServletRequest has more objects than resource. Sometimes there is a need of using two injectors one from request and one from resource, And then we need to tell annotation explicitly that we are coming via resource. |

###### **@Default**

|  |  |
| --- | --- |
| @Model(adaptables = Resource.**class**)  **public** **class** TestInjectAnnotation {  @Inject @Default(values="No Name")  String firstName;  **public** String getFirstName() {  **return** firstName;  }  } | * A default value can be provided for Strings or primitive data types. If there is no value of that property, default value takes place. * When the component is dropped on the page it will show the default content before authoring as the property value is not yet provided |

###### **@Optional and @Required**

|  |  |
| --- | --- |
| @Model(adaptables = Resource.**class**)  **public** **class** TestInjectAnnotation {  @Inject @Optional  String path;  @Inject @Required  String title;  **public** String getTitle() {  **return** title;  }  **public** String getPath() {  **return** path;  }  } | * @Optional and @Required: In the sling models, by default all the fields supposed to be required.Sometimes there is a need to mark them as optional and required specifically.So injector fields can be annotated with @Optional and @Required. * If a majority of @Injected fields/methods are optional, it is possible to change the default injection strategy by using adding defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL to the @Model annotation on a class level |

#### PASSING VALUE TO SLING MODEL

###### **@RequestAttribute**

|  |  |
| --- | --- |
| @Model(adaptables = SlingHttpServletRequest.**class**)  **public** **class** TestInjectAnnotation {    **@RequestAttribute(name = "name")**  String nameParam;  **public** String getNameParam() {  **return** "Hello "+nameParam+ " !";  }  } | **PASSING PARAMETER FROM SLING MODEL**  The attributes of RequestAttribute annotation are:   * 1. name   2. injectionStrategy |
| **HTL**  <sly data-sly-use.testInjectAnnotation="${'com.aem.community.core.models.TestInjectAnnotation'  **@name='Amit'**}">  ${testInjectAnnotation.nameParam}  </sly> | |

#### MULTIFIELD IN SLING MODEL

### INITIALIZING WCM

|  |  |  |
| --- | --- | --- |
| **CQ APIs** | currentPage.getTitle() | |
| **Sling Api** | properties.get("jcr:title") | |
| **JCR API** | currentNode.getProperty("jcr:title").getString() | |
| CQ and Sling Api are written on top of JCR API. But the preference is CQ Api 🡪Sling Api🡪 JCR Api | | |
| **JSP INCLUDE –** Included at compile time | | <%@ include file="myScript.jsp" %> |
| **CQ INCLUDE –** Included at runtime | | <cq:include script="myScript.jsp" /> |
| **SLING INCLUDE** | | <sling:include path="layout-link.jsp" /> |
| **INCLUDING GLOBAL.JSP** | | < [%@include file="/libs/foundation/global.jsp"%](mailto:%25@include%20file=%22/libs/foundation/global.jsp%22%25) >  Can able to use JSTL ,CQ tags ,Apache Sling taglibs  JSP+ CQ implicit objects - **<cq:defineObjects />** |

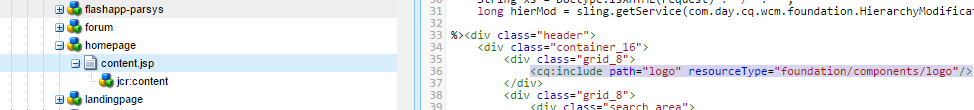
### DIALOG & DESIGN DIALOGS

#### DESIGN DIALOGS

* Used to store template level properties, which can be common across the website.e.g. Logos
* Design dialog are visible in Design Mode only

|  |  |
| --- | --- |
| CREATING A DIALOG | Create dialog 🡪 name must be **design\_dialog** |
| RETRIVING VALUE | **<%=currentStyle.get("siteurl")%>** |

EXAMPLE: Let us consider a component “**homepage**” having a component included “**logo**”.



|  |  |
| --- | --- |
|  | The logo component has a design dialog which will be visible in the design mode of the homepage |
|  | |
| To save the values of the design dialog to a desired location , the template has a property - **cq:designPath : /etc/designs/geometrixx**”. It means the value of design dialog will be saved at **/etc/designs/geometrixx** under the component node i.e homepage (name of the template). The property saved under logo node are the values feed in the design dialog of homepage | |
|  | |
| **Question**: How values are stored in “par” in /etc/design/homepage – shown above?  **Answer**: Suppose the homepage component has a parsys . in that parsys component can be droped(like section). Now if that compoent has a design dialog , the values will be stored – homepage[component name]/par/section | |
| **EXTENDING A DIALOG**    STEP 1: Create a dialog of the custom component  Step 2: Create a widget and add below proper   * path : /libs/foundation/components/page/tab\_basic.infinity.json * xtype : cqinclude | |

#### CREATING TOUCH UI DIALOG

All OOB widgets that will be used to create Touch UI dialog are in- **/libs/granite/ui/components/foundation/<sub\_folders>**

|  |  |
| --- | --- |
| **CLASSIC UI DIALOG** | **TOUCH UI DIALOG** |
| Based on EXTJs | Based on Granite.js |
| Not Responsive | Responsive |
| Root Node is - dialog | Root node is – cq:dialog |
| For fields in classic ui dialog xtypes properties are used | For fields in classic ui dialog sling ResourceType properties are used |
| **cq:dialog –[** sling:resourceType: **cq/gui/components/authoring/dialog]** | |
| **content [sling:resourceType = granite/ui/components/coral/foundation/container]** | |
| **layout[sling:resourceType = granite/ui/components/coral/foundation/fixedcolumns]** | |
| **Items** | |
| **Columns** | |
| **Items** | |
| **Name(field)** | |

##### VALIDATION IN TOUCH UI DIALOGS

**LOADING A CUSTOM JAVASCRIPT FOR A DIALOG / CUSTOM VALIDATION IN TOUCH UI DIALOG**

We can load custom library for a touch-ui dialogs (which can be later used for custom validation of dialog fields ) .We can load the custom libabries using 2 ways

* 1. ***ADDING JS VIA CQ.AUTHORING.DIALOG***
  2. ***ADDING JS USING INCLUDECLINETLIBS***

**ADDING JS VIA CQ.AUTHORING.DIALOG**

* To write a custom validation which we will be applied to the entire dialog (even OOTB AEM dialog) we use this way
* For this we create a client library with a category name “**cq.authoring.dialog**”, so when the page loads in AEM in author mode all the client library named as “**cq.authoring.dialog**” are clubbed together and loads on Page (all.js).

**All.js will load when the page loads in author environment.**



* Since all.js is loaded upfont at page load, the custom validation is applicable for all the dialogs

|  |  |
| --- | --- |
| * Create a client library which a category name “**cq.authoring.dialog**”. * Write the custom validation in the JS of the client library(dialog-validation.js) |  |

**ADDING JS USING INCLUDECLINETLIBS**

* This way is used to write the custom validation for a specific dialog . To accomplish this follow the below steps

|  |  |
| --- | --- |
| Create a client library which which has the JS files having custom validation in it.  For example  Categories= **cq.include** |  |
| Create a nt:unstructured node in the under “items” . example with node name = **include-clientLib**. Set the below properties on that node   * 1. sling:resourceType = granite/ui/components/coral/foundation/includeclientlibs   2. js = **cq.include (**Category name of the library where the custom validation resides**)**   **Note : This library will load when we open the dialog of corresponding component is loaded not globally as in cq.authiring.dialog** | |
|  | |

**CUSTOM VALIDATION USING GRANITE UI**

* To execute the custom validation logic in the JS files we need to load the JS file using either of the above methods.
* Previously for form validation JQuery validator is used which has been deprecated and “foundation-validation-validator”is recommended

|  |
| --- |
| **SAMPLE CUSTOM VALIDATION CODE [foundation-validation-validator]** |
| $(window).adaptTo("foundation-registry").register("foundation.validation.validator", {  selector: "[data-should-contain]",  validate: function(el) {  var shouldContain = el.getAttribute("data-should-contain"); //aem  console.log('validating text contains aem');  console.log('input should contain ' + shouldContain);  var input = el.value; //input added by author  if (input.indexOf(shouldContain) === -1 ) {  return "The field should contain " + shouldContain + ". It's current value is " + el.value + ".";  }  }  });   * Add the above code in the JS file for custom validation |
| **HOW TO APPLY THE CUSTOM VALIDATION ON A DIALOG FIELD** |
| **BINDING THE CUSTOM VALIDATION WITH A DIALOG FIELD**   * Create a nt:unstructured node named **granite:data** * Add a property to it - **should-contain [**referringselector attribute in the JS file**]**     **DIALOG** |

### CLIENTLIBRARY

|  |  |
| --- | --- |
| By default, cq:ClientLibraryFolder nodes can be placed anywhere within the /apps, /libs and /etc subtrees of the repository (these defaults, and other settings can be controlled through the **Adobe Granite HTML Library** |  |

|  |  |
| --- | --- |
| **Categories** | * Category name of client librarys The categories property, * It’s a multi-valued hence allows a library folder to be part of more than one category(Refer above) |
| **dependencies** | This is a list of other client library categories on which this library folder depends. For example, given two cq:ClientLibraryFolder nodes F and G, if a file in F requires another file in G in order to function properly, then at least one of the categories of G should be among the dependencies of F. |
| **Embed** | You can embed code from a client library into another client library. At runtime, the generated JS and CSS files of the embedding library includes the code of the embedded library. |
| **Channel** | Refer Below |
| **allowProxy** | In previous versions, client library folders were located below /etc/clientlibs in the repository. This is still supported; however **it is recommended that client libraries now be located under /apps**. This is to locate the client libraries near the other scripts, which are generally found below /apps and /libs.  For the client libraries under /apps to be accessible, a **proxy servelt** is used. The ACLs are still enforced on the client library folder, **but the servlet allows for the content to be read via /etc.clientlibs/ if the allowProxy property is set to true**.  A static resource can only be accessed via the proxy if it resides below a resource below the client library folder.  **EXAMPLE**   * You have a clientlib in /apps/myproject/clientlibs/foo or a static image in /apps/myprojects/clientlibs/foo/resources/icon.png * Then you set the allowProxy property on foo to true. * You can then request **/etc.clientlibs/myprojects/clientlibs/foo.js** or can reference the image via **/etc.clientlibs/myprojects/clientlibs/foo/resources/icon.png** |
| **EXAMPLE**  Clientlib path before **allowProxy / allowProxy= false**    Clientlib path after **allowProxy= true** | |

### CLIENT-LIBRARY PROPERTIES

|  |  |  |
| --- | --- | --- |
|  | * Create folders css and js below clientlib folder * Create the required js and css files in js and css folders * Create two txt files css.txt and js.txt in clientlib folder ; * Enter the file names in js.txt (all js file ) and css.txt(all css file); one entry per line; Example : css/default.css;js/default.js or use #base=[root]   Replace [root] with the path to the folder that contains the source files, relative to the TXT file. For example, use the following text when the source files are in the same folder as the TXT file:  #base=.  The following code sets the root as the folder named mobile below the cq:ClientLibraryFolder node:  #base=mobile  On the lines below #base=[root], type the paths of the source files relative to the root. Place each file name on a separate line. | |
| **IMPORTANT POINTS**   * Client library folders located below /apps take precedence over same-named folders that are similarly located in /libs. For example, /apps/cq/ui/widgets takes precedence over /libs/cq/ui/widgets. When these libraries belong to the same category, the library below /apps is used.This is called Overlaying. * If the name of the Client library of folder **ClientLibA.** Then the name of the minified JS and CSS file will be [can be seen in Firebug Console] will be **ClientLibA.js** and **ClientLibA.css** | | |
| Below are three client library named as dependency, embedded &  main under clientlibs folder.  http://2.bp.blogspot.com/-HkmxyMuHP3E/VHwzIvMOkgI/AAAAAAAAA6s/E8cAaS39yzA/s1600/image_1.1.png | | **Property for main.client client library**  **http://4.bp.blogspot.com/-_uJzID_DwLA/VHw6XCgyttI/AAAAAAAAA7E/_7IGNO0M0KI/s1600/main_1.png**  **Property for dependency client library**  **http://1.bp.blogspot.com/-IJu5ISqi5kU/VHw5xV_4oEI/AAAAAAAAA68/Nkg3GLMr8_I/s1600/dependency_1.png**  **property for embedded client library**  **http://3.bp.blogspot.com/-Bpt3qIhyeG4/VHw6qWiVByI/AAAAAAAAA7U/pk9vVuBLWfg/s1600/embedded_1.png** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Lets consider we have 2 client libraries – ClientA(js-a.js) and ClientB(js- b.js)   |  |  | | --- | --- | | Client A : Js.txt  #base=js  a.js | Client B : Js.txt  #base=js  b.js |   If we add the **dependencies** property, For every dependency property value there is an individual hit to the server for loading these categories i.e. if you have four values in this property then 4 hits will be shown at debugging console network tab.    When we add embed property. For all embed property value there is only one hit to the server for loading these categories i.e. if you have four values in this property then only one combined hits will be shown at debugging console network tab.   |  |  | | --- | --- | |  |  | |

**CHANNEL PROPERTY**

|  |  |
| --- | --- |
| http://3.bp.blogspot.com/-HrCVLC2SnlU/VIBPRCl86FI/AAAAAAAAA7s/F7xr-xiOnoo/s1600/channels_1.png | * It is multivalued property. * It is used to some additional functionality for different channels. For example : if you want to perform some JS functionality only for ie6 not for other browsers, or you want to add some CSS or JS for only touch UI then this channel property comes into picture. |

**USE CASE FOR CHANNEL PROPERTY:** If you want to achieve some functionality for touch only devices then just create two client libraries with same category name & in first client lib add common code for all channels & in second client lib use this property channels and set it's value as touch. So that for touch devices both of these client libs are loaded and you will get your desired functionality. For non touch devices only first client lib is loaded as second have this channel property it will not be available here.

#### USING THE CLIENR LIBRARY

|  |  |
| --- | --- |
| **JSP** | <cq:includeClientLib css="td.campaign" /> |
| **HTML** | **BOTH JS AND CSS**  sly data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html"  data-sly-call="${clientlib.all @ categories=['my-clientlib-category']}"/>  **ONLY CSS**  <sly data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html"  data-sly-call="${clientlib.css @ categories=['my-clientlib-category']}"/>  **ONLY JS**  <sly data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html"  data-sly-call="${clientlib.js @ categories=['my-clientlib-category']}"/> |

**ADVANTAGE OF CLIENT LIBRARY**

|  |  |
| --- | --- |
| At its basic level, the Client Library allows a developer to keep all of his or her stylesheets and JavaScript libraries organized in project folders in the CRX repository. **On the front end, all these files are compiled into a single location referenced on the page**. **The browser only makes a single request, which cuts down on a lot of HTTP chatter**. |  |

|  |  |
| --- | --- |
| **URL** | **Description** |
| <http://localhost:4502/libs/granite/ui/content/dumplibs.html> | To view all the clientlibs, its dependencies and embed clientLibraries. |
| <http://localhost:4502/libs/granite/ui/content/dumplibs.test.html> | If you want to know about a particular clientlibs,which css and js files are getting loaded, add a selector "test" |
| <http://localhost:4502/libs/granite/ui/content/dumplibs.rebuild.html> | **There is always a cache issue with clientlibs, when you make any change in files (CSS/JS),the changes doesn’t reflect on pages,because AEM cache the clientlibs under “/var/clientlibs”. If you want to rebuild the clientlibs or clear cache** |
| <http://localhost:4502/libs/granite/ui/content/dumplibs.validate.html> | Shows the table of all the clientlibs, its dependencies and embed client Libraries and validates it (whether the embed and dependencies will exist or not,or they exist for a particular category type i.e., css/js or not)  with different color codes. |

### OSGI

OSGI Definition here

#### OSGI BUNDLE

**WHAT’S AN OSGI BUNDLE?**

* A bundle is a deployment unit in OSGi. They are jar files (having class files and the related resources) with some headers (metadata) in the MANIFEST.MF file.

|  |  |
| --- | --- |
|  | **Q: How can we identify whether a JAR file is an OSGi Bundle?**  A: Any JAR file is a bundle If it’s a header “**Bundle-SymbolicName**” in it MANIFEST.MF file.   * It’s a unique identifier of the bundle and a formal name of the bundle. * No two bundles can have same symbolic name. * Bundle-SymbolicName should not contain space   **What is bundle version?**   * Each bundle is associated with a bundle version. It is identified by Bundle Version header in MANIFEST file * Bundle Version Format : **MAJOR.MINOR.MICRO.QUALIFIER e.g. 3.4.1*.<buildNumber OR Date>*** * The OSGi Container uses the combination of Bundle Symbolic Name and Bundle version header to determine uniqueness of the bundle. |

**WHO WRITES THE BUNDLE HEADERS?**

* **Bnd** tool is used to generate the Bundle headers. The instruction to generate the header are given in a **bnd.bnd** file

|  |  |
| --- | --- |
|  | **STEPS TO CREATE BUNDLES**   1. Create a src folder in apps folder 2. Right click on src🡪 Create bundle 🡪Creates a .bnd file 3. **It contains the META INFORMATION of the Project.**   Reference : [**http://www.hsufengko.com/home/develop-osgi-bundle-using-adoble-cq-crxde-lite**](http://www.hsufengko.com/home/develop-osgi-bundle-using-adoble-cq-crxde-lite) |

**BND FILE**

* # Export-Package: \* 🡪 package mentioned here (java classes) – are public[visible for other bundles]
* # Import-Package: \* 🡪 mention the package which we want to import in our resource bundle
* Private-Package: com.sapient.training
* # Include-Resource: 🡨 Resource Files mentioned here
* Bundle-Name: Training Module Bundle
* Bundle-Description: To implement the training module funcatioanlity
* Bundle-SymbolicName: com.adobe.training
* Bundle-Version: 1.0.0-**SNAPSHOT 🡨 SNAPSHOT** means it is in Dev State
* Bundle-Activator: com.sapient.training.Activator🡨It’s a kind of listener which listens the start and stop of bundle using below life cycle methods

**public void start(BundleContext context) throws Exception**

**public void stop(BundleContext context) throws Exception**

**TO BUILD THE BUNDLE**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**DIFFERENCE BETWEEN OSGI BUNDLE AND NORMAL JAR FILE?**

* With OSGi, just because a class is public doesn’t mean you can get to it. All bundles include an export list of package names, and if a package isn’t in the export list, it doesn’t exist to the outside world. This allows developers to build an extensive internal class hierarchy and minimize the surface area of the bundle’s API without abusing the notion of package-private visibility. A common pattern, for instance, is to put interfaces in one package and implementations in another, and only export the interface package.
* All OSGi bundles are given a version number, so it’s possible for an application to simultaneously access different versions of the same bundle (eg: junit 3.8.1 and junit 4.0.). Since each bundle has its own class-loader, both bundles classes can coexist in the same JVM.
* OSGi bundles declare which other bundles they depend upon. This allows them to ensure that any dependencies are met before the bundle is resolved. Only resolved bundles can be activated. Because bundles have versions, versioning can be included in the dependency specification, so one bundle can depend on version junit version 3.8.1 and another bundle depend on junit version 4.0.
* In OSGi bundle, there will be an Activator.java class in OSGi which is an optional listener class to be notified of bundle start and stop events.

**BENEFIT OF OSGI BUNDLE OVER JAR FILE?**

Bundles includes metadata such as the version and list of services imported and exported by the bundle.

1. OSGi bundle can be installed, updated, and uninstalled without taking down the entire application.
2. OSGi bundling allows multiple versions to exist.
3. OSGi (Open Services Gateway Initiative) defines an architecture for developing and deploying modular applications and libraries.

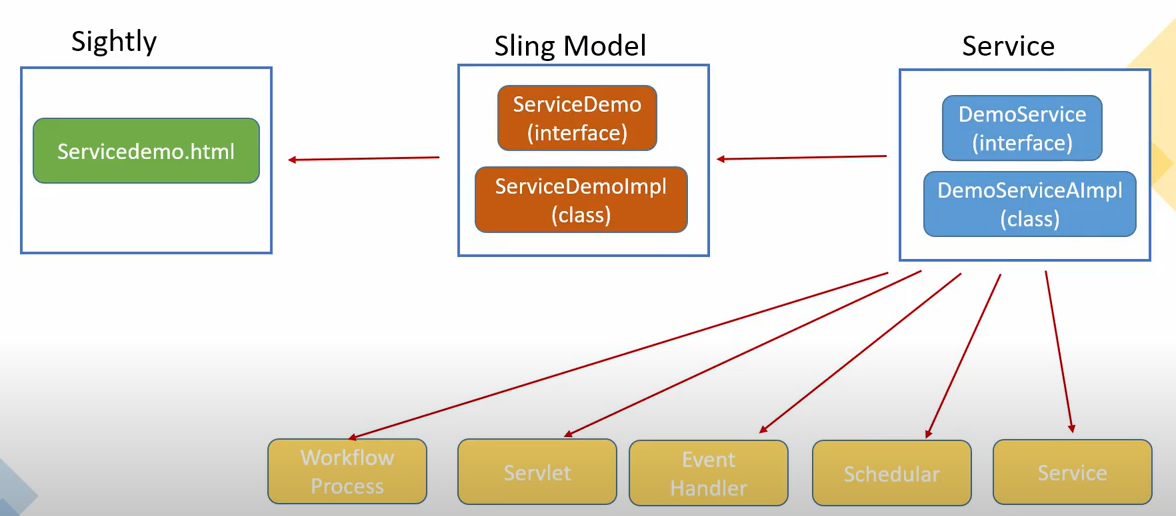
#### OSGI SERVICES

**Definition here**

##### CREATING AN OSGI SERVICE

* Step 1: Create an interfece
* Step 2: Create an implementation class and annotate it with @Component annotation

**Service can be called be called in another service, Sling Models, Servlets, Workflow process, Scheduler and Event Handlers**



##### CALLING AN OSGI SERVICE IN SLING MODEL

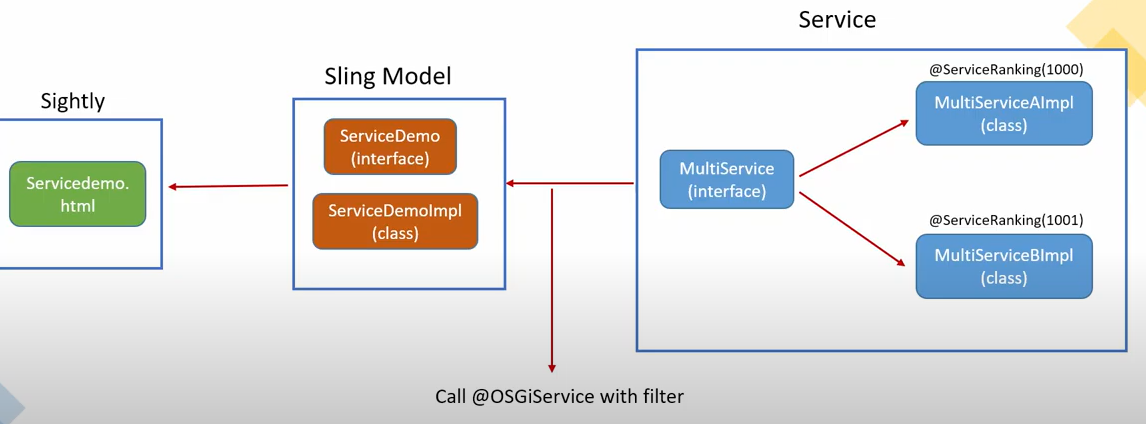
* We use **@OSGiService** annotation to call a OSGI service in a sling model

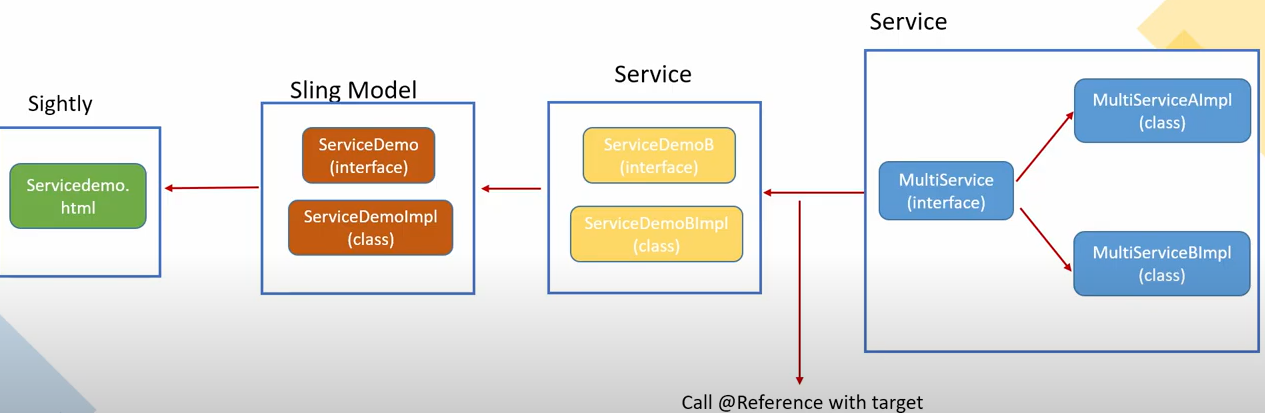
##### CALLING AN OSGI SERVICE IN ANOTHER SERVICE

* We use **@Reference** annotation to call a OSGI service in another service

##### SERVICE RANKING AND TARGETS

* In case of service - When we have multiple implementation of the same interface – We resolve the service us





### OSGI CONFIGURATION – USING R7 ANNOTATIONS

OSGI Configuration helps in passing the dynamic value in the Java Class like components etc.

#### CONFIGURING OSGI COMPONENTS USING SCR ANNOTATION

You can manage the configuration settings for such bundles by either:

1. using the Adobe CQ Web console
2. configuring content-nodes (sling:OsgiConfig) in the repository

Either method can be used though there are subtle differences, primarily in relation to Run Modes:

1. **Adobe CQ Web console**

* The Web Console is the standard interface for OSGi configuration. It provides a UI for editing the various properties, where possible values can be selected from predefined lists.
* Any configurations made with the Web Console are applied immediately and applicable to the current instance, irrespective of the current run mode, or any subsequent changes to the run mode.

1. **Content-nodes (sling:osgiConfig) in the repository**

* This requires manual configuration using CRXDE Lite.
* Due to the naming conventions of the **sling:OsgiConfig** nodes, you can tie the configuration to a specific run mode. You can even save configurations for more than one run mode in the same repository. Any appropriate configurations are applied immediately (dependent on the run mode).

#### DC- CREATING OSGI CONFIGURATION

|  |  |
| --- | --- |
| **INTERFACE THAT HOLDS THE CONFIG**  import org.apache.commons.lang3.StringUtils;  import org.osgi.service.metatype.annotations.AttributeDefinition;  import org.osgi.service.metatype.annotations.AttributeType;  import org.osgi.service.metatype.annotations.ObjectClassDefinition;  import org.osgi.service.metatype.annotations.Option;  @ObjectClassDefinition(name = "Annotation Demo Service - OSGi")  public @interface Configuration {  @AttributeDefinition(  name = "Boolean Property",  description = "Sample boolean value",  type = AttributeType.BOOLEAN  )  boolean servicename\_propertyname\_boolean() default true;  @AttributeDefinition(  name = "String Property",  description = "Sample String property",  type = AttributeType.STRING  )  String servicename\_propertyname\_string() default "foo";  @AttributeDefinition(  name = "Dropdown property",  description = "Sample dropdown property",  options = {  @Option(label = "DAYS", value = "DAYS"),  @Option(label = "HOURS", value = "HOURS"),  @Option(label = "MILLISECONDS", value = "MILLISECONDS"),  @Option(label = "MINUTES", value = "MINUTES"),  @Option(label = "SECONDS", value = "SECONDS")  }  )  String servicename\_propertyname\_dropdown() default StringUtils.EMPTY;  @AttributeDefinition(  name = "String Array Property",  description = "Sample String array property",  type = AttributeType.STRING  )  String[] servicename\_propertyname\_string\_array() default {"foo", "bar"};  /\*  \* To create password field, either set the AttributeType or have the  \* property name end with "\*.password" (or both).  \*/  @AttributeDefinition(  name = "Password Property",  description = "Sample password property",  type = AttributeType.PASSWORD  )  String servicename\_propertyname\_password() default StringUtils.EMPTY;  @AttributeDefinition(  name = "Long Property",  description = "Sample long property",  type = AttributeType.LONG  )  long servicename\_propertyname\_long() default 0L;  } | **INTERFACE**  public interface SampleOsgiService {  String getSettings();  }  **IMPL**  import org.apache.commons.lang3.ArrayUtils;  import org.apache.sling.api.resource.ResourceResolverFactory;  import org.osgi.service.component.annotations.Activate;  import org.osgi.service.component.annotations.Component;  import org.osgi.service.component.annotations.Deactivate;  import org.osgi.service.component.annotations.Modified;  import org.osgi.service.component.annotations.Reference;  import org.osgi.service.metatype.annotations.Designate;  import com.nateyolles.aem.osgiannotationdemo.core.services.SampleOsgiService;  @Component(  immediate = true,  service = SampleOsgiService.class,  configurationPid = "com.nateyolles.aem.osgiannotationdemo.core.services.impl.SampleOsgiServiceImpl"  )  @Designate( ocd = Configuration.class)  public class SampleOsgiServiceImpl implements SampleOsgiService {  @Reference  private ResourceResolverFactory resolverFactory;  boolean booleanProp;  String stringProp;  String dropdownProp;  String[] stringArrayProp;  char[] passwordProp;  long longProp;  @Override  public String getSettings() {  StringBuilder sb = new StringBuilder();  sb.append("Sample OSGi Service:\n");  sb.append("booleanProp: " + booleanProp + "\n");  sb.append("stringProp: " + stringProp + "\n");  sb.append("dropdownProp: " + dropdownProp + "\n");  sb.append("stringArrayProp: " + ArrayUtils.toString(stringArrayProp) + "\n");  sb.append("passwordProp: " + String.valueOf(passwordProp) + "\n");  sb.append("longProp: " + longProp + "\n");  return sb.toString();  }  @Activate  @Modified  protected final void activate(Configuration config) {  booleanProp = config.servicename\_propertyname\_boolean();  stringProp = config.servicename\_propertyname\_string();  dropdownProp = config.servicename\_propertyname\_dropdown();  stringArrayProp = config.servicename\_propertyname\_string\_array();  passwordProp = config.servicename\_propertyname\_password().toCharArray();  longProp = config.servicename\_propertyname\_long();  }  @Deactivate  protected void deactivate() {  }  } |

#### OSGI CONFIGURATION USING ADOBE CQ WEB CONSOLE

***Step 1: Create a Java class (Service /Component). This used Felix SCR annotationn***

@Service

@Component(name = "HelperServiceImpl", immediate = **true**, label = "Helper Service Configuration", description = "Helper Service Configuration", metatype = **true**)

**public** **class** HelperServiceImpl **implements** HelperService {

@Property(name = "myMessage", value = "Enter the message", description = "Message Configuration")

**private** String message;

Map<String, String> myPropertyMap = **new** HashMap<String, String>();;

@Activate

**protected** **void** activate(ComponentContext componentContext){

myPropertyMap.put("myMessage",PropertiesUtil.*toString*(componentContext.getProperties().get("myMessage"), "myMessage"));

}

**public** String sayHello() {

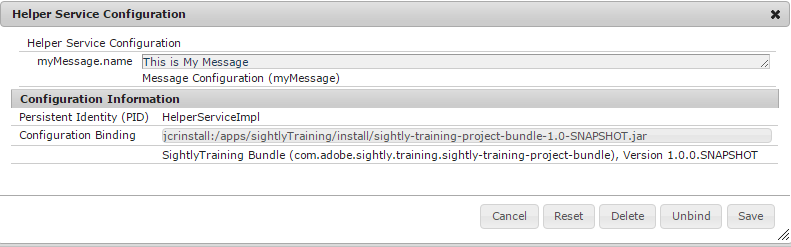
String message= **null**;

String configuredMessage= **null**;

message = "Helper Service Return Statement"+ myPropertyMap.get("myMessage").toString();

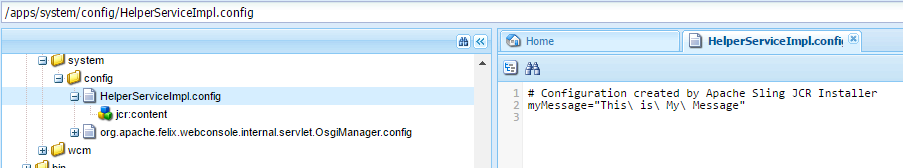
**return** message; }}

***Step 2: Configure the Property of the Service/Component from Web console [*** [***http://localhost:4502/system/console/configMgr***](http://localhost:4502/system/console/configMgr) ***]***



***Step 3: Question- Where the configuration value get stored in CRX when it is configured from WebConsole***

***Answer –*** Location - **/apps/system/config**



#### OSGI CONFIGURATION USING (SLING:OSGICONFIG) IN THE REPOSITORY

|  |  |
| --- | --- |
| ***Step 1 : Create an osgiConfig node in /apps/<project\_name>/config*** |  |
| ***Step 2: The configuration can be done at the property level*** | |
|  | |

***Step 3: How to retrieve the configuration value for the osgiConfig node?***

@Service

@Component(name = "HelperServiceImpl", immediate = **true**, label = "Helper Service Configuration", description = "Helper Service Configuration", metatype = **true**)

**public** **class** HelperServiceImpl **implements** HelperService {

@Reference

ConfigurationAdmin configurationAdmin;

**public** String sayHello() {

String configuredMessage = configurationAdmin.getConfiguration("com.adobe.sightly.training.impl.HelperServiceImpl").getProperties().

get("myMessage").toString();

}

**return** configuredMessage;

}

}

#### DIFFERENT WAYS OF OSGI CONFIGURATION CONFIGURED FROM WEB CONSOLE

Note : The OSGI configuration can set using various input fields like Multifield, checkboxes and drop down etc. Refer the below example for detail. This uses a Java code to create OSGI configuration.

**STEP 1: CREATE AN INTERFACE**

public interface MyOSGIConfig {

public Map<String, String> getAllConfiguration() throws IOException;

}

**STEP 2: CREATE AN IMPLEMENTATION CLASS**

@Component(name = "MyOSGIConfigConfig", immediate = true, label = "My OSGIConfig Config", description = "My OSGIConfig Config", metatype = true)

@Service

public class MyOSGIConfigImpl implements MyOSGIConfig {

@Property(name = "studentName", value = "Enter student Name", description = "This is Configuration for Student")

private String studentName;

@Property(name = "isSenior", label = "is Senior Secondary Student", boolValue = false, description = "please Check the Property")

private String isSenior;

@Property(name = "subjects", value = { "English", "Hindi" }, unbounded = PropertyUnbounded.*ARRAY*, label = "Subjects", cardinality = 50, description = "Example for Multi field config")

private String[] subjects;

@Property(name = "Gender", description = "Select Gender", options = {

@PropertyOption(value = "Male", name = "1.Male"),

@PropertyOption(value = "Female", name = "2.Female") }, value = "Male")

private String gender;

Map<String, String> myPropertyMap = new HashMap<String, String>();;

public Map<String, String> getAllConfiguration() throws IOException {

return myPropertyMap;

}

@Activate

protected void activate(ComponentContext componentContext){

myPropertyMap.put("studentName",PropertiesUtil.toString(componentContext.getProperties().get("studentName"), "studentName"));

myPropertyMap.put("gender",PropertiesUtil.toString(componentContext.getProperties().get("Gender"), "Gender"));

myPropertyMap.put("isSenior",PropertiesUtil.toString(componentContext.getProperties().get("isSenior"), "isSenior"));

String[] mySubjects = (String[]) componentContext.getProperties().get("subjects");

myPropertyMap.put("subjects",mySubjects.toString());

}

}

**MyOSGIConfigConfig** is the PID of the OSGI configuration



#### OSGI DECLARATIVE SERVICES ANNOTATIONS

**So far to create the Components, Sling servlets or Service we have used Felix SCR annotation.**

|  |  |
| --- | --- |
| **Plugin:**  For DS annotations we have to use “maven-bundle-plugin” instead of “maven-scr-plugin”. Version 3.2.0 or greater. | <plugin>  <groupId>org.apache.felix</groupId>  <artifactId>maven-bundle-plugin</artifactId>  <version>3.5.1</version>  <inherited>true</inherited>  </plugin> |
| **Dependencies**:  For DS annotations we need artifacts “org.osgi.service.metatype.annotations” and “org.osgi.service.component.annotations” instead “org.osgi.core” and “org.osgi.compendium”. | <dependency>  <groupId>org.osgi</groupId>  <artifactId>**org.osgi.service.component.annotations**</artifactId>  <version>1.3.0</version>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>org.osgi</groupId>  <artifactId>**org.osgi.service.metatype.annotations**</artifactId>  <version>1.3.0</version>  </dependency> |
| In DS Annotation, package “org.apache.felix.scr.annotations.\*” will be replaced with “org.osgi.service.component.annotations.\*” and “org,osgi.service.metatype.annotations.\*”. | |

##### DS-CREATING OSGI CONFIGURATION

|  |  |
| --- | --- |
| osgi.PNG | 1. Create a separate or an inner interface which would hold configurations. In example, we have created **Config** interface. 2. Add **@ObjectClassDefinition** annotation to the interface. Also, add desired attributes    1. **name**: The name would help you search the configuration in OSGi’s configuration manager. 3. Add **@Designate** to the Component that would consume the configurations.  The ocd attribute should refer to the Configuration interface created in #2. 4. Declare properties that you would like to configure via @AttributeDefinition 5. To define property names similar to blog.name , your method would be named blog\_name   Please note that there are 2 ways to define default values:   * 1. defaultValue attribute of @AttributeDefinition   2. Specifying default value in variable declarartion |

**CODE**

import org.osgi.service.component.annotations.Activate;

import org.osgi.service.component.annotations.Component;

import org.osgi.service.metatype.annotations.AttributeDefinition;

import org.osgi.service.metatype.annotations.AttributeType;

import org.osgi.service.metatype.annotations.Designate;

import org.osgi.service.metatype.annotations.ObjectClassDefinition;

import org.osgi.service.metatype.annotations.Option;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@Component

@Designate(ocd = ConfigurableComponentPorpertiesImpl.Config.class)

public class ConfigurableComponentPorpertiesImpl {

private static final Logger LOGGER = LoggerFactory.getLogger(ConfigurableComponentPorpertiesImpl.class);

@ObjectClassDefinition(name = "Techrevel Sample Configuration", description = "This is sample configuration")

public @interface Config {

**@AttributeDefinition(name = "Blog name", defaultValue = "techrevel", description = "Name of the blog")**

String blog\_name();

@AttributeDefinition(name = "Blog Url")

String blog\_URL() default "https://techrevel.blog";

// Multi-values property

**@AttributeDefinition(name = "Blog Topics")**

String[] blog\_Topics() default { "OSGi", "AEM" };

@AttributeDefinition(name = "Blog Count", description = "Total number of blogs", type = AttributeType.INTEGER)

int blogCount() default 0;

// Password

**@AttributeDefinition(name = "password", type = AttributeType.PASSWORD)**

String password();

// Checkbox

**@AttributeDefinition(name = "Blog is active?")**

boolean blogIsActive() default true;

// Dropdown

**@AttributeDefinition(name = "Blog is hosted at?", options = { @Option(label = "WordPress", value = "wordpress"),**

**@Option(label = "Blogspot", value = "blogspot") })**

String hostedAt() **default** "";

}

@Activate

protected void activate(final Config config) {

LOGGER.info("Blog name: " + config.blog\_name());

LOGGER.info("Blog URL: " + config.blog\_URL());

for (String topic : config.blog\_Topics()) { LOGGER.info("Blog Topics: " + topic); }

LOGGER.info("Blog Count: " + config.blogCount());

LOGGER.info("Blog Password: " + config.password());

LOGGER.info("Blog Is Active? " + config.blogIsActive());

LOGGER.info("Blog is hosted at? " + config.hostedAt());

}

}

##### DS – SLING SERVLETS

@Component(

immediate = true,

service = Servlet.class,

property = {

"sling.servlet.extensions=txt",

"sling.servlet.paths=/bin/osgi",

"sling.servlet.paths=/bin/foo",

"sling.servlet.methods=get"

},

configurationPid = "com.nateyolles.aem.osgiannotationdemo.core.servlets.SampleOsgiServlet"

)

@Designate(ocd=SampleOsgiServlet.Configuration.class)

public class SampleOsgiServlet extends SlingSafeMethodsServlet {

private static final long serialVersionUID = 1L;

@Reference

private SampleOsgiService sampleOsgiService;

private boolean enabled;

@Override

protected void doGet(final SlingHttpServletRequest req,

final SlingHttpServletResponse resp) throws ServletException, IOException {

PrintWriter out = resp.getWriter();

resp.setContentType("text/plain");

out.write("Annotation Demo Servlet - OSGi - enabled: " + enabled + "\n");

out.write(sampleOsgiService.getSettings());

}

@Activate

@Modified

protected void Activate(Configuration config) {

enabled = config.enabled();

}

@ObjectClassDefinition(name = "Annotation Demo Servlet - OSGi")

public @interface Configuration {

@AttributeDefinition(

name = "Enable",

description = "Sample boolean property"

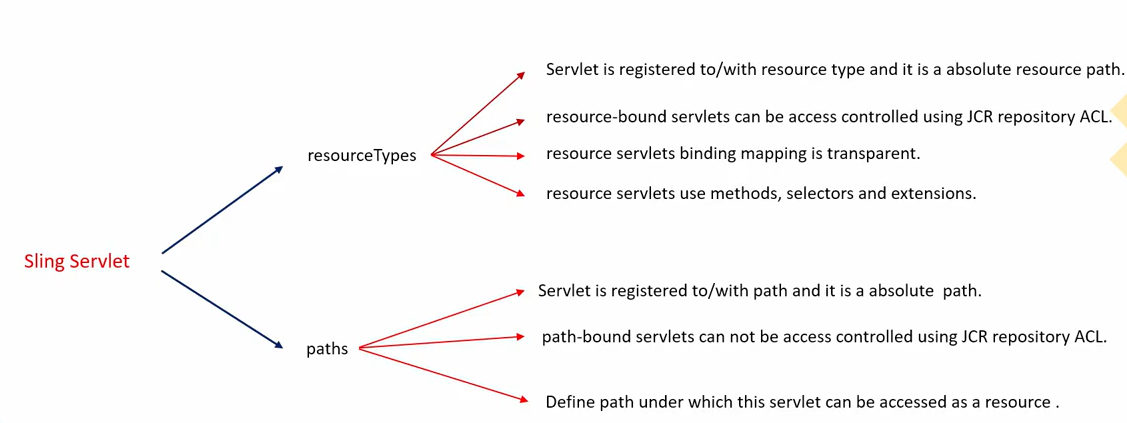
)

boolean enabled() default false;

}

}

### SLING SERVLET



**@SlingServlet(paths={"/bin/myservlet"})**

public class SampleServlet extends SlingSafeMethodsServlet {

@Override

protected void doGet(SlingHttpServletRequest request,SlingHttpServletResponse response) throws ServletException,IOException {

/\*Code Here \*/

}

}

In Sling, servlets can be registered as OSGi services by the below mentioned samples:

**1. The @SlingServlet annotation**

@**SlingServlet**(

resourceTypes = "sling/servlet/default",

selectors = "hello",

extensions = "html",

methods = "GET")

public class MyServlet extends SlingSafeMethodsServlet {

@Override

protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException {

/\*Write your logic here.\*/

}}

**2. The @Properties and @Property annotations**

@Component(metatype = true)

@Service(Servlet.class)

@Properties({

@Property(name = "sling.servlet.resourceTypes", value = "sling/servlet/default"),

@Property(name = "sling.servlet.selectors", value = "hello"),

@Property(name = "sling.servlet.extensions", value = "html"),

@Property(name = "sling.servlet.methods", value = "GET")

})

public class MyServlet extends SlingSafeMethodsServlet {

@Override

protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException {

/\*Write your logic here.\*/

}

}

**Besides the above methods , you can register a Sling Servlet using the Standard approaches:**

**1. REGISTERING THE SERVLET BY PATH**

*@SlingServlet(*

*paths={"/bin/customservlet/hashim"}*

*)*

*@Properties({*

*@Property(name="service.pid", value="com.day.servlets.SampleServlet",propertyPrivate=false),*

*@Property(name="service.description",value="SampleDescription", propertyPrivate=false),*

*@Property(name="service.vendor",value="SampleVendor", propertyPrivate=false)*

*})*

*public class SampleServlet extends* ***SlingAllMethodsServlet****{*

*@Override*

*protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException{*

*//Write your custom code here}*

*@Override*

*protected void doPost(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException*

*{//Write your custom code here*

*}*

*}*

**2. REGISTER SERVLET BY RESOURCE TYPE**

**Description** :if the servlet is registered using a resourceType then the servlet is invoked when we access the page having that resourceType.

***@SlingServlet(resourceTypes = "/apps/workflowapp/components/dropdown", methods = "GET",selectors="data")***

*public class MySafeMethodServlet extends* ***SlingSafeMethodsServlet*** *{*

*@Override*

*protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException {*

*List<String> selectBoxValue= new ArrayList<String>();*

*selectBoxValue.add("First Value");*

*selectBoxValue.add("Second Value");*

*request.setAttribute("selectBoxValue", selectBoxValue);*

*response.getWriter().print(selectBoxValue);*

*}*

*}*

**URL to invoke the servlet**: <http://localhost:4502/content/testpage.data.html>

* In case you want the servlet to work with resourceType the servlet should have an additional configuration for selectors property.This configuration let's you run a servlet in context of a resource (of a particular resourceType) instead of a global one.
* If you need your @SlingServlet to fetch some properties from Felix Console Configurations using @Properties , add the parameter **“metatype=true”** in the form of declaration where @SlingServlet is used. This is the parameter responsible for a Service component to be available in Felix Console configMgr.
* The Sling Servlet resolver can be used to check whether the servlet is resolved or not When URL is entered  Servlet Class is resolved[with full package]

**DIFFERENCE BETWEEN SLINGSAFEMETHODSSERVLET AND SLINGALLMETHODSSERVLET.**

**SlingSafeMethodsServlet** - Helper base class for read-only Servlets used in Sling. This base class is actually just a better implementation of the Servlet API HttpServlet class which accounts for extensibility. So extensions of this class have great control over what methods to overwrite. It supports **GET, HEAD, OPTIONS** etc methods. Read more

**SlingAllMethodsServlet** - Helper base class for data modifying Servlets used in Sling. This class extends the SlingSafeMethodsServlet by support for the POST, PUT and DELETE methods.

**SUPPOSE I HAVE ADDED SLING.SERVLET.PATHS AND SLING.SERVLET.RESOURCETYPES BOTH IN SERVLET. WHAT WILL HAPPEN?**

Suppose I have added sling.servlet.paths and sling.servlet.resourceTypes both in servlet. sling.servlet.resourceTypes property is ignored if the sling.servlet.paths property is set.

### EVENT LISTENER IN AEM

You can develop a custom event handler for Adobe Experience Manager (AEM) that responds to events that occur at the JCR level.

Ways of doing so are detailed here :

1. At the JCR level with observation
2. At the Sling level with event handlers and jobs
3. At the CQ level with workflows & launchers
4. Particular case of scheduled events
5. Particular case of POST to the repository

For example, you can write an event handler to respond to the following JCR events:

1. A node was added
2. A node was moved
3. A node was deleted
4. A property was added to a node
5. A property was changed
6. A property was deleted

To create an AEM event handler, create an OSGi bundle that contains a class that implements **javax.jcr.observation.EventListener**.

|  |  |
| --- | --- |
| https://helpx.adobe.com/experience-manager/using/events/_jcr_content/main-pars/image.img.png/Events.png | The following illustration shows a client making a change in an AEM web page that results in a new node being created in the AEM JCR. For example, a new customer signs up that results in a new node being created within the AEM JCR. The custom event handler listens for that event and responds. |

This development article guides you through creating an AEM event handler that responds to the **'Add Node'** event that occurs when a node is added under content/claim. Application logic that represents an Event Hander is implemented as an OSGi bundle that is built using Declarative Services (DS) and Maven. DS is used to inject a ResourceResolverFactory instance into the service. The OSGi bundle is a managed component, which means that the OSGi service container creates the ResourceResolverFactory instance.

**Example 1**:- This event will get triggred with Replication event occurs.

@Service(value = EventHandler.**class**)

@Component(immediate = **true**)

@Property(name = "event.topics", value = ReplicationAction.***EVENT\_TOPIC***)

**public** **class** ReplicationLogger **implements** EventHandler, JobConsumer {

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(ReplicationLogger.**class**);

@Reference

**private** ResourceResolverFactory resourceResolverFactory;

@Reference

**private** SlingRepository repository;

@Override

**public** **void** handleEvent(Event event) {

ReplicationAction action = ReplicationAction.*fromEvent*(event);

**if** (action.getType().equals(ReplicationActionType.***ACTIVATE***)) {

**try** {

ResourceResolver resourceResolver = **null**;

Session session;

session = repository.loginService(**null**, repository.getDefaultWorkspace());

Map<String, Object> newmap = **new** HashMap<String, Object>();

newmap.put(JcrResourceConstants.***AUTHENTICATION\_INFO\_SESSION***, session);

resourceResolver = resourceResolverFactory.getResourceResolver(newmap);

**final** PageManager pm = resourceResolver.adaptTo(PageManager.**class**);

**final** Page page = pm.getContainingPage(action.getPath());

**if**(page != **null**) {

***LOGGER***.info("\*\*\*\*\*\*\*\* Activation of page {}", page.getTitle());

}

**if**(resourceResolver != **null** && resourceResolver.isLive()) {

resourceResolver.close();

}

}

**catch** (LoginException e) { e.printStackTrace(); }

**catch** (javax.jcr.LoginException e) { e.printStackTrace(); }

**catch** (RepositoryException e) { e.printStackTrace(); }

}

process (**null**);

}

@Override

**public** JobConsumer.JobResult process(Job event) {

***LOGGER***.info("\*\*\*\*\*\*\*\*processing job");

**return** JobConsumer.JobResult.***OK***;

}

}

**Important Points**

1. The Repication Type is an ENUM

public enum **ReplicationActionType** {

*ACTIVATE, DEACTIVATE, DELETE, TEST, REVERSE, INTERNAL\_POLL;*

}

1. To retrieve the session object we are relaying on the user configured in “**Apache Sling Service User Mapper**” from : <http://localhost:4502/system/console/configMgr>



#### EVENT LISTENER USING OBSERVATION MANAGER

The Observation Manager API is responsible for listening all the events occours in the repository, Below Code will put a (!) mark with the jcr:title when a the jcr:title property is changed/added

@Component

**public** **class** TitlePropertyListener **implements** EventListener {

p**rivate** **final** Logger LOGGER = LoggerFactory.*getLogger*(TitlePropertyListener.**class**);

@Reference

**private** SlingRepository repository;

**private** Session session;

**private** ObservationManager observationManager;

**protected** **void** activate(ComponentContext context) **throws** Exception {

session = repository.loginService(**null**, repository.getDefaultWorkspace());

observationManager = session.getWorkspace().getObservationManager();

observationManager.addEventListener(**this**, Event.***PROPERTY\_ADDED*** | Event.***PROPERTY\_CHANGED***, "/", **true**, **null**, **null**, **true**);

LOGGER.info("\*\*\*\*\*\*\*\*added JCR event listener");

}

**protected** **void** deactivate(ComponentContext componentContext) {

**try** {

**if** (observationManager != **null**) {

observationManager.removeEventListener(**this**);

LOGGER.info("\*\*\*\*\*\*\*\*removed JCR event listener");

}

}

**catch** (RepositoryException re) {

LOGGER.error("\*\*\*\*\*\*\*\*error removing the JCR event listener", re);

}

**finally** {

**if** (session != **null**) {

session.logout();

session = **null**;

}

}

}

**public** **void** onEvent(EventIterator it) {

**while** (it.hasNext()) {

Event event = it.nextEvent();

**try** {

LOGGER.info("\*\*\*\*\*\*\*\*new property event: {}", event.getPath());

Property changedProperty = session.getProperty(event.getPath());

**if** (changedProperty.getName().equalsIgnoreCase("jcr:title")

&& !changedProperty.getString().endsWith("!")) {

changedProperty.setValue(changedProperty.getString() + "!");

session.save();

}

}

**catch** (Exception e) { LOGGER.error(e.getMessage(), e); }

}

}

}

Analysis of 🡪 observationManager.addEventListener(**this**, Event.***PROPERTY\_ADDED*** | Event.***PROPERTY\_CHANGED***, "/", **true**, **null**, **null**, **true**);

Syntax🡪 [**addEventListener**](https://docs.adobe.com/docs/en/spec/jsr170/javadocs/jcr-1.0/javax/jcr/observation/ObservationManager.html#addEventListener(javax.jcr.observation.EventListener, int, java.lang.String, boolean, java.lang.String[], java.lang.String[], boolean))**([EventListener](https://docs.adobe.com/docs/en/spec/jsr170/javadocs/jcr-1.0/javax/jcr/observation/EventListener.html" \o "interface in javax.jcr.observation) listener, int eventTypes, java.lang.String absPath, boolean isDeep, java.lang.String[] uuid, java.lang.String[] nodeTypeName, boolean noLocal)**

In the [**addEventListener**](https://docs.adobe.com/docs/en/spec/jsr170/javadocs/jcr-1.0/javax/jcr/observation/ObservationManager.html#addEventListener(javax.jcr.observation.EventListener, int, java.lang.String, boolean, java.lang.String[], java.lang.String[], boolean)) the listener is registered based on the condition which is derived by ANDing all the conditions, passed in the parameter of [**addEventListener**](https://docs.adobe.com/docs/en/spec/jsr170/javadocs/jcr-1.0/javax/jcr/observation/ObservationManager.html#addEventListener(javax.jcr.observation.EventListener, int, java.lang.String, boolean, java.lang.String[], java.lang.String[], boolean)).

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| listener | [EventListener](https://docs.adobe.com/docs/en/spec/jsr170/javadocs/jcr-1.0/javax/jcr/observation/EventListener.html) | Event Listener Object which we want to register |
| eventTypes | int | Event on which the event lsitener will beb triggered |
| absPath | String | Specify the path where you want the event listener to listen to, means it will get trigger of any changes occurs in this path only |
| isDeep | boolean | **"True"** - if we want the listener to listen when the changes occurs in the children /subchildern as well otherwise **false. Preference is project path** |
| uuid | String[] | Array of UUID: This is the unique ID associated with each and every node. So by specifiying the values means the listener will be triggered for the changes happening only on those nodes |
| nodeTypeName | String[] | Nodes, only on which the event listener will be triggered, when changes occours Type e.x : nt:unstructured,cq:Page etc.. |
| noLocal | boolean | This parameter helps in avoiding the listerner to go in infinite loop when its set as **true.** Explained below |

**Question : What is the purpose of noLocal(boolean)** [**addEventListener**](https://docs.adobe.com/docs/en/spec/jsr170/javadocs/jcr-1.0/javax/jcr/observation/ObservationManager.html#addEventListener(javax.jcr.observation.EventListener, int, java.lang.String, boolean, java.lang.String[], java.lang.String[], boolean)) **method?**

Answer: To explain lets take the below example where noLocal parameter is set to “true”

observationManager.addEventListener(**this**, Event.***PROPERTY\_ADDED*** | Event.***PROPERTY\_CHANGED***, "/", **true**, **null**, **null**, **true**);

This event listener is triggered when and property changes is occurring at the root(“/”) path. Just in case , in the code if we are changing some property of the node in the next line itself, just after registering the listener. This will in turn call the listerner again and again . This may end up with recursive calling of same listener and cause a condition of infinite looping.The noLocal parameter avoid this situation when we set the its value as true

**Question : What is UUID on the nodes?**

Ans : UUID is an unique id associated with each node in CRX. The UUID is not be visible unless and until the page is not activated. Once we activate the page the UUID will be visible as shown below



### SCHEDULERS

**Sample Codes:**

**Example 1:**

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.apache.felix.scr.annotations.Component;

**import** org.apache.felix.scr.annotations.Service;

**import** org.apache.felix.scr.annotations.Property;

@Component

@Service(value = Runnable.**class**)

@Property( name = "scheduler.expression", value = "0 \* \* \* \* ?")

**public** **class** ScheduledCronJob **implements** Runnable {

**protected** **final** Logger log = LoggerFactory.*getLogger*(**this**.getClass());

**public** **void** run() {

log.info("Executing a cron job (job#1) through the whiteboard pattern");

}

}

Example 2:

**package** com.workflow.impl;

**import** java.util.Dictionary;

**import** javax.jcr.RepositoryException;

**import** javax.jcr.Session;

**import** org.apache.felix.scr.annotations.Component;

**import** org.apache.felix.scr.annotations.Property;

**import** org.apache.felix.scr.annotations.Reference;

**import** org.apache.felix.scr.annotations.Service;

**import** org.apache.sling.jcr.api.SlingRepository;

**import** org.osgi.service.component.ComponentContext;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

@Component(immediate = **true**, metatype = **true**, label = "Cleanup Service")

@Service(value = Runnable.**class**)

@Property(name = "scheduler.expression", value = "\*/5 \* \* \* \* ?")// Every 5 seconds

**public** **class** CleanupServiceImpl **implements** Runnable {

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(CleanupServiceImpl.**class**);

@Reference

**private** SlingRepository repository;

@Property(label = "Path", description = "Delete this path", value = "/tmp/mypath")

**public** **static** **final** String ***CLEANUP\_PATH*** = "cleanupPath";

**private** String cleanupPath;

**protected** **void** activate(ComponentContext componentContext) {

configure(componentContext.getProperties());

}

**protected** **void** configure(Dictionary<?, ?> properties) {

**this**.cleanupPath = (String.*valueOf*(properties.get(***CLEANUP\_PATH***)) != **null**) ? String

.*valueOf*(properties.get(***CLEANUP\_PATH***)) : **null**;

***LOGGER***.info("configure: cleanupPath='{}''", **this**.cleanupPath);

}

@Override

**public** **void** run() {

***LOGGER***.info("running now-Cleanup path: " + **this**.cleanupPath);

Session session = **null**;

**try** {

session = repository.loginService(**null**,repository.getDefaultWorkspace());

***LOGGER***.info("Logged in with " + session.getUserID()+ " & Is session Live: " + session.isLive());

**if** (session.itemExists(cleanupPath) == **true**) {

session.removeItem(cleanupPath);

***LOGGER***.info("node deleted");

session.save();

}

} **catch** (RepositoryException e) {

***LOGGER***.error("exception during cleanup", e);

} **finally** {

**if** (session != **null**) {

session.logout();

}

}

}

}

**WHAT IS APACHE JACKRABBIT?**

Apache Jackrabbit is a fully featured content repository that implements the entire JCR API.

|  |  |
| --- | --- |
| **OAK**  •The OAK is the new version of Jack rabbit. JackRabbit/ JackRabbit2 are the abstract implementation of JCR.  •AEM is the implementation of JackRabbit3 in form of CRX. JackRabbit 3 is called OAK. | Advantage of OAK   * Scalability * Big repositories * Distributed, many cluster nodes * Write throughput * Parallel writes * Write performance * Many child nodes * Many ACLs |

### SLING MODEL EXPORTER

#### JACKSON EXPORTER

#### XML EXPORTER

### HTML TEMPLATE LANGUAGE (HTL) – SIGHTLY

|  |  |
| --- | --- |
| * Sightly is the new AEM templating system * Preferred script for developing new components * Sightly is HTML5 * Separation of Concerns(UI and AEM) * Secure by Default : Avoid Cross Site Scripting or Malacious attack | Note:  A JSP can include a Sightly template:  **<cq:include script="footer.sly"/>**  A Sightly template can include a JSP:  **<div data-sly-include="footer.jsp"/>** |

**FUNDAMENTAL CONCEPTS OF HTL**

The HTML Template Language uses an expression language to insert pieces of content into the rendered markup, and HTML5 data attributes to define statements over blocks of markup (like conditions or iterations). As HTL gets compiled into Java Servlets, the expressions and the HTL data attributes are both evaluated entirely server-side, and nothing remains visible in the resulting HTML.

|  |  |
| --- | --- |
| <h1 **data-sly-test**="${properties.jcr:title}">  **${properties.jcr:title}**  </h1> | **BLOCK STATEMENTS 🡪** To conditionally display the <h1> element, a **data-sly-test** **HTML5 data attribute** is used. HTL provides multiple such attributes, which allow to attach behavior to any HTML element, and all are prefixed with **data-sly**.  **EXPRESSION LANGUAGE 🡪** HTL expressions are delimited by characters **${ and }.** At runtime, these expressions are evaluated and their value is injected into the outgoing HTML stream. |

#### SLY ELEMENT

Sly elements are never generated in the HTML source code. Sometimes there might not be an existing element at the exact location where a block statement has to be inserted. For such cases, it is possible to insert a special SLY element that will be automatically removed from the output, while executing the attached block statements and displaying its content accordingly.

|  |  |
| --- | --- |
| **SLY ELEMENT**  <sly data-sly-test="${properties.jcr:title}">  <h1>${properties.jcr:title}</h1>  </sly>  GENERATED HTML : No HTML generated for <sly> | **BLOCK ELEMENT**  <div data-sly-test="${properties.jcr:title}">  <h1>${properties.jcr:title}</h1>  </div>  GENERATED HTML |

#### BLOCK ELEMENT

HTML Template Language (HTL) block statements are custom data attributes added directly to existing HTML. This allows easy and unobtrusive annotation of a prototype static HTML page, converting it to a functioning dynamic template without breaking the validity of the HTML code.

|  |  |  |
| --- | --- | --- |
| **Block Statements[https://docs.adobe.com/docs/en/htl/docs/block-statements.html]** | | |
|  | **Syntax** | **Description** |
| use | data-sly-use | Initializes a helper object (defined in JavaScript or Java) and exposes it through a variable |
| 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 HTL presentation logic but are not desired in the actual output. |
| text | data-sly-text | Replaces the content of its host element with the specified text. |
| attribute | data-sly-attribute | Adds attributes to the host element. |
| 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. |
| resource | data-sly-resource | Includes the result of rendering the indicated resource through the sling resolution and rendering process. |
| include | data-sly-include | To include another file |
| template & call | data-sly-template data-sly-call |  |

#### DATA-SLY-USE

|  |  |
| --- | --- |
|  | **navigation.js**  use(function () {  return {  foo: "Hello World"  };  });  **htlAppComponent.html**  <div data-sly-use.nav='/apps/htlApp/components/navigation.js'>${nav.foo}</div> |
| **INCLUDING CLIENTLIBS**  STEP 1 : <sly data-sly-use.**clientLibs**="${'/libs/granite/sightly/templates/clientlib.html'}"/>  STEP 2 :  **For JS** :<sly data-sly-call="${clientLibs.**js** @ categories='company.headline'}" data-sly-unwrap/>  **For CSS**:<sly data-sly-call="${clientLibs.**css** @ categories='company.headline'}" data-sly-unwrap/>  **For Both** : <sly data-sly-call="${clientLibs.**all** @ categories='company.headline'}" data-sly-unwrap/>  **OR , we can combine STEP1 & STEP 2**  <sly data-sly-use.**clientLibs**="${'/libs/granite/sightly/templates/clientlib.html'}" data-sly-call="${clientLibs.**all** @ categories='company.headline'}" /> | |
| **INCLUDING MULTIPLE CLIENTLIBS**  <sly data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html"  data-sly-call="${clientlib.all @ categories=['myCategory1', 'myCategory2']}"/> | |

#### DATA-SLY-UNWRAP

Removes the host element while retaining its content. This is basically same as the previous one, only that it **removes the enclosing/wrapper elements**, in this case it removes the **<div></div>** elements.

|  |  |
| --- | --- |
| **<div data-sly-use.nav="navigation.js" data-sly-unwrap>${nav.foo}</div>** | Output 🡪Hello World |

#### DATA-SLY-TEST

<p data-sly-test="${properties.showtext}">text</p>

Note :$ {properties.showText} is some statement which evaluates to true.Hence displayed.

To demonstrate this example we need to follow below Steps

* Create a component having a widget [text field]

|  |  |
| --- | --- |
|  |  |

* Parsys component on the container component.

**<div data-sly-resource="${'content' @resourceType='wcm/foundation/components/parsys'}"></div>**

Here data-sly-resource is used to include a resource [Component]

1. **data-sly-include**

**Syntax :**

* **To Include HTML <div data-sly-include="../showTextComponent/includeFile.html"/>**
* **To Include JSP <div data-sly-include="../showTextComponent/includeFile.jsp"/>**

|  |  |
| --- | --- |
|  |  |

1. **data-sly-text** : Replaces the content of its host element with the specified text. The advantage this method is that is allows the unobtrusive annotation of HTML while keeping the static placeholder content from the original designer.

Example: <p data-sly-text="${properties.name}">Name Here</p> which is equivalent to <p>${properties.name}</p>

Example: Reading from Java Class

|  |  |
| --- | --- |
| **JAVA CLASS**  **public** **class** StudentBean {  **private** String studentName ="Amit";  **public** String getStudentName() {  **return** studentName;  }  **public** **void** setStudentName(String studentName) {  **this**.studentName = studentName;  }  } | **IN HTML (USING HTL)**  <sly data-sly-use.studentBean="com.adobe.training.core.StudentBean"/>  <p data-sly-text="${studentBean.studentName}">Name Here</p> |

1. **data-sly-attribute**: Adds attributes to the host element.

|  |  |  |
| --- | --- | --- |
| <div title="${properties.jcr:title}"></div> | is equivalent to | <div title="Lorem Ipsum" data-sly-attribute.title="${properties.jcr:title}"></div> |

Both will set the title attribute to the value of jcr:title. The advantage of the second method is that is allows the unobtrusive annotation of HTML while keeping the static placeholder content from the original designer.

* **Attributes are resolved left to right, with the rightmost instance of an attribute (either literal or defined via data-sly-attribute) taking precedence over any instances of the same attribute (defined either literally or via data-sly-attribute) defined to its left.**
* Note that an attribute (either literal or set via data-sly-attribute) whose value evaluates to the empty string will be removed in the final markup. The one exception to this rule is that a literal attribute set to a literal empty string will be preserved. For example,

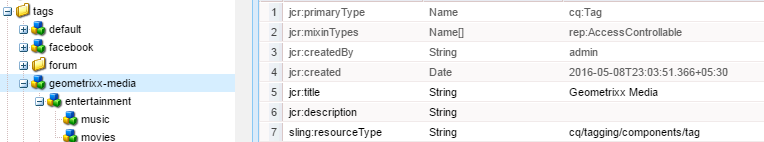
|  |  |  |
| --- | --- | --- |
| <div class="${''}" data-sly-attribute.id="${''}"></div> | produces | <div></div> |
| <div class="" data-sly-attribute.id=""></div> | produces | <div class=""></div> |

To set multiple attributes, pass a map object hold key-value pairs corresponding to the attributes and their values. For example, assuming,

|  |  |
| --- | --- |
| attrMap = {  title: "myTitle",  class: "myClass",  id: "myId"  } | <div data-sly-attribute="${attrMap}"></div>  Produces  <div title="myTitle" class="myClass" id="myId"></div> |

### TAGGING

1. The tag must exist as a node of type cq:Tag(**jcr:primaryType**) under the taxonomy root node**(/etc/tags**). A tag has optional meta information such as a title, localized titles and a description. The title should be displayed in user interfaces instead of the TagID, when present.



1. The TagID is added to the content node's cq:tags property and resolves to a node of type cq:Tag. Here the values correspomding to cq:tags property are the TagId of the tags. Tags are identified by a unique TagID.

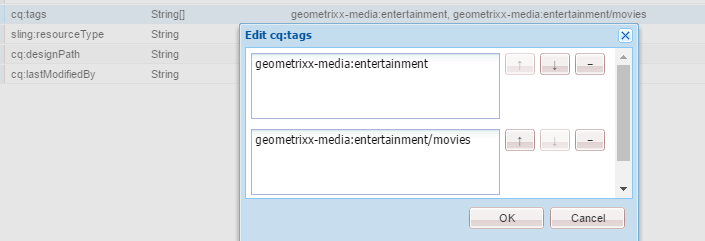


**TAG CHARACTERISTICS**

* node type is **cq:Tag**
* node name is a component of the TagID(name of the node is its Tag Id)
* the TagID always includes a namespace
* optional jcr:title property (the Title to display in the UI)
* optional jcr:description property
* when containing child nodes, is referred to as a container tag
* is stored in the repository below a base path called the taxonomy root node(/etc/tags)

**TAGID**

* A TagID identifies a path which resolves to a **tag node** in the repository. (node name of the tag is tag id of that tag)
* Typically, the TagID is a **shorthand TagID** starting with the namespace or it can be an **absolute TagID** starting from the taxonomy root node.
* When content is tagged, if it does not yet exist, the cq:tags property is added to the content node and the TagID is added to the property's **String array value**.



**TAXONOMY ROOT NODE 🡪**The taxonomy root node is the base path for all tags in the repository. The taxonomy root node must not be a node of type cq:Tag. In AEM, the base path is **/etc/tags and the root node is of type cq:Folder**.

**TAG NAMESPACE**

* Namespaces allow to group things. The most typical use-case is to have a namespace per (web)site (e.g. public, internal, portal, etc.) or per larger application (e.g. WCM, Assets, Communities) but namespaces can be used for various other needs. Namespaces are used in the user interface to only show the subset of tags (i.e. tags of a certain namespace) that is applicable to the current content.
* The tag's namespace is the first level in the taxonomy subtree, which is the node immediately below the taxonomy root node. A namespace is a node of type cq:Tag whose parent is not a cq:Tag node type.

**Note : All tags have a namespace. If no namespace is specified, the tag is assigned to the default namespace, which is TagID default (Title is Standard Tags), i.e., /etc/tags/default.**

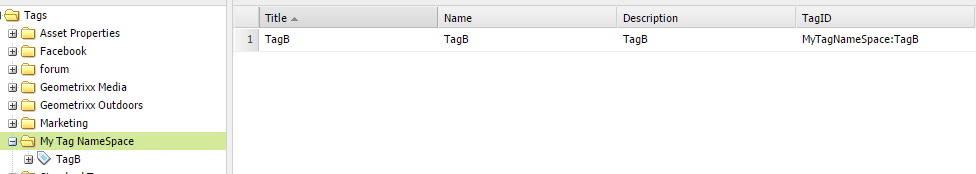
**CONTAINER TAGS**

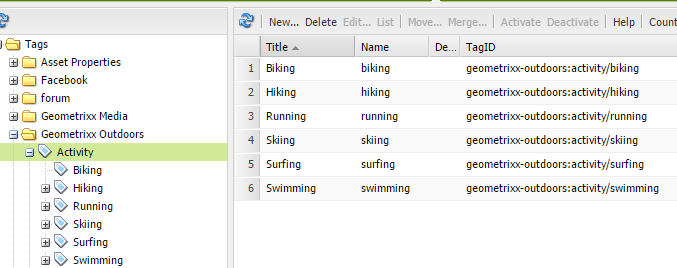
A container tag is a node of type cq:Tag containing any number and type of child nodes, which makes it possible to enhance the tag model with custom metadata.

Furthermore, container tags (or super-tags) in a taxonomy serve as the sub-summation of all sub-tags: for example content tagged with fruit/apple is considered to be tagged with fruit as well, i.e. searching for content just tagged with fruit would also find the content tagged with fruit/apple.

**RESOLVING TAGIDS**

If the tag ID contains a colon ":", the colon separates the namespace from the tag or sub-taxonomy, which are then separated with normal slashes "/". If the tag ID does not have a colon, the default namespace is implied.





* The standard and only location of tags is below /etc/tags.
* Tag referencing non-existing paths or paths that do not point to a cq:Tag node are considered invalid and are ignored.

**The following table shows some sample TagIDs, their elements, and how the TagID resolves to an absolute path in the repository :**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TagID** | **Namespace** | **Local ID** | **Container tag(s)** | **Leaf tag** | **Repository Absolute tag path** |
| dam:fruit/apple/braeburn | dam | fruit/apple/braeburn | fruit, apple | braeburn | /etc/tags/dam/fruit/apple/braeburn |
| color/red | default | color/red | color | red | /etc/tags/default/color/red |
| sky | default | sky | (none) | sky | /etc/tags/default/sky |
| dam: | dam | (none) | (none) | (none, the namespace) | /etc/tags/dam |
| /etc/tags/category/car | category | car | car | car | /etc/tags/category/car |

**LOCALIZATION OF TAG TITLE 🡪**When the tag includes the optional title string (jcr:title) it is possible to localize the title for display by adding the property **jcr:title.<locale>.**

**Overview of the Tagging API**

* **JcrTagManagerFactory** - returns a JCR-based implementation of a TagManager. It is the reference implementation of the Tagging API.
* **TagManager** - allows for resolving and creating tags by paths and names.
* **Tag** - defines the tag object.

**GETTING A JCR-BASED TAGMANAGER**

To retrieve a TagManager instance, you need to have a JCR Session and to call getTagManager(Session):

**@Reference**

**JcrTagManagerFactory jcrTagManagerFactory;**

**TagManager tagManager = jcrTagManagerFactory.getTagManager(session);**

In the typical Sling context you can also adapt to a TagManager from the ResourceResolver:

**TagManager tagManager = resourceResolver.adaptTo(TagManager.class);**

**RETRIEVING A TAG OBJECT :** A Tag can be retrieved through the TagManager, by either resolving an existing tag or creating a new one:

**Tag tag = tagManager.resolve("my/tag"); // for existing tags**

**Tag tag = tagManager.createTag("my/tag"); // for new tags**

For the JCR-based implementation, which maps Tags onto JCR Nodes, you can directly use Sling's adaptTo mechanism if you have the resource (e.g. such as /etc/tags/default/my/tag):

**Tag tag = resource.adaptTo(Tag.class);**

While a tag may only be converted from a resource (not a node), a tag can be converted to both a node and a resource :

**Node node = tag.adaptTo(Node.class);**

**Resource node = tag.adaptTo(Resource.class);**

**GETTING AND SETTING TAGS**

1. Getting the tags of a Resource: **Tag[] tags = tagManager.getTags(resource);**
2. Setting tags to a Resource: **tagManager.setTags(resource, tags);**

**SEARCHING FOR TAGS**

1. Searching for the Resource objects that are tagged with the tag object: **Iterator<Resource> it = tag.find();**
2. Retrieving the usage count of the tag object: long count = **tag.getCount();**
3. Searching for the Resource objects that are tagged with the tagID String:

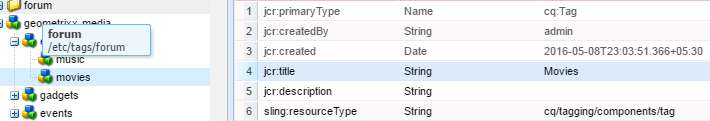
**RangeIterator<Resource> it = tagManager.find(tagID);**

**DELETING TAGS :** tagManager.deleteTag(tag);

**REPLICATING TAGS :I**t is possible to use the replication service (Replicator) with tags because tags are of type nt:hierarchyNode:

**replicator.replicate(session, replicationActionType, tagPath);**

For Example if tag is : **/etc/tags/geometrixx-media/entertainment/movies**



**@Reference**

private JcrTagManagerFactory jcrTagManagerFactory;

Session session = request.getResourceResolver().adaptTo(Session.class);

TagManager tagManager = jcrTagManagerFactory.getTagManager(session);

Tag[] tags =tagManager.getTags(resource);

for (Tag tag: tags) {

String tagId = tag.getTagID(); 🡨 geometrixx-media:entertainment/movies

String localtagId = tag.getLocalTagID();🡨entertainment/movies

String localtagtitle = tag.getTitle();🡨jcr:title of Tag –(**Movies**)

String tagName = tag.getName();🡨movies

}

### WORKFLOW

WorkFlow Model 🡪Workflows enable you to automate processes for managing resources and publishing content in AEM environment. Workflows are comprised of a series of steps, and each step accomplishes a discrete task. A workflow is made of steps. Automated steps, also called process steps, can be defined by using either an ECMA script or a service (a Java class in a bundle).

WorkFlow Console: <http://localhost:4506/libs/cq/workflow/content/console.html>

|  |  |
| --- | --- |
| **There are different types of workflow steps:**   * Participant (User/Group) * Process (Script, * Java method call) * Container (Sub Workflow) * OR Split/Join * AND Split/Join | **The following actions are possible on a workflow instance:**   * Terminate * Suspend * Resume * Restart |
| **WorkFlow Terms** | **Description** |
| Model | Is made of Workflow nodes and WorkflowTransitions. The transitions connect the nodes and define the "flow". The Model has always a start node and an end node. |
| Step | There are different types of workflow steps:  1. Participant (User/Group): These types of steps generate a work item and assign it to a user or group. A user must complete the work item to advance the workflow.  2. Process (Script, Java method call): This type of step is executed automatically by the system. An ECMA script or Java class implements the step. 3. Container (Sub Workflow): This step starts another workflow model. 4. OR Split/Join: Uses logic to decide which step to execute next in the workflow. 5. AND Split/Join: Executes multiple steps simultaneously. |
| Transition | Defines the link between two consecutive steps. |
| WorkItem | 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) or a resource by a URL or by a serialized java object**. Referencing a resource in the repository is very flexible and in conjunction with sling very productive: for example the referenced node could be rendered as a form. |
| 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** |
| Inbox | Each logged in user has its own workflow inbox in which the assigned WorkItems are accessible. The WorkItems are assigned either to the user itself or to the group to which he belongs. |

|  |  |
| --- | --- |
|  | 1. Workflows resides in /etc/workflow/<custom\_workflow> folder in CRX |

**CUSTOM WORKFLOW**

|  |  |
| --- | --- |
| Step 1 : Navigate to workflow console  Step 2: Click on new  Step 3: Enter the title and Name of the workflow  Step 4 : Double Click on the work flow to open the workflow |  |

Step 4: To create participant (example admin) 🡪 Participant Step🡪 From **Workflow** section

1. To create branches drop “**Process Step**”.
2. **Handler Advance**
   1. The handler advance option when true (checked), will advance the workflow to the next step after the current process is done with its execution.
   2. In case the handler advance option is false (unchecked), the process script has to take care of advancing the workflow to the next step. In case the script doesn't handle this, the workflow would remain in the running state without proceeding further from the current step.

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
| **Timeout: The** period after which the step will be "timed out". You can select between Off, Immediate, 1h, 6h, 12h, 24h.  **Timeout Handler:** The handler which will control the workflow when the step times out; for example **com.day.cq.workflow.timeout.autoadvance.AutoAdvancer.**  For “No Action”- In case of rejection: Drag “No Operation Component” |  |
| To Attach the workflow to the page 🡪 Go to Site admin and right click and select “workflow and then select “Training Work Flow” (we created this in above steps) from drop down | **v** |

The Admin will get an email notification in Inbox🡪 select the message and click “complete” which will give an option to “Approve OR Reject”

**WorkFlow Example :** When an approver approves a work flow a new version of that page will be created

|  |  |
| --- | --- |
| **Step 1**: We associate a page with a workflow (created below) from WCM console- shown above |  |
| **Step 2**: When the work flow is triggered a mail land in the approver’s mailbox. Right click on the task🡪 Select the desired option 🡪 if Approved a new version will be created |  |

|  |  |
| --- | --- |
| **Workflow Tabs** | |
| **Models** | Lists the workflow models currently available. Here you can start, create, edit or delete workflow models. |
| **Instances** | Shows you details of workflow instances which 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. |
| **Failures** | Enables you to monitor and manage failed worklow instances. |

**Workflow Created**

|  |  |
| --- | --- |
|  |  |

#### Custom Work Flow

|  |  |
| --- | --- |
| Work Flow class must implement **WorkflowProcess** interface  **Sample Java Class**  **import** org.apache.felix.scr.annotations.Component;  **import** org.apache.felix.scr.annotations.Service;  **import** com.adobe.granite.workflow.WorkflowException;  **import** com.adobe.granite.workflow.WorkflowSession;  **import** com.adobe.granite.workflow.exec.WorkItem;  **import** com.adobe.granite.workflow.exec.WorkflowProcess;  **import** com.adobe.granite.workflow.metadata.MetaDataMap;  @Component  @Service  **public** **class** TrainingWorkFlowProcess **implements** WorkflowProcess {  @Override  **public** **void** execute(WorkItem item, WorkflowSession workflowSession,MetaDataMap metaDataMap) **throws** WorkflowException {  item.getWorkflowData().getPayload().toString();  **/\*\* /content/geometrixx/de(Returns Page Path) \*\*/**  System.***out***.println("My Work Flow Process Executed");  }} | **Associate the custom workflow steps** |

Trigger The work flow process by creating a page, as we have set the **launcher** on “**Created**”🡪 automatically the launcher will associate the workflow(custom work flow we created above) with Page

**Sample workflow process that sets an approve property to the payload based on the process argument value.**

@Component

@Service

public class MyProcess implements WorkflowProcess {

@Property(value = "An example workflow process implementation.")

static final String DESCRIPTION = Constants.SERVICE\_DESCRIPTION;

@Property(value = "Adobe")

static final String VENDOR = Constants.SERVICE\_VENDOR;

@Property(value = "My Sample Workflow Process")

static final String LABEL = "process.label";

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 {

Session jcrSession = session.getSession();

Node node = (Node) jcrSession.getItem(path);

if (node != null) {

node.setProperty("approved", readArgument(args));

jcrSession.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");

}

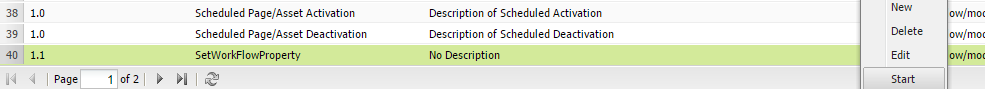
}

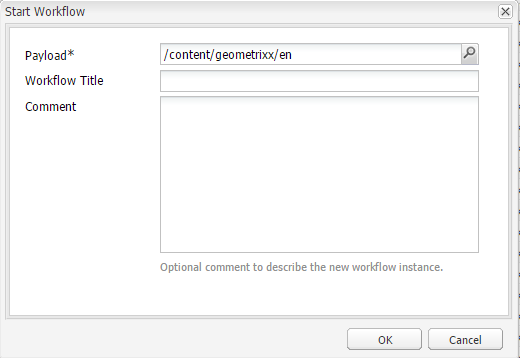
To execute of this work flow by passing the arguments:

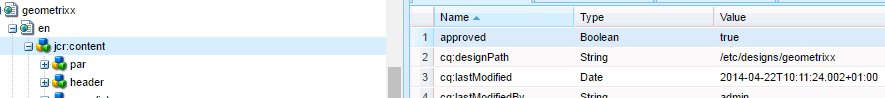
Step 1: Create a New Workflow Model

|  |  |
| --- | --- |
| Step 2: Drop the “Process Step” component and then select the workflow Process | Step 3: Pass the required argument in Arguments fields |

Step 4: Start the workflow and we can see the approved property added at the jcr:content of the payload







### REPLICATION

|  |  |
| --- | --- |
| C:\Users\asi104\Desktop\New folder\1359637917097.png | * Path to configure replication agent : <http://localhost:4506/miscadmin#/etc/replication> * Default Replicating Servlet :   http://<host\_name >:<port\_name>/bin/receive?sling:authRequestLogin=1 |
|  | **CREATING REPLICING AGENT**  Type of Agents :   * Replication Agent - * Reverse Replication Agent * Static Delivery Agent – This agent is used to flush the dispatcher. |
|  |  |

#### Reverse Replication

Some features allow users to enter data on a publish instance.

* In some cases, a type of replication known as **reverse replication**, is needed to return this data to the author environment from where it is redistributed to other publish environments. Due to security considerations, any traffic from publish to the author environment must be strictly controlled.
* Reverse replication uses an agent in the publish environment which references the author environment. This agent places the data into an outbox. This outbox is matched with replication listeners in the author environment. The listeners poll the outboxes to collect any data entered and then distribute it as necessary. This ensures that the author environment controls all traffic.

**How is content moved from publish instance to author instance?**

Consider the scenario where your website is having a blog or a forum, and the users are posting comments in the blog. Then that comments will only be in publish instance. The content is moved from publish instance to author instance using reverse replication and the job is done by reverse replication agent.

The reverse replication agent places any content updates in an outbox configured in publish instance. The replication listeners in author environment keep listening to the publish outbox and whenever any content is placed in publish outbox, the listeners update the content in author instance.

### VLT (VAULT) TOOL

CRXLite does not provide the facility to import/export projects from/to file system (only provides checkout from SVN- shown below) – so we use VLT tool for file system project Export and import. For Example if we have created a project in CRX and we need to import it those component in our file system (then the file system files can me imported as maven project in eclipse.). This can be done using VLT tool.

**To install VLT tool**

1. Extract from location : D:\AEM6.1\author\crx-quickstart\opt\filevault
2. Prerequisites
3. Set path for JAVA\_HOME
4. Set the path of VLT tool : D:\AEM6.1\author\crx-quickstart\opt\filevault\filevault\vault-cli-3.1.16\bin
5. Check on the command prompt : vlt --version
6. If we have done an operation once by providing credential– then on the second time we don’t have to give the credentials again because the credential get stored : C:\Users\asi104\.vault 🡪 file name as **auth.xml**

*<?xml version="1.0" encoding="UTF-8"?>*

*<auth version="1.0">*

*<repository uri="http://localhost:4506/crx/server/null">*

*<credentials type="simple">*

*<user name="admin" password="{DES}91ec9d85310ef2e33ac184dab74d2b5b"/>*

*</credentials>*

*</repository>*

*</auth>*

1. **To Export(CRX🡪File System) : vlt --credentials admin:admin export -v http://localhost:4506/crx /apps/training**
2. **vlt co http://localhost:4502/crx/-/jcr:root/apps/company company**

|  |  |
| --- | --- |
|  |  |

**Backup created on file Sytem**

1. **Import (File System 🡪 CRX) : vlt --credentials admin:admin co http://localhost:4502/crx/ . --force**

**Explanation:**  This command will pull the data from CRX to file system. Now the question is to which all location from CRX this will pull the data in file system. The location can be configured in **filter.xml** file.

Location of filter.xml file: <project\_name>\content\src\main\content\META-INF\vault

**Configuration:**

*<?xml version="1.0" encoding="UTF-8"?>*

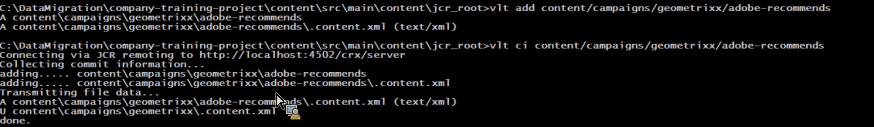
*<workspaceFilter version="1.0">*

***<filter root="/apps/workflowapp"/> 🡨 This specifies the location in CRX from where the content is pulled in file system.***

*</workspaceFilter>*

Now, suppose opposite to that, if we added a new content in the file system and now we have to push the content to CRX(although this can be done by maven too.)

1. Create the content in the filesystem
2. Add the file to the version : **vlt add** <*path\_of\_content\_in\_file\_system*>
3. Push the content to CRX : **vlt ci** <*path\_of\_content\_in\_file\_system*>



### AEM RESPONSIVE GRID SYSTEM

#### RESPOSIVE LAYOUT

|  |  |
| --- | --- |
|  | **AEM allows us to have a responsive layout for pages by using the Layout Container component.**   * This provides a paragraph system that allows you to position components within a responsive grid. This grid can rearrange the layout according to the device/window size and format. The component is used in conjunction with the **Layout mode**, which allows you to create and edit your responsive layout dependent on device.   **LAYOUT CONTAINER COMPONENT**   * Provides horizontal snap to grid, together with the ability to place components into the grid side-by-side and define when they should collapse/reflow. * Uses pre-defined breakpoints (e.g. for phone, tablet, etc.) to allow you to define the required behavior of content for related devices/orientation. For example, you can customize the component size or whether the component can be seen on specific devices. * Can be nested to allow column control.   The user can then see how the content will be rendered for specific devices using the emulator. |
| **ENABLING RESPONSIVE GRID / LAYOUT CONTAINER** | The responsive grid is enabled by including the responsivegrid in the container “.html” page (e.g. **body.html**)  **<div data-sly-resource="${'responsiveGrid' @resourceType='wcm/foundation/components/responsivegrid'}"></div>** |

#### ACTIVATING LAYOUT MODE

|  |  |
| --- | --- |
| screen_shot_2018-03-23at090140 | **Once the layout container is positioned on the page we can use the Layout mode to position content within the responsive grid.**  **STEPS TO ENABLE LAYOUT MODE**   1. Copy the cq:responsive node below the jcr:content node of the content page. 2. Copy the grid.css and styles.css file to the client – library   Hint : Copy the css from : **/libs/screens/core/components/multiscreenchannel/clientlibs** |
| **NODE STRUCTURE – LAYOUT MODE** | |
|  | |
| **BREAKPOINTS IN LAYOUT MODE** | |
|  | |

### SITEADMIN – IN DETAIL

|  |  |
| --- | --- |
|  |  |
| **When the SiteAdmin is loaded**   * It takes JSON as an input to render the data * The left tree of content is rendered using –tree.json * The right view is generated using – pages.json | |

**SITEADMIN IN CRX IS NOT VISIBLE, WHAT MAY BE THE REASON?**

|  |  |
| --- | --- |
|  | * Go to : <http://localhost:4502/system/console/configMgr> and search for Apache Sling Get Servlet * Check the property “Enable JSON” |

Note : We can even customize the view of siteadmin by overlaying the siteadmin component[/libs/wcm/core/content/siteadmin].

<https://docs.adobe.com/docs/en/cq/5-6-1/developing/customize_siteadmin.html>

**ADDING A CUSTOM COLUMN TO THE SITE ADMIN CONSOLE**

The Websites Administration console can be extended to display custom columns. The console is built based on a JSON [page.json and tree.json]object that can be extended by creating an OSGI service implementing the ListInfoProvider interface. Such a service modifies the JSON object that is sent to the client to build the console.

It consists of the following steps:

1. [Creating the OSGI service](https://docs.adobe.com/docs/en/cq/5-6-1/developing/customize_siteadmin.html#Creating the OSGI Service) and deploying the bundle containing it to the CQ server.
2. (optional) [Testing the new service](https://docs.adobe.com/docs/en/cq/5-6-1/developing/customize_siteadmin.html#Testing the New Service) by issuing a JSON call to request the JSON object that is used to build the console.
3. [Displaying the new column](https://docs.adobe.com/docs/en/cq/5-6-1/developing/customize_siteadmin.html#Displaying the New Column) by extending the nodes structure of the console in the repository.

**CREATING THE OSGI SERVICE**

The **ListInfoProvider** interface defines two methods:

* **updateListGlobalInfo**, Update list global information with custom properties
* **updateListItemInfo**, Update list item information with custom properties

The arguments for both methods are:

* request 🡪 the associated Sling HTTP request object,
* info 🡪 the JSON object to update, which is respectively the global list or the current list item,
* Resource 🡪 a Sling resource.

The sample implementation below:

* Adds a starred property for each item, which is true if the page name starts with an “e”, and false otherwise.
* Adds a starredCount property, which is global for the list and contains the number of starred list items.

To create the OSGI service: In CRXDE Lite, [create a bundle](https://docs.adobe.com/docs/en/cq/5-6-1/core/developing/development_tools/developing_with_crxde_lite.html#Managing%20a%20Bundle). And add the sample code below.**Add the dependencies [**com.day.cq.commons**] in the pom.xml of bundle**

|  |  |
| --- | --- |
| **package** com.test;  **import** com.day.cq.commons.ListInfoProvider;  **import** com.day.cq.i18n.I18n;  **import** com.day.cq.wcm.api.Page;  **import** org.apache.felix.scr.annotations.Component;  **import** org.apache.felix.scr.annotations.Service;  **import** org.apache.sling.api.SlingHttpServletRequest;  **import** org.apache.sling.api.resource.Resource;  **import** org.apache.sling.commons.json.JSONException;  **import** org.apache.sling.commons.json.JSONObject;    @Component(metatype = **false**)  @Service(value = ListInfoProvider.**class**)  **public** **class** StarredListInfoProvider i**mplements** ListInfoProvider {  **private** **int** count = 0;  **public** **void** updateListGlobalInfo(SlingHttpServletRequest request, JSONObject info, Resource resource) **throws** JSONException {  info.put("starredCount", count);  count = 0; // reset for next execution  }  **public** **void** updateListItemInfo(SlingHttpServletRequest request, JSONObject info, Resource resource) **throws** JSONException {  Page page = resource.adaptTo(Page.**class**);  **if** (page != **null**) {  // Consider starred if page name starts with 'e'  **boolean** starred = page.getName().startsWith("e");  **if** (starred) {  count++;  }  I18n i18n = **new** I18n(request);  info.put("starred", starred ? i18n.get("Yes") : i18n.get("No"));  }  }  } |  |

* *If your ListInfoProvider implementation defines a property that already exists in the response object, its value will be overwritten by the one you provide*.
* *You can use*[*service ranking*](http://www.osgi.org/javadoc/r2/org/osgi/framework/Constants.html#SERVICE_RANKING)*to manage the execution order of multiple ListInfoProviderimplementations.*

**TESTING THE NEW SERVICE**

When you open the Websites Administration console and browse through your site, the browser is issuing an ajax call to get the JSON object that is used to build the console.

|  |  |
| --- | --- |
| file | 1. For example, when you browse to the /content/geometrixxfolder, the following request is sent to the CQ server to build the console:   <http://localhost:4502/content/geometrixx.pages.json?start=0&limit=30&predicate=siteadmin>   1. To make sure that the new service is running after having deployed the bundle containing it:Point your browser to the following URL:   <http://localhost:4502/content/geometrixx.pages.json?start=0&limit=30&predicate=siteadmin>   1. The response should display the new properties as follows(started property added) |

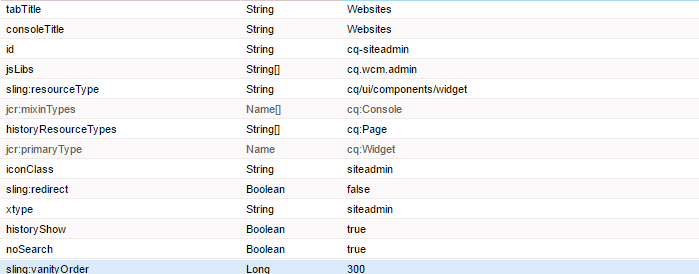
**DISPLAYING THE NEW COLUMN**

The last step consists in adapting the nodes structure of the Websites Administration console to display the new property for all the Geometrixx pages by overlaying /libs/wcm/core/content/siteadmin. Proceed as follows:

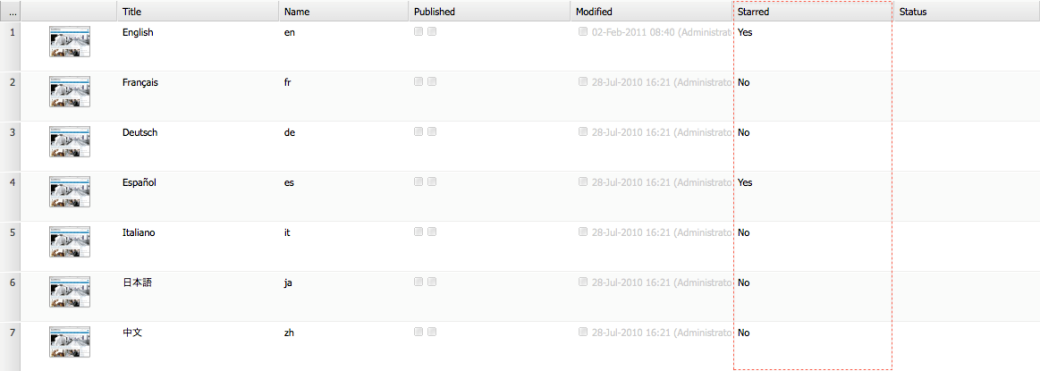
1. In CRXDE Lite, create the nodes structure /apps/wcm/core/content with nodes of type sling:Folder to reflect the structure /libs/wcm/core/content.
2. Copy the node /**libs/wcm/core/content/siteadmin** and paste it below **/apps/wcm/core/content**.
3. Copy the node /apps/wcm/core/content/siteadmin/grid/assets to /apps/wcm/core/content/siteadmin/grid/geometrixx and changes its properties:
4. Remove pageText
5. Set pathRegex to /content/geometrixx(/.\*)? 🡪This will make the grid configuration active for all geometrixx websites.
6. Set storeProxySuffix to .pages.json
7. Edit the storeReaderFields multivalued property and add the starred value.
8. Add a starred node (of type nt:unstructured) below/apps/wcm/core/content/siteadmin/grid/geometrixx/columns with the following properties:

* dataIndex: starred of type String
* header: Starred of type String
* xtype: gridcolumn of type String
* (optional) Drop the columns you do not want to display at/apps/wcm/core/content/siteadmin/grid/geometrixx/columns
* /siteadmin is a vanity path that, as default, points to /libs/wcm/core/content/siteadmin.

Note To redirect this to your version of siteadmin on /apps/wcm/core/content/siteadmin define the propertysling:vanityOrder to have a value higher than that defined on /libs/wcm/core/content/siteadmin. The default value is 300, so anything higher is suitable.



1. Go to the Websites Administration console and navigate to the Geometrixx site:
2. http://localhost:4502/siteadmin#/content/geometrixx.
3. The new column called Starred is available, displaying custom information as follows:



### SLING SCRIPT RESOLUTION

**REQUEST URL: /content/corporate/jobs/developer.html**

**STEP 1:** Sling checks - **/content/corporate/jobs/developer ,** If the node is not found it will return 404

**STEP 2:** if found ,Sling then looks for a special property on that node named "**sling:resourceType**," which (if present) determines the resource type for that node.

**STEP 3: Sling will look under /apps (then /lib) to find a script that applies to the resource type**.

**EXAMPLE 1**

Let's consider a very simple example. Suppose that the resource type for the above node is "hr/job." In that case, Sling will look for a script called **/apps/hr/job/job.jsp**. However, if such a script doesn't exist, Sling will then look for **/apps/hr/job/GET.jsp** to service the GET request. Sling will also count apps**/hr/job/html.jsp** as a match, if it finds it.

Here is the priority

**html.jsp🡪 jobs.jsp🡪GET.jsp**

**SLING SCRIPT RESOLUTION USING SELECTORS**

* **RESOURCE URL : /content/homepage.india.html**
* **SLING RESOURCETYPE : /**company/components/page/homePageComp
* **SELECTOR** : india
* EXTENSION : html

1. india.html. jsp
2. india. jsp
3. html. Jsp
4. homePageComp.jsp
5. GET.jsp

**EXAMPLE 2**

Let's consider the following script paths for a request of a resource whose resource type is **sling\sample** and the request selectors are **print.a4** and the request extension is html:

(0) GET.jsp

(1) sample.jsp

(2) html.jsp

(3) print.jsp

(4) print/a4.jsp

(5) print.html.jsp

(6) print/a4.html.jsp

(7) a4.html.jsp

(8) a4/print.html.jsp

**print/a4.html.jsp🡪 print/a4.jsp 🡪 print.html.jsp 🡪 print.jsp 🡪 html.jsp 🡪 sample.jsp 🡪 GET.jsp**

The priority of script selection would be (starting with the best one): (6) - (4) - (5) - (3) - (2) - (1) - (0). Note that (4) is a better match than (5) because it matches more selectors even though (5) has an extension match where (4) does not. (7) is not a candidate because it does not include the first selector (print) and (8) is not a candidate because it has the wrong order of selectors.

### SECURITY

#### TO CHANGE PASSWORDS: -

**Step 1 : Login into CRX Repo:** [**http://localhost:4506/crx/explorer/index.jsp**](http://localhost:4506/crx/explorer/index.jsp)

|  |  |
| --- | --- |
| **Option 1:** Click on Content Explorer | **Option 2:-**Click “User Administration” 🡪 Find the required user(Example) 🡪 change password link |

#### Creation of User Group and Permission

URL <http://localhost:4502/useradmin>

Question : How usually we create a user and grant permission to it?

1. We can create user group
2. Assign the permission to that group
3. Create a user/ users
4. Then assign user group to the user.

**Creating a user group**

|  |  |
| --- | --- |
|  |  |

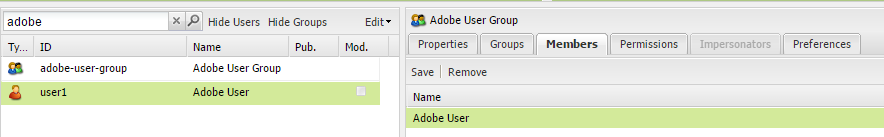
**Assigning permission to the user group**

|  |  |
| --- | --- |
|  | **Group has Read permission on content folder** |

**Creating an User**

|  |  |
| --- | --- |
|  |  |

**Assigning the group to the user :** Then in that case user has same permission as user group has.



**Let’s discuss a unique user case**

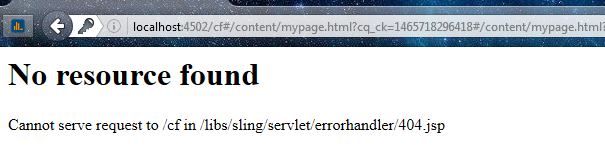
Let’s create 2 user groups

1. allowaccess
2. denyaccess

* allowaccess group has read permission on content/mypage file
* denyaccess user group has denied read access permission on content/mypage folder

Now let’s create a user and assign the user in both the user group. This will lead to conficting permission for the user. Then what will happen in that scenario?

So when the user tries to access anything in content folder he will get 404 error.



**So the deny access permission on the path always takes the priority. Now to resolve it?**

|  |  |
| --- | --- |
| Go to the explorer : <http://localhost:4502/crx/explorer/index.jsp>  Explorer to mypage file. Click on Security 🡪Access Control Editor – As shown below |  |
| As shown in Local Access Control Policy Tab  Denyaccess group has higher precedence over allowacess that the reason “user1” is getting 404 error for the page |  |
| * To resolve the issue Drag and Drop allowaccess before denyacess. Click 🡪Apply🡪 OK. This will resolve the issue of conficted permission for a user.   **Note :** Lower in the above list has higher precedence **,** in terms of permission  **Why?- \*! 🡪allowed/denied** |  |

#### Assigning Permission to users programmatically

Use case: Suppose AEM server is connected to a LDAP server, then in that case we can we can read the users from the LDAP server or when when a new user is created in the LDAP server and then set the user’d permission accordingly. We can place the code of granting permission in a listener, which can listen when a new user created or when user is being fetched from the LDAP server.

**Desciption: The below code will set permission for a particular user a certain path**

**package** com.workflow.impl;

**import** java.util.NoSuchElementException;

**import** javax.jcr.RepositoryException;

**import** javax.jcr.Session;

**import** javax.jcr.security.AccessControlList;

**import** javax.jcr.security.AccessControlManager;

**import** javax.jcr.security.AccessControlPolicyIterator;

**import** javax.jcr.security.Privilege;

**import** org.apache.sling.jcr.api.SlingRepository;

**import** org.apache.felix.scr.annotations.Activate;

**import** org.apache.felix.scr.annotations.Component;

**import** org.apache.felix.scr.annotations.Reference;

**import** org.apache.jackrabbit.api.security.JackrabbitAccessControlList;

**import** org.apache.jackrabbit.api.security.user.Authorizable;

**import** org.apache.jackrabbit.api.security.user.UserManager;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

@Component

**public** **class** ModifyPermissions {

**private** **static** **final** String ***CONTENT\_GEOMETRIXX\_FR*** = "/content/geometrixx/fr";

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(ModifyPermissions.**class**);

@Reference

**private** SlingRepository repo;

@Activate

**protected** **void** activate() {

***LOGGER***.info("ModifyPermissions activated");

modifyPermissions();

}

**private** **void** modifyPermissions() {

Session adminSession = **null**;

**try** {

adminSession = repo.~~loginAdministrative~~(**null**);

UserManager userMgr = ((org.apache.jackrabbit.api.JackrabbitSession) adminSession)

.getUserManager();

AccessControlManager accessControlManager = adminSession

.getAccessControlManager();

Authorizable denyAccess = userMgr.getAuthorizable("deny-access");

AccessControlPolicyIterator policyIterator = accessControlManager

.getApplicablePolicies(***CONTENT\_GEOMETRIXX\_FR***);

AccessControlList acl;

**try** {

acl = (JackrabbitAccessControlList) policyIterator

.nextAccessControlPolicy();

} **catch** (NoSuchElementException nse) {

acl = (JackrabbitAccessControlList) accessControlManager

.getPolicies(***CONTENT\_GEOMETRIXX\_FR***)[0];

}

Privilege[] privileges = { accessControlManager

.privilegeFromName(Privilege.***JCR\_READ***) };

acl.addAccessControlEntry(denyAccess.getPrincipal(), privileges);

accessControlManager.setPolicy(***CONTENT\_GEOMETRIXX\_FR***, acl);

adminSession.save();

} **catch** (RepositoryException e) {

***LOGGER***.error("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Repo Exception", e);

} **finally** {

**if** (adminSession != **null**)

adminSession.logout();

}

}

}

### QUERYBUILDER API

|  |  |
| --- | --- |
| Find all pages and order them by last modified | type=cq:Page  orderby=@jcr:content/cq:lastModified |
| Find all pages and order them by last modified, but descending | type=cq:Page  orderby=@jcr:content/cq:lastModified  orderby.sort=desc |
| Fulltext search, ordered by score | fulltext=Management  orderby=@jcr:score  orderby.sort=desc |
| Search for pages tagged with a certain tag | type=cq:Page  tagid=marketing:interest/product  tagid.property=jcr:content/cq:tags |
| Search under multiple paths (using groups) | fulltext=Management  **group.p.or=true**  group.1\_path=/content/geometrixx/en/company/management  group.2\_path=/content/geometrixx/en/company/bod |
| Searching for all pages of a given template, using the cq:template property | type=cq:PageContent  property=cq:template  property.value=/apps/geometrixx/templates/homepage  Drawback : This returns that the jcr:content nodes of the pages, not the pages themselves |
| Searching for all **pages** of a given template, using the cq:template property | type=cq:Page  property=jcr:content/cq:template  property.value=/apps/geometrixx/templates/homepage |
| **SEARCH FOR MULTIPLE PROPERTIES**  When using the property predicate multiple times, you have to add the number prefixes again: | type=cq:Page  1\_property=jcr:content/cq:template  1\_property.value=/apps/geometrixx/templates/homepage  2\_property=jcr:content/jcr:title  2\_property.value=English |
| **SEARCH FOR MULTIPLE PROPERTY VALUES**  To avoid big groups when you want to search for multiple values of a property ("A" or "B" or "C"), you can provide multiple values to the property predicate: | property=jcr:title  property.1\_value=Products  property.2\_value=Square  property.3\_value=Events |
| **REFINING WHAT IS RETURNED**  By default, the **QueryBuilder JSON Servlet** will return a default set of properties for each node in the search result (e.g. path, name, title, etc.). In order to gain control over which properties are returned, you can do one of the following: | Specify p.hits=full, in which case all properties will be included for each node:  property=jcr:title  property.value=Triangle  p.hits=full  Use p.hits=selective and specify the properties you want to get in p.properties, separated by a space:  property=jcr:title  property.value=Triangle  p.hits=selective  p.properties=sling:resourceType jcr:primaryType |
| Another thing you can do is include child nodes in the QueryBuilder response. In order to do this you need to specify p.nodedepth=**n**, where **n** is the number of levels you want the query to return. Note that, in order for a child node to be returned, it must be specified by the properties selector (p.hits=full). Example: | property=jcr:title  property.value=Triangle  **p.hits=full**  **p.nodedepth=5** |

### MULTI SITE MANAGER

Multi Site Manager (MSM) enables you to easily manage multiple web sites that share common content. MSM lets you define relations between the sites so that content changes in one site are automatically replicated in other sites. For example, web sites are often provided for international audiences such that most of the content is common for each country, and a subset of the content is specific to each country.

MSM reduces the time it takes to manage your websites and increases the re-use of common content:

* Efficiently manage different language versions of a website.
* Automatically update one or more sites based on a source site:
  1. Enforce a common base structure and use common content across multiple sites.
  2. Maximize the use of available resources.
  3. Maintain a common look and feel.
  4. Focus efforts on managing the content that differs between the sites.

The content and structure that a site inherits from the source site can be managed at a page and paragraph level. Generally, using MSM involves performing the following tasks:

* Identifying the source content.
* Identifying the location of the copy.
* Configuring what items of source content are automatically updated in the copy.
* Configuring when content updates occur in the copy.

**What is Live Copy?**

MSM enables you to create a copy of an existing site and automatically update the copy with changes that occur to the source site. The copy is called a Live Copy. For each Live Copy, a Rollout Config determines how content is automatically updated. A Rollout Config consists of the following items:

* An event that triggers the update, such as a change to the content of the source page.
* One or more actions that occur, such as updating the content on the Live Copy.

**Live Copy pages can use any page as the source. A Live Copy can also use a blueprint as the source**. Blueprints define a source website for Live Copy pages and consist of the following items:

* The path of the root page of a website. The path defines the pages that are copied as Live Copy pages.
* One or more Rollout Configs. Live Copy pages made from blueprints inherit the blueprint Rollout Configs.

**Creating a Live Copy Based on an Existing Branch or Page**

Create a Live Copy of any page and its child pages, or of a single page. When you create the Live Copy, you can optionally specify the rollout configurations to use for automatically updating the content:

* The selected rollout configurations apply to all of the Live Copy pages that are created for the source page and its child pages.
* If you specify no rollout configurations, either the system default rollout configuration is used, or the default rollout configuration for the branch is used.

**The following procedure creates a Live Copy using the classic UI.**

1. Open the Websites console.
2. Select the folder or page below which you want to locate the Live Copy pages.
3. Click New 🡪 New Live Copy.
4. In the Source selection tab, define the Live Copy:

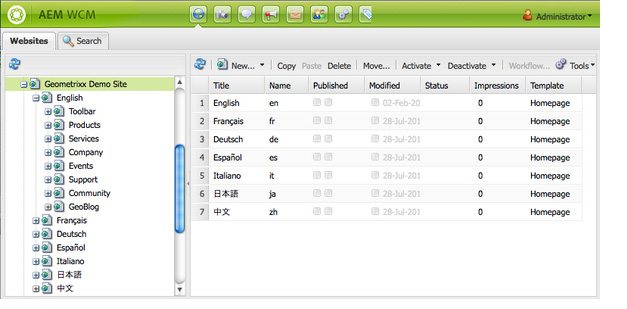
* Title: The title of the root Live Copy page, e.g. My Geometrixx Live Copy
* Name: The name of the page node, e.g. mygeometrixx\_livecopy
* Live Copy From: Browse to select the page to use as the source of the Live Copy, e.g. /content/geometrixx
* Exclude sub pages: Select this option to exclude the child pages from the Live Copy.

1. On the Sync config tab, specify one or more Rollout Configs (Mostly **standard rollout** is used) to use for the Live Copy.
2. Click Create.

#### CREATING BLUEPRINT

Create a blueprint to identify an existing web site that you want to use as the source for one or more Live Copy pages. **When you create a blueprint, you select a template that defines the internal structure of the blueprint**. The default Blueprint Template assumes that the source web site has the following characteristics:

* The web site has a root page.
* The immediate child pages of the root are language branches of the web site. The Name of each page is a language code. When creating a Live Copy, the languages are presented as optional content to include in the copy.
* **Each language page contains one or more child pages. When creating a Live Copy, child pages are presented as a chapter that you can include in the copy.** For example, the Geometrixx Demo Site contains language folders named en, fr, de, es, it, ja, and zn. The language folders contain chapters such as Toolbar, Products and Services etc...



|  |  |
| --- | --- |
| **To create a blueprint:**   1. Open the Tools console. 2. Select Tools, then **MSM Control Center**. Click New... in the top-middle toolbar. 3. In the Create Page dialog, define the blueprint:    1. Title, the page title, e.g. My Blueprint    2. Name, the page (node) name, e.g. myblueprint    3. Select the Blueprint Template    4. Click Create. 4. Open the newly created page and click the Edit button beside Settings. 5. In the Blueprint Settings dialog, define the blueprint:    1. Name, the blueprint name that was defined earlier can be changed.    2. Description, e.g. This is my blueprint    3. Source Path: set the path of the blueprint, e.g. /content/geometrixx    4. Thumbnail Image (optional): this thumbnail will appear in the live copy creation. |  |

#### CREATING A LIVE COPY BASED ON A BLUEPRINT

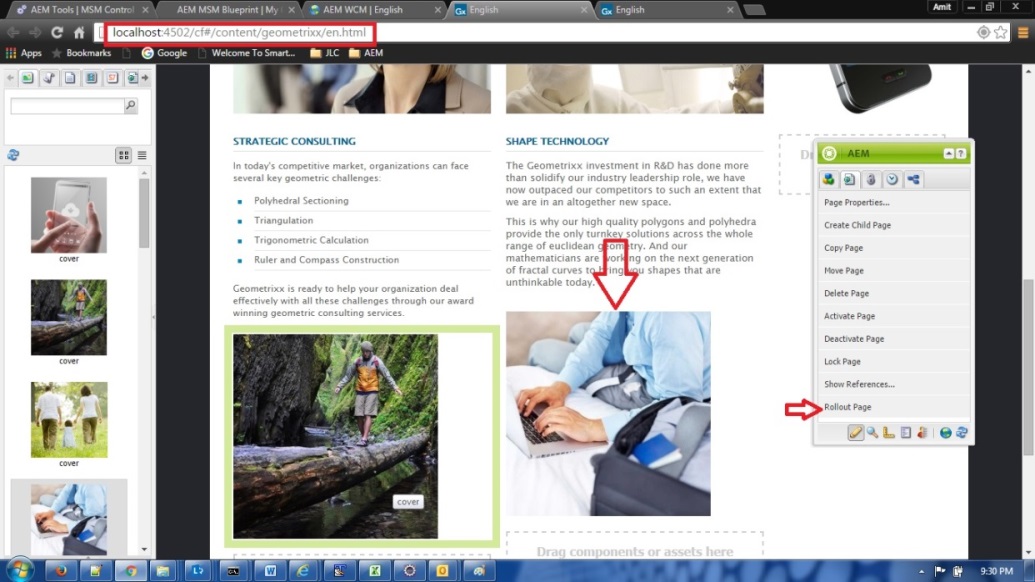
Create a Live Copy of a blueprint to create a site based on the blueprint content. When you create a Live Copy of a blueprint, you select one or more language branches of the blueprint source to copy and then you select the chapters to copy from the language branches. (See the previous Creating a Blueprint section.).If you omit some language branches or chapters from the Live Copy, you can add them later. (See Creating a Live Copy Within a Live Copy.)

The Rollout Configs that are used for the Live Copy are inherited from the blueprint. However, when creating the Live Copy, you can override the blueprint and specify the Rollout Configs to use. The following procedure creates a Live Copy from a blueprint using the classic UI.

1. Open the Websites console.
2. Select the folder or page below which you want to locate the Live Copy pages.
3. Click New > New Site.
4. In the Create Site dialog, define:
   1. Title: The title of the root page of the Live Copy, e.g. My Geometrixx Live Copy
   2. Name: The name of the page node, e.g. mygeometrixx\_livecopy

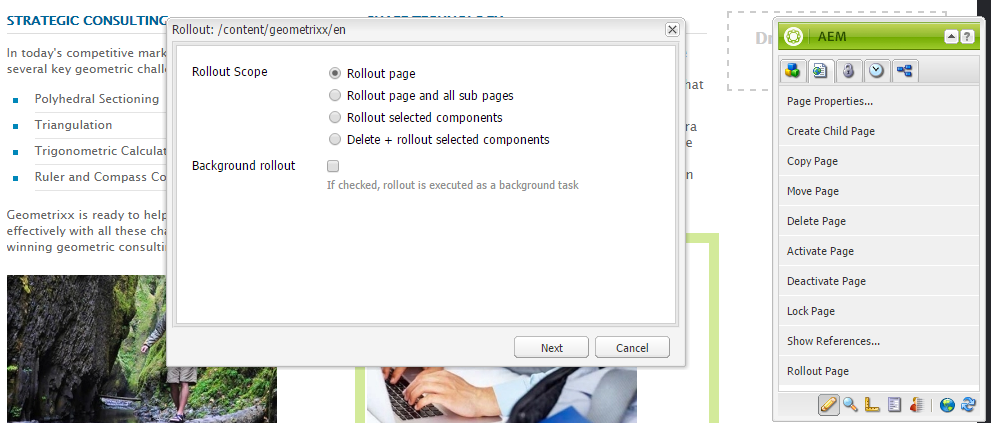
|  |  |
| --- | --- |
| 1. Select the blueprint to use as the source and click Next. | 1. Select the language versions of the blueprint site to include in the Live Copy, and click Next |
| 1. Select the chapters to include in the Live Copy and click Next. | 1. Specify values for the following properties to define the Live Copy      * Site Owner: choose an existing user. Note: The Site Owner entry is a descriptive property and does not affect feature behavior. * Live Copy: Clearing this option creates a static copy of the blueprint. * Rollout Configs: Select one or more Rollout Configs to override the blueprint. Specify no Rollout Configs to use those that the blueprint defines |
| 1. Click Next. 2. Click Create Site. |  |

When we create a live copy from Blueprint. Then the Source Website 🡪 Used to create a Blueprint (source Website is used as parent) 🡪 One or more website are created based on the Blue print. So when we update the source website content (i.e in Blue print), the website created with that blueprint gets updated just by Rollout of the parent website

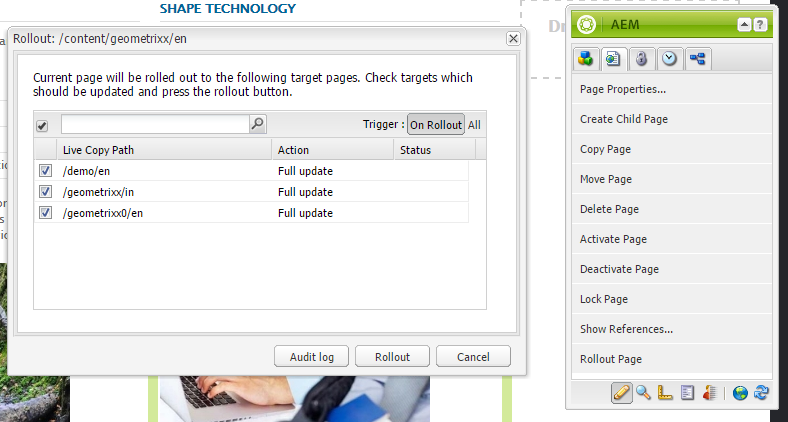


**A New Content has been added in the parent website (By which Blue print is created – To propagate the same changes in the child website we need to rollout the page as shown below**

Step 1:



Step 2:



#### CREATING A LIVE COPY WITHIN A LIVE COPY

CQ enables you to create a Live Copy within another Live Copy. The process is similar to the one described in Creating a Live Copy Based on an Existing Branch or Page, except that you select a page of an existing Live Copy as the parent of the new Live Copy.

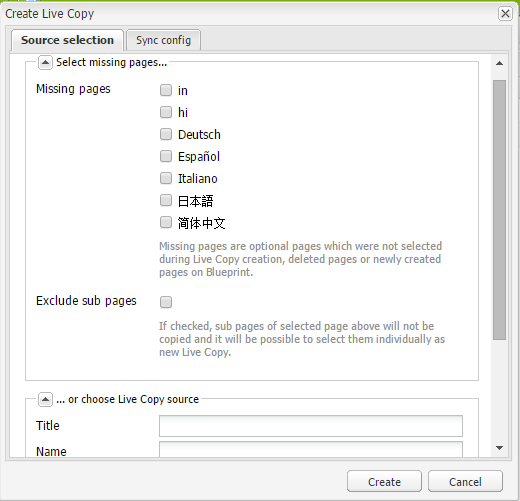
When you insert a Live Copy into the Live Copy of a blueprint, you can insert any language versions or chapters that were not included when the Live Copy of the blueprint was originally created. For example, a Live Copy of the Geometrixx Demo Site blueprint is created, but only the English version is included in the LiveCopy, and only a some of the English chapters were included.

In this case, you can insert the content of the Geometrixx Demo Page blueprint that was not included:

* Select the root page of the Live Copy to insert one or more of the language versions.
* Select the English page of the Live Copy to insert any of the chapters that were not included.

When you are inserting a Live Copy into a Live Copy of a blueprint, and the Live Copy does not include all content of the blueprint, the Create Live Copy dialog includes a collapsible fieldset called select missing pages. You use this fieldset to add the missing content.

To insert a Live Copy of other content, use the Or Choose Live Copy Source fieldset to select the source content as you normally would.

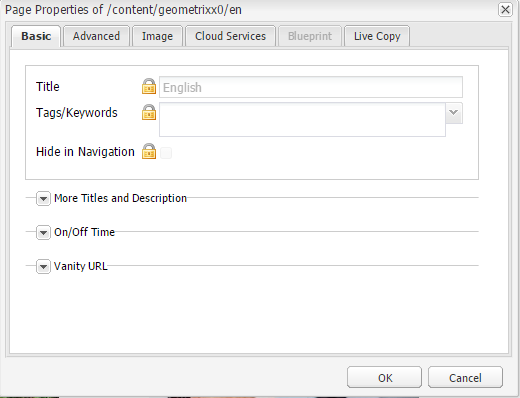


#### Overriding Live Copy Configurations

You can override the Live Copy configurations for any page that inherits the root Live Copy page, as well as for any component on a Live Copy page.

**Overriding Live Copy Inheritance for Components On a Page**

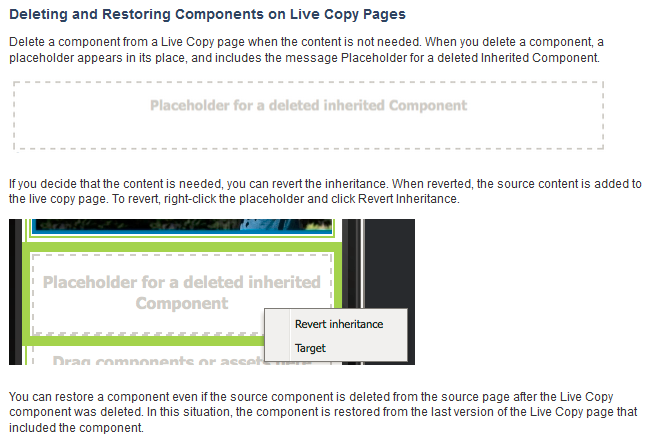
When you open a Live Copy page, a closed or an open lock icon represents the Live Copy inheritance status for each component. The inheritance for a component can be disabled or enabled.



|  |  |
| --- | --- |
| **Live Copy Inheritance Status** | **Description** |
| Enabled | Modifications to the source component are propagated to the Live Copy component. This is the default status when a Live Copy page is created.   The status icon is a locked padlock and the component border is green when it is highlighted. |
| Cancelled | Deletion, update or reordering of the source component does not affect the Live Copy component. The component has this status under the following conditions: 1. The Live Copy component has been modified, for example the text or styles are edited. Inheritance is disabled. 2. When a Live Copy component container (such as a paragraph system) has this status, the order of the components inside the container is not inherited from the source. 3. The status icon is an unlocked padlock and the component border is green when it is highlighted. |

#### CANCELLING INHERITANCE IN LIVE COPY.

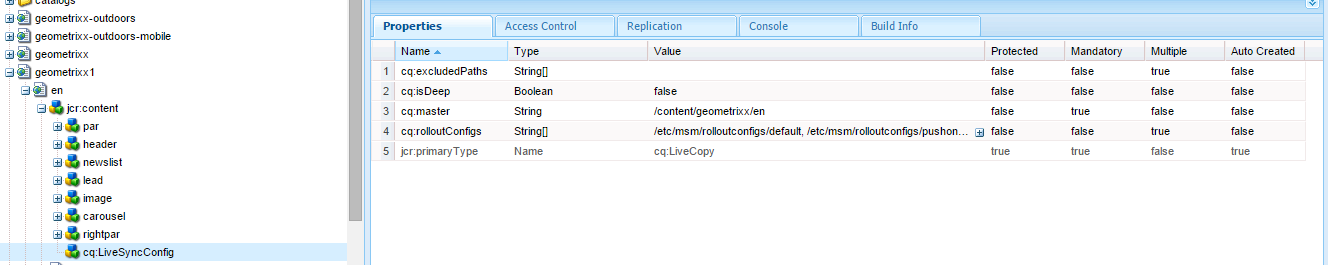
|  |  |
| --- | --- |
| 1. Open the Live Copy page. 2. In Sidekick, click the Live Copy Status button to view the status of all the components of the page. 3. This button appears only when you are editing a Live Copy page. 4. Locate the component in which you are interested and note the Live Copy status icon. 5. Click the Live Copy status icon to toggle inheritance, and then click **yes** in the dialog to confirm the change.   Note: You can also change the Live Copy inheritance status using the component Edit dialog box. The status icon appears in the bottom left-hand corner of the edit dialog. |  |



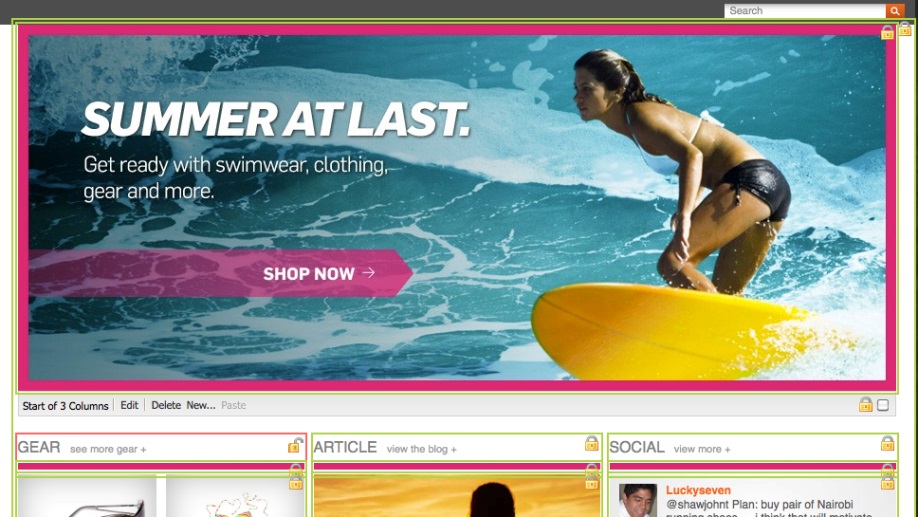
|  |  |
| --- | --- |
|  | **We can open the page properties dialog of the Live Copy page using either of the following methods:**   1. On the Siteadmin page, right-click that page and click Properties. 2. With the pag open in Edit mode, on the Page tab of SideKick click Page Properties.   Configurations in Page Propeties   1. **Live Copy Inheritance**: Select this option to cause child pages to inherit the live copy configurations of the current page. Clear this option to disable live copy inheritance for child pages so that child pages and their components no longer behave as live copy pages. Note that when cleared, the current page still behaves as a live copy. 2. **Live Copy Suspended**: Select this option to prevent the synchronization actions of the Rollout Configs from occurring for this page. Optionally, you can also suspend rollouts for child pages. 3. **Rollout Config**: Change the rollout configurations that are applied to the page. |

#### Overriding live copy configurations

When you create a Live Copy, the **Live Copy configuration is saved to the properties of the root page of the copied pages**. All child pages of the root page inherit the Live Copy configurations. The Live Copy configuration is applied to all components on each of the copied pages. Components within container components inherit the status of the container component. You can override the Live Copy configurations for any page that inherits the root Live Copy page, as well as for any component on a Live Copy page.



Overriding Live Copy Inheritance for Components On a Page 🡪When you open a Live Copy page, a closed or an open lock icon represents the Live Copy inheritance status for each component. The inheritance for a component can be disabled or enabled.



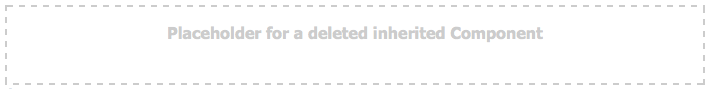
1. Open the Live Copy page.
2. In Sidekick, click the Live Copy Status button to view the status of all the components of the page. (Fig A)
3. This button appears only when you are editing a Live Copy page.
4. Click the Live Copy status icon to toggle inheritance, then click Yes in the dialog to confirm the change.

**Note: You can also change the Live Copy inheritance status using the component Edit dialog box. The status icon appears in the bottom left-hand corner of the edit dialog.(Fig B)**

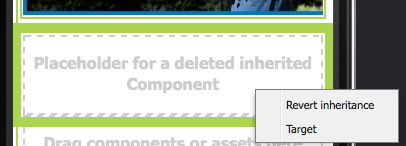
|  |  |
| --- | --- |
| **Fig A** | **Fig B** |

**Deleting and Restoring Components on Live Copy Pages**

Delete a component from a Live Copy page when the content is not needed. When you delete a component, a placeholder appears in its place, and includes the message **Placeholder for a deleted Inherited Component**.



If you decide that the content is needed, you can revert the inheritance. When reverted, the source content is added to the live copy page. To revert, right-click the placeholdfer and click **Revert Inheritance**.



You can restore a compo*n*ent even if the source component is deleted from the source page after the Live Copy component was deleted. **In this situation, the component is restored from the last version of the Live Copy page that included the component**.

|  |  |
| --- | --- |
| For any page in a Live Copy, you can configure the following properties to override the inherited Live Copy configurations: | file |

|  |  |
| --- | --- |
| **Live Copy Inheritance** | Select this option to cause child pages to inherit the live copy configurations of the current page. Clear this option to disable live copy inheritance for child pages so that child pages and their components no longer behave as live copy pages. Note that when cleared, the current page still behaves as a live copy. |
| **Live Copy Suspended** | Select this option to prevent the synchronization actions of the Rollout Configs from occurring for this page. Optionally, you can also suspend rollouts for child pages. |
| **Rollout Config** | Change the rollout configurations that are applied to the page. |

**SPECIFYING DEFAULT ROLLOUT CONFIGURATIONS**

To use a rollout configuration, you apply it to blueprint and/or live copy pages. The following logic determines which rollout configuration to use for a live copy page:

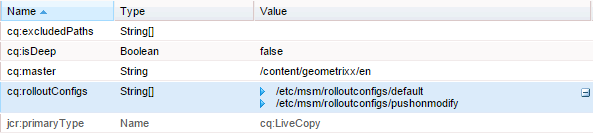
* When a live copy page has a rollout configuration, that configuration is used.
* When a live copy page does not have a rollout configuration, the rollout configuration of the corresponding blueprint page is used.
* When both the live copy page and the blueprint page have no rollout configuration, the rollout configuration that applies to the live copy page's parent page is used.
* When the rollout configuration of the live copy's parent page cannot be determined, the **default rollout configuration** is used.

**The default configuration can either be:**

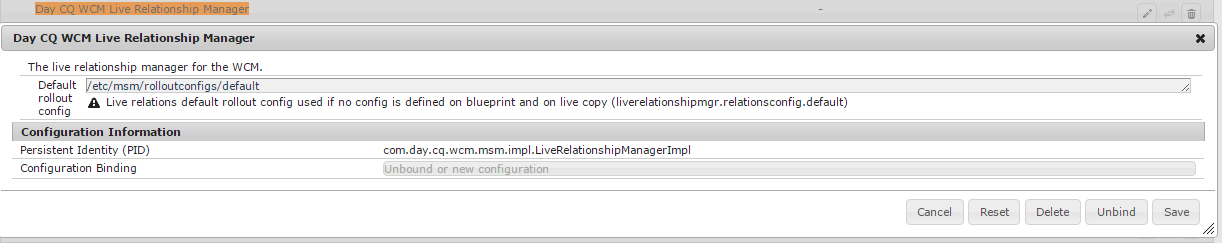
1. The default rollout configuration for the branch if it is set.
2. The system default rollout configuration if no configuration is set.
3. **Setting a Default Rollout Configuration for a Branch 🡪**To set a default rollout configuration for a branch:

In CRXDE, go to the jcr:content node of the page.

* Create the following node below **jcr:content**:
  + name: **cq:LiveSyncConfig**
  + type: **cq:LiveCopy**
* Create the following property for the **cq:LiveSyncConfig** node:
  + name: **cq:rolloutConfigs**
  + type: **String[]**
* value: the path to the rollout configuration, e.g.: **/etc/msm/rolloutconfigs/default**
* Save the changes.



1. **Changing the System Default Rollout Configuration**
2. The system default rollout configuration is set to: /etc/msm/rolloutconfigs/default.
3. When working with AEM there are several methods of managing the configuration settings for such services; see Configuring OSGi for full details. For example: Go to the Configuration tab of the Web Console: <http://localhost:4502/system/console/configMgr>
4. Open the **Day CQ WCM Live Relationship Manager** configuration window.
5. In Default rollout config, set the path of the default rollout configuration.
6. Click Save.



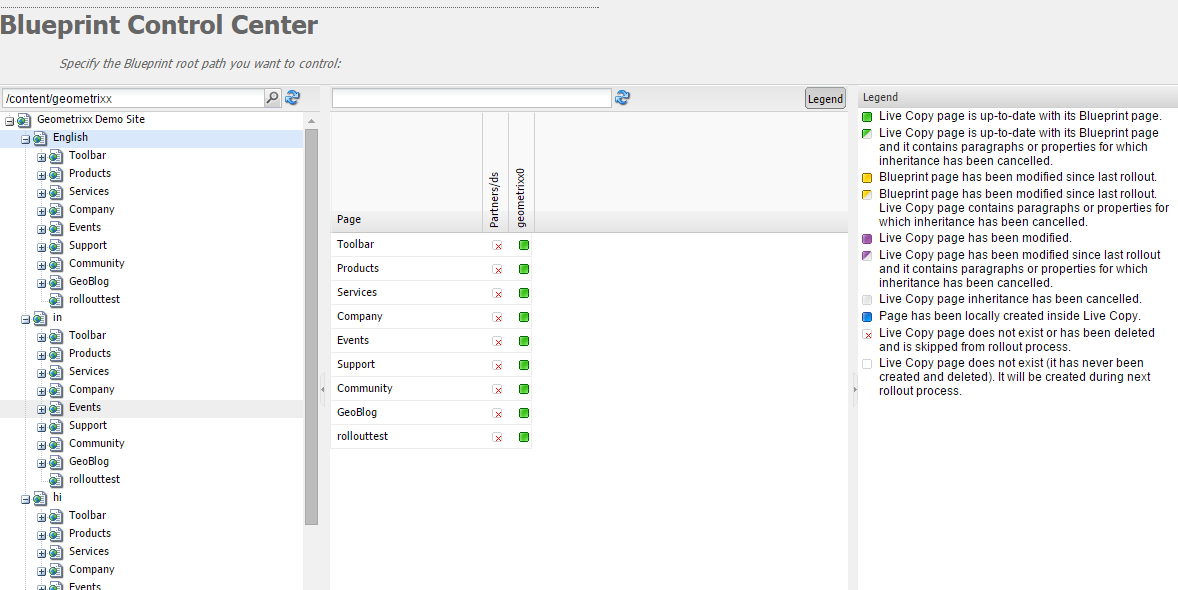
#### Managing a Blueprint and its Live Copies: Blueprint Control Center

The Blueprint Control Center enables you to efficiently manage a blueprint and its live copies. You can define the blueprint, navigate

to any blueprint page and view the status of all the related live copies. You can also filter, based on a path, the live copies that are displayed. A context menu enables you to perform all necessary actions on a page (like rolling out or restoring a page) and also to access information about the current state of a page.

**To open the Control Center:**

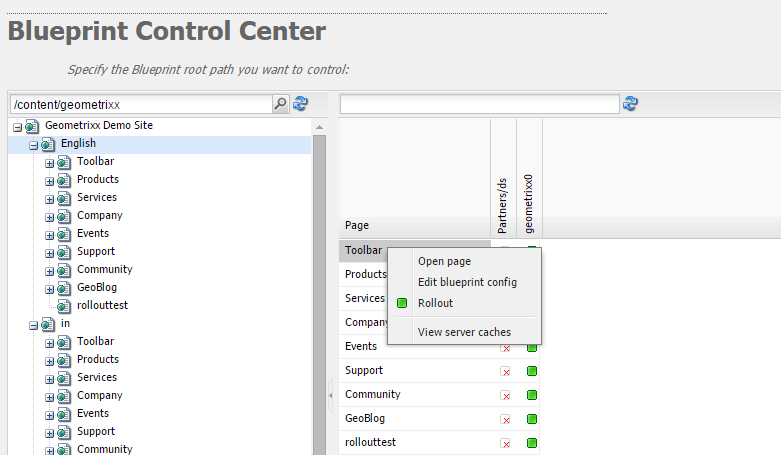
* Go to the Tools console. (<http://localhost:4502/miscadmin>)
* Double-click the MSM Control Center folder.
* Specify the blueprint path in the top left path field and click the refresh button.
* In the left panel, navigate to a blueprint page. The related live copy pages are displayed in the middle panel with a status icon.
* Note: navigating through the blueprint tree does not update the top left blueprint field.
* In the pathfield in the middle panel, specify the live copy path to filter the live copies and click the refresh button.
* On the top right, click Legend to display the status icons and their meanings.



The Page column in the central panel is the title of the Blueprint page. Columns to the right of the Page column contain status indicators of the Live Copy pages.

**MANAGING BLUEPRINTS WITH THE CONTROL CENTER**

When right-clicking a blueprint page (either in the left panel or in the first column of the central panel), the following context menu is displayed, shown below



**Creating a new synchronisation action**

|  |  |
| --- | --- |
| **Terms** | **Defination** |
| **Blueprint** | A Blueprint defines the pages from which a live copy can inherit content. The use of a blueprint is optional but is required to push modifications to live copies that are inheriting from this blueprint or to define the default rollout configuration for the live copy that is in relation with the blueprint. |
| **LiveCopy** | A LiveCopy is the configuration of the relationship (LiveRelationship) between a page and its blueprint page. Use the LiveCopy class to access to the path of the page, the path of the blueprint page, the Rollout Configs, and whether child pages are also included in the LiveCopy. |
| **LiveRelationship** | The LiveRelationship represents the relationship between a resource and a blueprint. LiveRelationship objects provide access to the Rollout Configs, LiveCopy, and LiveStatus objects of the relationship. It also provides access to the paths of the target and source pages. For example, a Live Copy is created in /content/copy from the blueprint at /content/geometrixx. The resources /content/geometrixx/en/jcr:content and /content/copy/en/jcr:content form a relationship. |
| **LiveStatus** | LiveStatus objects provide access to the runtime status of a LiveRelationship. Use to query the synchronization status of a Live Copy. |
| **LiveAction** | A LiveAction is an action that is executed on each resource that is involved in the rollout. |
| **LiveActionFactory** | Creates LiveAction objects given a LiveAction configuration. Configurations are stored as resources in the repository. |

Create custom synchronisation actions to use with your Rollout Configs. Create a syncrhonization action when the installed actions(ex. Push on modify) do not meet your specific application requirements. To do so, create two classes:

* An implementation of the **com.day.cq.wcm.msm.api.LiveAction** interface that performs the action.
* An OSGI component that implements the **com.day.cq.wcm.msm.api.LiveActionFactory** interface and creates instances of your LiveAction class.

The LiveAction class is not registered as an OSGi service. Typically, the LiveAction class is used only by one LiveActionFactory so it is convenient to define the LiveAction class as a static nested class of the LiveActionFactory class.

LiveAction classes include the following methods:

* **getName**: Returns the name of the action The name is used to refer to the action, for example in rollout configurations.
* **execute**: Performs the tasks of the action.

LiveActionFactory classes include the following members:

* **LIVE\_ACTION\_NAME**: A field that contains the name of the associated LiveAction. This name must coincide with the value that is returned by the getName method of the LiveAction class.
* **createAction**: Creates an instance of the LiveAction. The optional Resource parameter can be used to provide configuration information.
* **createsAction**: Returns the name of the associated LiveAction.

Note that when creating a rollout configuration, the name of the LiveAction is the name used for the cq:LiveSyncAction node that you add to a rollout configuration. (See Creating a Rollout Configuration.)

**Accessing the LiveAction Configuration Node**

Use the LiveAction configuration node in the repository to store information that affects the runtime behaviour of the LiveAction instance. The node in the repository that stores the LiveAction configuration is available to the LIveActionFactory object at runtime. Therefore, you can add properties to the configuration node to and use them in your LiveActionFactory implementation as needed.

For example, a LiveAction needs to store the name of the blueprint author. A property of the configuration node includes the property name of the blueprint page that stores the information. At runtime, the LiveAction retrieves the property name from the configuration, then obtains the property value.

The parameter of the LiveActionFactory.createAction method is a Resource object. This Resource object represents the cq:LiveSyncAction node for this Live Action in the Rollout Config. (See Creating a Rollout Configuration.) As usual when using a configuration node, you should adapt it to a ValueMap object:

public LiveAction **createAction**(Resource resource) throws WCMException {

ValueMap config;

if (resource == null || resource.adaptTo(ValueMap.class) == null) {

config = new ValueMapDecorator(Collections.<String, Object>emptyMap());

} else {

config = resource.adaptTo(ValueMap.class);

}

return new MyLiveAction(config, this);}

**The following LiveActionFactory class implements a LiveAction that logs messages about the source and target pages, and copies the cq:lastModifiedBy property from the source node to the target node. The name of the Live Action is exampleLiveAction.**

|  |
| --- |
| @Component(metatype = false)  @Service  public class ExampleLiveActionFactory implements **LiveActionFactory**<LiveAction> {  @Property(value="exampleLiveAction")  static final String actionname = LiveActionFactory.LIVE\_ACTION\_NAME;  public LiveAction createAction(Resource config) {  ValueMap configs;  /\* Adapt the config resource to a ValueMap \*/  if (config == null || config.adaptTo(ValueMap.class) == null) {  configs = new ValueMapDecorator(Collections.<String, Object>emptyMap());  } else {  configs = config.adaptTo(ValueMap.class);  }  return new ExampleLiveAction(actionname, configs);  }  public String createsAction() { return actionname; }  /\*\*\*\*\*\*\*\*\*\*\*\*\* LiveAction \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  private static class ExampleLiveAction implements LiveAction {  private String name;  private ValueMap configs;  private static final Logger log = LoggerFactory.getLogger(ExampleLiveAction.class);  public ExampleLiveAction(String nm, ValueMap config){  name = nm; configs = config;  }  public void **execute**(Resource source, Resource target, LiveRelationship liverel, boolean autoSave, boolean isResetRollout)  throws WCMException {  String lastMod = null;  log.info(" \*\*\* Executing ExampleLiveAction \*\*\* ");  /\* Determine if the LiveAction is configured to copy the cq:lastModifiedBy property \*/  if ((Boolean) configs.get("repLastModBy")){  /\* get the source's cq:lastModifiedBy property \*/  if (source != null && source.adaptTo(Node.class) != null){  ValueMap sourcevm = source.adaptTo(ValueMap.class);  lastMod = sourcevm.get(com.day.cq.wcm.api.NameConstants.PN\_PAGE\_LAST\_MOD\_BY, String.class);  }  /\* set the target node's la-lastModifiedBy property \*/  Session session = null;  if (target != null && target.adaptTo(Node.class) != null){  ResourceResolver resolver = target.getResourceResolver();  session = resolver.adaptTo(javax.jcr.Session.class);  Node targetNode;  try{  targetNode=target.adaptTo(javax.jcr.Node.class);  targetNode.setProperty("la-lastModifiedBy", lastMod);  log.info(" \*\*\* Target node lastModifiedBy property updated: {} \*\*\*",lastMod);  }catch(Exception e){  log.error(e.getMessage());  } }  if(autoSave){  try { session.save();  } catch (Exception e) {  try { session.refresh(true);  } catch (RepositoryException e1) { e1.printStackTrace();  }  e.printStackTrace();  } } } }  public String getName() { return name; }  /\*\*\*\*\*\*\*\*\*\*\*\*\* Deprecated \*\*\*\*\*\*\*\*\*\*\*\*\*/  @Deprecated  public void execute(ResourceResolver arg0, LiveRelationship arg1, ActionConfig arg2, boolean arg3) throws WCMException {  }  @Deprecated  public void execute(ResourceResolver arg0, LiveRelationship arg1, ActionConfig arg2, boolean arg3, boolean arg4) throws WCMException {  }  @Deprecated  public String getParameterName() { return null; }  @Deprecated  public String[] getPropertiesNames() { return null; }  @Deprecated  public int getRank() { return 0; }  @Deprecated  public String getTitle() { return null; }  @Deprecated  public void write(JSONWriter arg0) throws JSONException { }  }  } |

Create custom synchronisation actions to use with your Rollout Configs. Create a syncrhonization action when the installed actions do not meet your specific application requirements. To do so, create two classes:

* An implementation of the **com.day.cq.wcm.msm.api.LiveAction** interface that performs the action.
* An OSGI component that implements the com.day.cq.wcm.msm.api.LiveActionFactory interface and creates instances of your LiveAction class.

The LiveAction class is not registered as an OSGi service. Typically, the LiveAction class is used only by one LiveActionFactory so it is convenient to define the LiveAction class as a static nested class of the LiveActionFactory class.

LiveAction classes include the following methods:

* getName: Returns the name of the action The name is used to refer to the action, for example in rollout configurations.
* execute: Performs the tasks of the action.

LiveActionFactory classes include the following members:

LIVE\_ACTION\_NAME: A field that contains the name of the associated LiveAction. This name must coincide with the value that is returned by the getName method of the LiveAction class.

createAction: Creates an instance of the LiveAction. The optional Resource parameter can be used to provide configuration information.

createsAction: Returns the name of the associated LiveAction.

Note that when creating a rollout configuration, the name of the LiveAction is the name used for the cq:LiveSyncAction node that you add to a rollout configuration. (See Creating a Rollout Configuration.)

Accessing the LiveAction Configuration Node

Use the LiveAction configuration node in the repository to store information that affects the runtime behaviour of the LiveAction instance. The node in the repository that stores the LiveAction configuration is available to the LIveActionFactory object at runtime. Therefore, you can add properties to the configuration node to and use them in your LiveActionFactory implementation as needed.

For example, a LiveAction needs to store the name of the blueprint author. A property of the configuration node includes the property name of the blueprint page that stores the information. At runtime, the LiveAction retrieves the property name from the configuration, then obtains the property value.

The parameter of the LiveActionFactory.createAction method is a Resource object. This Resource object represents the cq:LiveSyncAction node for this Live Action in the Rollout Config. (See Creating a Rollout Configuration.) As usual when using a configuration node, you should adapt it to a ValueMap object:

### INTERNATIONALIZATION

Java and Javascript APIs enable you to internationalize strings in the following types of resources:

* Java source files.
* JSP scripts.
* Javascript in client-side libraries or in page source.
* JCR node property values used in dialogs and component configuration properties.

**INTERNATIONALIZING STRINGS IN JAVA AND JSP CODE**

The **com.day.cq.i18n** Java package enables you to display localized strings in your UI. The I18n class provides the **get method** that retrieves localized strings from the AEM dictionary. The only required parameter of the get method is the string literal in the English language. English is the default langauge for the UI. The following example localizes the word Search🡪 **i18n.get("Search");**

**Creating the I18n Java object**

The I18n class provides two constructors. How you determine the user's preferred language determines the constructor to use.

1. To present the string in the language that is specified in the user account, use the following constructor (after importing

**com.day.cq.i18n.I18n**)

**I18n i18n = new I18n(slingRequest);**🡨 The constructor uses the SlingHTTPRequest to retrieve the user's language setting.

1. To use the page locale to determine the language, **you first need to obtain the ResourceBundle for the language of the requested page:**

**Locale pageLang = currentPage.getLanguage(false);**

**ResourceBundle resourceBundle = slingRequest.getResourceBundle(pageLang);**

**I18n i18n = new I18n(resourceBundle);**

**Internationalizing a String :** Use the get method of the I18n object to internationalize a string. The only required parameter of the get method is the string to internationalize. The string corresponds with a string in a Translator dictionary. The get method looks up the string in the dictionary and returns the translation for the current language.

The first argument of the get method must comply with the following rules:

* The value must be a string literal. A variable of type String is not acceptable.
* The string literal must be express on a single line.
* The string is case-sensitive.
* i18n.get("Enter a search keyword");

**USING TRANSLATION HINTS**

**Specify the translation hint of the internationalized string to distinguish between duplicate strings in the dictionary. Use the second, optional parameter of the get method to provide the translation hint. The translation hint must exactly match the Comment property of the item in the dictionary**.

For example, the dictionary contains the string Request twice: once as a verb and once as a noun. The following code includes the translation hint as an argument in the get method:

**i18n.get("Request","A noun, as in a request for a web page");**

**Including Variables in Localized Sentences**

Include variables in the localized string to build contextual meaning into a sentence. For example, after logging into a web application, the home page displays the message "Welcome back Administrator. You have 2 messages in your inbox." The page context determines the user name and the number of messages.

**In the dictionary, the variables are represented in strings as bracketed indexes. Specify the values of the variables as arguments of the get method. The arguments are placed following the translation hint, and the indexes correspond with the order of the arguments:**

**i18n.get("Welcome back {0}. You have {1} messages.", "user name, number of messages", user.getDisplayName(), numItems);**

The internationalized string and the translation hint must exactly match the string and comment in the dictionary. You can omit the localization hint by providing a null value as the second argument.

**USING THE STATIC GET METHOD**

The I18N class defines a **static get method** that is useful when you need to localize a small number of strings. In addition to the parameters of an object's get method, the static method requires the **SlingHttpRequest** object or the **ResourceBundle** that you are using, according to how you are determining the user's preferred language:

**Use the user's language preference: Provide the SlingHttpRequest as the first parameter**.

I18n.get(slingHttpRequest, "Welcome back {}. You have {} messages.", "user name, number of messages", user.getDisplayName(), numItems);

**Use the page language: Provide the ResourceBundle as the first parameter.**

I18n.get(resourceBundle,"Welcome back {}. You have {} messages.", "user name, number of messages", user.getDisplayName(), numItems);

**INTERNATIONALIZING STRINGS IN JAVASCRIPT CODE**

The Javascript API enables you to localize strings on the client. As with Java and JSP code, the Javascript API enables you to identify strings to localize, provide localization hints, and include variables in the localized strings. The **granite.utils client library** folder provides the Javascript API. To use the API, include this client library folder on your page. Localization functions use the **Granite.I18n namespace**. **Before you present localized strings, you need to set the locale using the Granite.I18n.setLocale function.**

**The function requires the language code of the locale as an argument:** Granite.I18n.setLocale("fr");

To present a localized string, use the Granite.I18n.get function: **Granite.I18n.get("string to localize");**

**Granite.I18n.setLocale("fr");**

**Granite.I18n.get("string to localize", [variables], "localization hint");**

The function parameters are different than the Java I18n.get method:

* The first parameter is the string literal to localize.
* The second parameter is an array of values to inject into the string literal.
* The third parameter is the localization hint.

The following example uses JavaScript to localize the "Welcome back Administrator. You have 2 messages in your inbox." sentence:

**Granite.I18n.setLocale("fr");**

**Granite.I18n.get("Welcome back {0}. You have {1} new messages in your inbox.", [username, numMsg], "user name, number of messages");**

**INTERNATIONALIZING STRINGS FROM JCR NODES**

UI strings are often based on JCR node properties. For example, the jcr:title property of a page is typically used as the content of the h1 element in the page code. The I18n class provides the getVar method for localizing these strings.

The following example JSP script retrieves the jcr:title property from the repository and displays the localized string on the page:

**<% title = properties.get("jcr:title", String.class);%>**

**<h1><%=i18n.getVar(title) %></h1>**

**Specifying Translation Hints for JCR Nodes**

Similar to translation hints in the Java API, you can provide translation hints to distinguish duplicate strings in the dictionary. Provide the translation hint as a property of the node that contains the internationalized property. The name of the hint property is comprised of the name of the internationalized property name with the \_commentI18n suffix:

**${prop}\_commentI18n**

For example, a cq:page node includes the jcr:title property which is being localized. The hint is provided as the value of the property named jcr: title\_commentI18n.

#### I18N IMPLEMENTATION

Step 1: Create a node “i18n” of type 🡪**sling:folder** in apps/<project>/ directory

Step 2: Create a node of “en” of type 🡪**sling:folder** below i18n folder

|  |  |
| --- | --- |
| Language Node property details  **jcr:language = en (for english)**  **jcr:mixinTypes :mix:language** | All the jcr:language nodes has a fixed value e.g for English its en.AEM maps the language vaues from  **/libs/wcm/core/resources/languages** |

Step 3: Create a node of type sling:MessageEntry and add the below property



Step 4: To retrieve i18N value from key

|  |  |
| --- | --- |
| <%=slingRequest.getResourceBundle(slingRequest.getLocale()).getString("mkey") %> | <% I18n i18n = new I18n(slingRequest.getResourceBundle(slingRequest.getLocale())); %>  <%=i18n.get("cart")%> |

* AEM provides a console for managing the various translations of texts used in component UI. This console is available at http://<hostname>:<port-number>/libs/cq/i18n/translator.html; for example: <http://localhost:4502/libs/cq/i18n/translator.html>

### PERSONALIZATION

**What is Personalization?**

Personalization centers on providing the user with a tailor-made environment displaying dynamic content that is selected according to their specific needs; be this on the basis of predefined profiles, user selection, or interactive user behavior.

#### Campaign Management

Campaign management provides digital marketers the opportunity to deliver personalized content and so create dedicated experiences for visitors.

**Elements of Campaign Management**

* **Brands** 🡪 In CQ, brands are the top level unit and form a collection of Campaigns.
* **Campaigns** 🡪 A campaign is a collection of individual Experiences.
* **Experiences** 🡪The focused content forms the various experiences, presented to the visitor at Touchpoints.

**There are several types of experiences available:**

1. **Teasers 🡪**Teaser Pages / Paragraphs are used to steer specific visitor Segments to content that is focused on their interests.

Teaser pages can:

* present a range of options for the visitor to choose from
* Show only one teaser paragraph that is based on the specific visitor segment; for example, the teaser paragraph shown may be dependent on the age of the visitor. Typically a teaser page is a temporary action that will last for a specific period of time, until it is replaced by the next teaser page.

1. **Newsletters 🡪**E-mail Communications are used to engage users and encourage them to visit your web site. These usually take the form of a newsletter, sent to your **Leads** (**which are usually grouped into Lists**).
2. **Test &Target 🡪**This allows integration with Adobe's Test&Target which gives marketers a conversion website optimization tool with the necessary capabilities to continually make their online content and offers more relevant to their customers—yielding greater conversion. Test&Target provides an intuitive interface for designing and executing tests, creating audience segments and targeting content—all from a single application.

* **Touchpoints 🡪**These are the points of contact between the visitor and your campaign. The touchpoints are connected to the experiences that you have created. For example, for teasers it is the content page where the teaser paragraph is located, for a newsletter it is the mailing list.
* **Leads 🡪**The information that you have collected about your visitors and how to contact them forms the basis for your leads. For example forms can be Newsletter or Teasers
* **Lists 🡪**Leads are usually grouped into lists so that you can take collective action on them.

* **Segments🡪**Site visitors have different interests and objectives when they come to a site. Analyzing this according to factors such as activity on the website, profile information registered and activity on other websites, helps you to define segments. Content can then be specifically targeted to the visitor's needs and interests according to the segment(s) they match.
* **MCM🡪**The **Marketing Campaign Manager** (MCM) is a console that allows you to access all the functionality you need to create and control your campaigns, brands, experiences, touchpoints, leads, lists, segments and reports. It can be accessed from various locations (labelled as Campaigns), or with, for example, the URL: <http://localhost:4502/libs/mcm/content/admin.html>

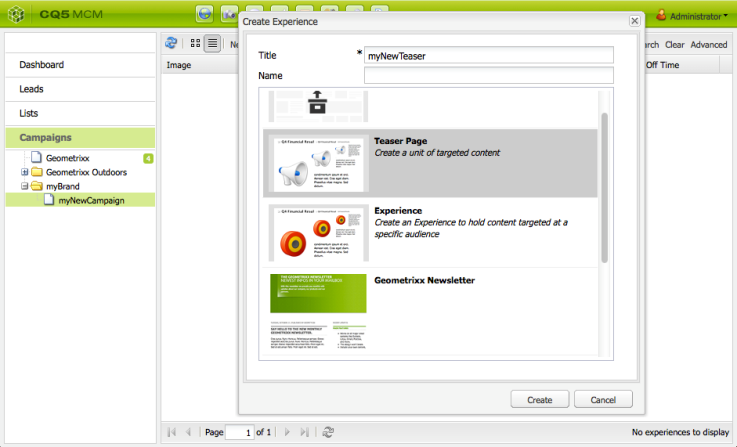
#### Teasers and Strategies

Campaigns often use teasers as a mechanism to entice a specific segment of the visitor population through to content focused on their interests.

|  |  |
| --- | --- |
|  | Brand pages are stored within the Campaigns section of the website. A brand contains the individual campaigns. |
| Campaign pages are stored within the Campaigns section of the website. Each campaign has an individual page, under which the teaser definitions are held. The container, or overview, page also holds certain information and statistics concerning the individual teaser pages.  🡪Teaser pages are stored under the appropriate campaign page and hold the definitions of the teaser paragraphs available for each specific campaign. **These definitions are used when displaying the teaser paragraphs; including content variations, the segment to be used for selecting a variation and boost factor.**  🡪The Teaser component is available out of the box and allows you to create an instance of your specific teaser paragraph in a content page. You can drag the teaser component from the sidekick and then specify your teaser definition to create your own teaser paragraph.  🡪Teaser paragraphs are actual instances of your teaser within a content page. These entice a segment of visitors through to content focused on their interests.  🡪Pages that hold the campaign content focused on a specific visitor segment. Usually the teaser paragraphs will lead the visitor to such pages. |

#### Creating a teaser experience

1. Select your new campaign in the left pane, or double-click it in the right pane.
2. Select the list view, using the icon: C:\Users\asi104\Desktop\New folder\1331640789000.png
3. Click New. Specify the Title, Name and type of experience to be created; in this case, Teaser Page. Click **Create**.



#### Adding content to your teaser

|  |  |
| --- | --- |
| Step 1: Select your campaign, then from the list view, select your teaser experience and click Edit. The teaser page will open. Use the sidekick to add components, then add your content to the teaser page:  C:\Users\asi104\Desktop\New folder\1362656039725.png | **Step 2: Assocociate the segment with the teaser for Teaser’s page properties** |

#### Creating a touchpoint for your teaser

After creating your teaser you need to create a touchpoint (where the visitor will see and access the teaser) to lead to your campaign.

**To Place the teaser paragraph within your content:**

|  |  |
| --- | --- |
| **Step 1:** Navigate to the content page where you want to place the teaser paragraph that will lead to your campaign page.  **Step 2:**  Add a Teaser component (available in the Personalization section of sidekick) in the required position. When first created it will show that the campaign path is not yet configured: | C:\Users\asi104\Desktop\New folder\1362659237900.png |
| Edit the teaser component to add the:  **Campaign Path -** Path to the campaign page that holds the individual teaser page; segments determine exactly which teaser is shown  **Strategy -** Method used for selection when multiple segments resolve successfully.(Refer below for details) | C:\Users\asi104\Desktop\New folder\1362660266494.png |

🡪Click OK to save. Depending on the segments you have set on the teaser and the profile of the user you are currently logged in as, the appropriate content will be displayed:

🡪Mouse over the teaser paragraph to reveal the question mark icon (bottom right corner of the component). Click this to view the segments applied and whether they currently resolve.

|  |  |
| --- | --- |
| C:\Users\asi104\Desktop\New folder\1362660407907.png | C:\Users\asi104\Desktop\New folder\1362660543969.png |

#### Strategies

When adding a teaser paragraph to a page you need to define the Strategy. **This is for the case that several teasers are available for selection as their assigned segments all resolve successfully.** **The Strategy then specifies extra criteria used to select the teaser** shown:

|  |  |
| --- | --- |
| Strategy | Comments |
| First | First in the list of resolved segments. The order is that of the teasers within the campaign container page. |
| Random | Uses the random factor generated for a page, this can be seen with the client context. |
| Clickstream Score | Based on the tags and related tag hits held within the visitor's client context (show how often a visitor has clicked on pages containing the respective tag). The hit rates for tags defined on the teaser page are compared. |

🡪 The Boost Factor of the segment also has an impact on the selection. This is a weighting factor added to a segment definition to increase/decrease the relative likelihood of it being selected.

For example, If the following segments have already been created and assigned their respective Boost Factor:

|  |  |
| --- | --- |
| Segment | Boost Factor |
| S1 | 0 |
| S2 | 0 |
| S3 | 10 |
| S4 | 30 |
| S5 | 0 |
| S6 | 100 |

And we use the following teaser definitions:

|  |  |  |  |
| --- | --- | --- | --- |
| Campaign | Teaser | Assigned Segment(s) | Assigned Tags |
| C1 | T1 | S1, S2 | Business, Marketing |
| C1 | T2 | S1 |  |
| C1 | T3 | S3, S4 |  |
| C1 | T4 | S2, S5 |  |
| C1 | T5 | S1, S2, S6 | Marketing |
| C1 | T6 | S6 | Business |

**Then if we apply this to a visitor where:**

* S1, S2 and S6 resolve successfully,
* the tag Marketing has 3 hits and
* the tag Business has 6 hits

**We can see the resulting:**

1. Match success - do any of the segments assigned to the teaser resolve successfully for the current visitor?
2. Boost factor - **The highest boost factor of all applicable segments**
3. Clickstream score - **The accumulative total for all applicable tag hits**

That is calculated before applying the appropriate strategy:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Campaign | Teaser | Assigned Segment(s) | Tags | Successful Match? | Resulting Boost Factor | Resulting Clickstream Score |
| C1 | T1 | S1, S2 | Business, Marketing | Yes | 0 | 9 |
| C1 | T2 | S1 |  | Yes | 0 |  |
| C1 | T3 | S3, S4 |  | No |  |  |
| C1 | T4 | S2, S5 |  | Yes | 0 |  |
| C1 | T5 | S1, S2, S6 | Marketing | Yes | 100 | 3 |
| C1 | T6 | S6 | Business | Yes | 100 | 6 |

These values are used to determine the teaser(s) that the visitor will see, depending on the Strategy applied to the teaser paragraph:

|  |  |  |
| --- | --- | --- |
| Strategy | Resulting Teaser | Comments |
| First | T5 | Only T5 and T6 are considered as their segments all resolve *and* they have the highest boost factor. The list returned is in the order T5, T6; so T5 is selected and shown. |
| Random | T5 or T6 | Both teasers have segments that all resolve and the same boost factor. Therefore, the two teasers are shown in equal proportion. |
| Clickstream Score | T6 | Segments for T1, T4, T5 and T6 all resolve for the visitor. The higher boost factors of T5 and T6 then excludes T1 and T4. Finally the higher Clickstream Score of T6 result in this being selected. |

#### Understanding Segmentation

|  |  |
| --- | --- |
| Segmentation terminology | |
| **Visitor** | A visitor is a person visiting a website. That person's visit typically starts from a referring page, then moves on to one or more page views on your own website. A behavioural profile can be created from the details of that person's visit. |
| **User** | A user is a visitor who registers with the website to receive an account profile. To generate their profile they provide additional identification, such as an email address and gender, amongst others. Additional information can also be collected, including community activity and purchase patterns, again amongst others. Based on the information provided in the profile, a demographic profile can be created. |
| **Trait** | A trait is a characteristic or property of a visitor that can be used to determine membership in a specific segment. |
| **Segment** | A segment is a collection of visitors that share certain traits. Segments should be distinctive, with a minimum of overlap with other segments. |
| **Behavioral Traits** | Behavioral traits are those that relate to a visitor's behaviour on the website. These include: 1. Interest within your website; including pages visited and products bought. 2.Interest on the referring website; including search terms used, or adverts clicked on. 3. Interest on other sites; determined using tools such as Spyjax. 4. Visitor loyality; duration of the visit, frequency of visits. |
| **Subsegment** | A segment can be subdivided into several subsegments. This is done by defining additional traits. |
| **Teaser Page** | A teaser page is directed at a specific audience. It contains re-usable content that can be used in the teaser paragraph. |
| **Campaign** | A campaign is a collection of teaser pages and e-mail marketing pages, such as newsletters or invitations. Typically a campaign runs for a limited period and is superceded by another campaign. |
| **Teaser Paragraph** | This is a paragraph that pulls content from another page dependent on a selection strategy. This selection strategy can take segments and campaigns into consideration. |
| **List** | A list is extracted from a segment of registered users. For example, the location used to steer the contents of the teaser paragraph. |
| **Demographic Traits** | These are selected population characteristics including:Age ;Income ;Family size ;Marital status ;Gender ;Location; Derived Traits |

#### Creating a new segment

|  |  |
| --- | --- |
| 1. Open the Tools console. 2. Click on the Segmentation page in the left pane, and navigate to the required location. 3. Create a new page using the Segment template. 4. Open the new page to see the segment editor: | C:\Users\asi104\Desktop\New folder\1405083267610.jpg |
| 1. Use either the sidekick or the context menu (usually right mouse button click, then select New... to open the Insert New Component window) to find the segment trait you need. Then drag it to the Segment Editor it will appear in the default AND container. 2. Double-click on the new trait to edit the specific parameters; for example the mouse position: Click OK to save your definition | C:\Users\asi104\Desktop\New folder\1328174662000.png |
| **Using and and or containers**  The top level of the definition is always the AND container that is initially created; this cannot be changed, but does not have an effect on the rest of your segment definition. Ensure that the nesting of your container makes sense. The containers can be viewed as the brackets of your boolean expression.The following example is used to select visitors who are either:  **Male and between the ages of 16 and 65 OR Female and between the ages of 16 and 62**  As the main operator is OR you need to start with an OR Container. Within this you have 2 AND statements, for each of these you need an AND Container, into which you can add the individual traits. | C:\Users\asi104\Desktop\New folder\1405083675212.jpg |

#### Creating Campaign Revisted

**Step 1 : Create a Segment**

|  |  |  |
| --- | --- | --- |
|  |  |  |

**Step 2: Create Brand🡪Campaign🡪 Teaser**

|  |  |  |
| --- | --- | --- |
| 1. **Creating Brand** | 1. **Creating Campaign** | 1. **Creating Teaser** |

|  |  |
| --- | --- |
| **Step 3: Adding Content to Teaser** |  |
| **Step 4: Associate the Treaser with the Segment from its page properties.** |  |

**Step4 : Create Touchpoint on the page where the teaser can be viewed**

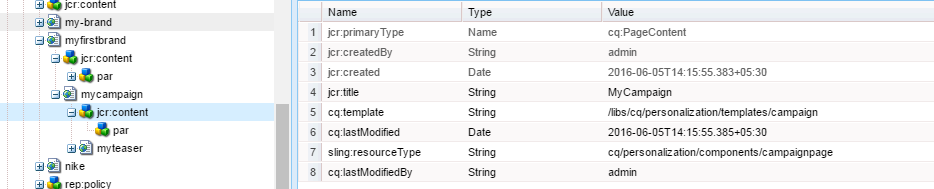
|  |  |
| --- | --- |
|  |  |

**Step 5**: Testing : Press Cntl+Alt+C 🡪 to open the client Context. Note to view the client content we must include the client content component in our jsp 🡪 **<cq:include path="clientcontext" resourceType="cq/personalization/components/clientcontext"/>**

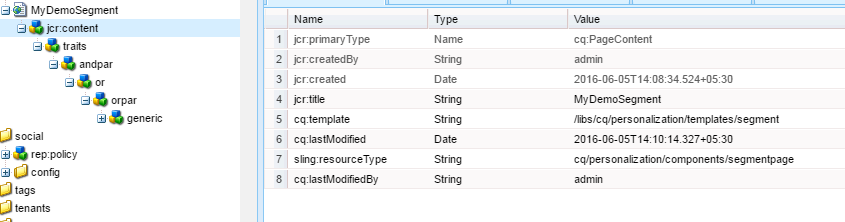
|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
| As per the rule in the Segment , the teaser wil be visible when the user is MALE. If we change the user to female the Teaser will disappear |  |

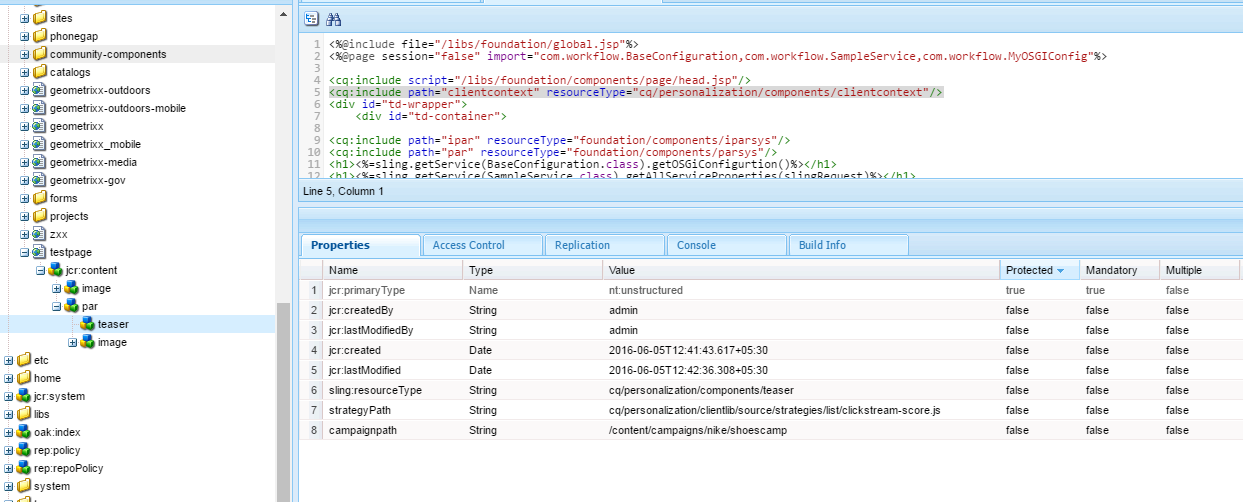
* The campaign are stored at **/content/campaigns** in CRX



* Segments are stored at **/etc/segmentation /etc/segmentation** in CRX



* **Touchpoints are created on the page(Teaser as an Experience)**



#### ContextHub

1. Introduced in AEM 6.2 (ClientContext has been deprecated , ContextHub has been recommended)
2. ***ContextHub is basically a framework for storing, manipulating, and presenting context data*** (data in which audience/end user is interested). With the help of client-side JavaScript API you can access data for personalizing content.
3. ContextHub data gets stored in Local Storage of the browser as **ContextHubPersistance** object.

|  |  |
| --- | --- |
|  | Personalization in AEM comes with 3 options   1. **Activities** - Activities is collecting a same kind of audience (segments) in a brand. 2. **Offers -** 3. **Audiences** - Audiences has conditions on the basis of which personalized the content will show off (same as segmentation) |

**WHERE CONTEXTHUB DATA IS STORED?**

AEM stores the user information in the ContextHubPersistance Object in the local storage.

Use Case: We want to personalize content on the basis of browsers like Google Chrome, Mozilla etc. So to detect which browser it is, we have OOTB store called "**surferinfo**". If you see the browser Family of Google Chrome, it is "Chrome" and for Mozilla Firefox it is Firefox. So let’s create the audiences basis of these two conditions.

**Audiences –** These are conditions on the basis of which personalized the content

**CREATING AUDIENCES**

|  |  |  |  |
| --- | --- | --- | --- |
| Go to Tools 🡪 Configuration Browser 🡪Create | Go to Personalization Console 🡪 Audiences🡪 Go to the Configuration created in Step 1 🡪 Create Context Hub segment    Similarly create Context Hub segment for **Firefox** | | Open the context hub segment tp set the rules for each Segment created.   * Drag and drop Comparision Property Component * Configure the values as shown below. * The background will be highlighted in Green, if the segment is resolved, as shown below. |
| C:\Users\asinha37\Desktop\Untitled.png | |  | |

**CREATING ACTIVITY**

Activities are collecting a same kind of audience (segments) in a brand.

|  |  |
| --- | --- |
| Go to Personalization console 🡪 Activities 🡪Create 🡪Create Brand 🡪Next |  |
| * Enter the brand title [**Browser Brand**] 🡪Create * Click & Open the new brand to configure the activities as shown below. * Click Next 🡪Save |  |
|  | |

**ENABLING PERSONALIZATION**

|  |  |
| --- | --- |
| 1. To enable personalization on a website, we need to add this line in page component. 2. **<sly data-sly-resource = "${'contexthub' @ resourceType='granite/contexthub/components/contexthub'}"/>** 3. To enable particular segments on a page, we need to go to page properties and add segments path.You can also enable a particular brand from the Brand Configuration. |  |

**ADDING PERSONALIZED CONTENT ON PAGE**

|  |  |  |
| --- | --- | --- |
| To add personalized content   * First add a default content on the page * Switch to Targeting Mode * Select the BRAND and ACTIVITY from drop down as shown below * Click on “Start Targeting” | |  |
|  | | |
|  | **SETTING UP DEFAULT CONTENT**   * Select DEFAULT from the Right Rail * Select the content to be personalized and click on the TARGET icon as shown below * The defauly content will appear when no Segmentation Rule is satisfied. | |
| **SETTING UP CROME BROWSER CONTENT** | * Select Chrome Users from Right rail to set-up chrome specific content * Select the content to change and update it. | |
| **SETTING UP FIREFOX BROWSER CONTENT** | * Select Firefox Users from Right rail to set-up chrome specific content * Select the content to change and update it.   **Click NEXT and SAVE** | |

**VALIDATION**

* Switch to Edit Mode
* Open the same page in different browser to see the personalized content (specific to each browser)

|  |  |  |
| --- | --- | --- |
| **CHROME BROWSER** | **FIREFOX BROWSER** | **IE BROWSER (DEFAULT CONTENT)** |

**ISSUE WITH CONTEXT-HUB MANAGED PERSONALIZED CONTENT**

Personalization using contexthub is that there is a lag in loading personalized content on a page.It first load default content on a page and then after some time it loads personalized content. This happens because when the page loads it runs contexthub library, resolves the segments and then show personalized content.

**OFFERS**

* In above steps to personalize the content we have updated the same component with having different content i.e same component having variation in the content.
* **But - using offer we can have different set of components which will be driven based on Context Hub Segmentation rules.**

### SOME USEFUL CODE

|  |
| --- |
| How to create a node using JCR API? <%@page import="javax.jcr.\*,javax.jcr.query.\*,java.util.\*,com.day.cq.commons.jcr.JcrUtil"%>  <%@page contentType="text/html; charset=utf-8"%>  <%@include file="/libs/foundation/global.jsp"%>  <%  String q = "/jcr:root/content/campaigns/geometrixx//\*" +"[@sling:resourceType='foundation/components/parsys']";  Query query = currentNode.getSession().getWorkspace().getQueryManager().createQuery(q, "xpath");  NodeIterator result = query.execute().getNodes();  int counter = 0;  while (result.hasNext()) {  Node n = result.nextNode();  Node newTextNode = JcrUtil.createUniqueNode(n, "newtext", "nt:unstructured", currentNode.getSession());  newTextNode.setProperty("sling:resourceType", "foundation/components/text");  newTextNode.setProperty("text", "<p>even more text</p>");  newTextNode.setProperty("textIsRich", "true");  counter++;  }  currentNode.getSession().save();  out.println("Added nodes: " + counter);  %> |
| Creating a page and Activation using PageManager API <%@page import="com.day.cq.tagging.\*, com.day.cq.wcm.api.\*, com.day.cq.replication.\*" %>  <%@include file="/libs/foundation/global.jsp" %>  <%  Page p=pageManager.create("/content/geometrixx/en", "mypage","/apps/geometrixx/templates/contentpage", "Hey a new page!");  Replicator r=sling.getService(Replicator.class);  r.replicate(currentNode.getSession(),  ReplicationActionType.ACTIVATE, p.getPath());  %> |
| Creating a image node in repository This code will fetch an image from an URL [from Restful Website ]and create a image component of that image  <%@page import="com.day.cq.tagging.\*,com.day.cq.dam.api.\*,com.day.cq.replication.\*,java.net.URL" %>  <%@page contentType="text/html; charset=utf-8" %>  <%@include file="/libs/foundation/global.jsp" %>  URL fileURL = new URL("http://dev.day.com/content/dam/portal/banner-deepdive.png");  AssetManager am = resource.getResourceResolver().adaptTo(AssetManager.class);  Asset myAsset=am.createAsset("/content/dam/geometrixx/myImage.png",fileURL.openStream(),"image.png",true);  TagManager tm=resource.getResourceResolver().adaptTo(TagManager.class);  String pathMetadata = myAsset.getPath() + "/jcr:content/metadata";  Resource resourceImage=resource.getResourceResolver().resolve(pathMetadata);  tm.setTags(resourceImage,new Tag[]{tm.resolve("/etc/tags/stockphotography/technology/")},true);  %> |

#### How to prepropulate the dialog from external source [JSON/JAVA CLASS]?

|  |  |
| --- | --- |
|  |  |
|  |  |
| Steps:   * Create a component and set the following properties(fig. 2) * Create a jsp [optionValue.json.jsp] and set the options =$PATH.optionValue.json * Fig 3:Shows the JSON Content or from external source[Java class]   Java Class:  <%@include file="/apps/richemont-jlc/core/global.jsp"%>  <%@page import="com.richemont.cms.jlc.utils.JLCTemplateDataUtils"%>  <% pageContext.setAttribute("templateDataMap", JLCTemplateDataUtils.getTemplatesDataMap(slingRequest)); %>  {"templateList":[  <c:forEach var="templateData" items="${templateDataMap}" varStatus="loop">  { "name": "${templateData.key}", "value": "${templateData.value}" } ${not loop.last ? ',' : ''}  </c:forEach>  ]} | |

### STEPS SETTING UP MONGO DB WITH AEM 6.0

**Download a msi or zip file form Mongo DB installation and a new directory with AEM Jar and licence.properties file**

|  |  |
| --- | --- |
| Step 1: Create a **data** and **log** folder in Mongo DB home directory  Step 2: Create mongod.cfg file in the home Mongo DB home directory  **mongod.cfg**  port = 27017  quiet = false  dbpath = C:\Program Files\MongoDB\Server\3.2\data  logpath = C:\Program Files\MongoDB\Server\3.2\log\mongod.log  logappend = true  journal = true  directoryperdb = true  rest=true  httpinterface=true  profile=2 |  |

**Step 3**: Start your mongo db server from bin directory by executing the below command

C:\Program Files\MongoDB\Server\3.2\bin>**mongod.exe --config "C:\Program Files\MongoDB\Server\3.2\mongod.cfg"**

if we see the log file in log folder it will show

[initandlisten] waiting for connections on port 27017 –

It means the mongo Db server is waiting for AEM server to start

**Step 4:** Start the AEM server but not by clicking the AEM jar file

* + Unpack the Jar file 🡪 **java -jar aem6.1-author-p4502.jar –unpack .** The unpacking will create the crx-quickstart folder
  + Go to the bin folder and open the start.bat file to change some startup configuration for Mongo Db server connection

Config required

1. Provide RunMode as **crx3mongo**

if not defined CQ\_RUNMODE set CQ\_RUNMODE=author,**crx3mongo**

1. Provide the path of MongoDB Server and increase the Xmx and PermSize

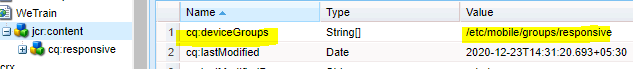
**::\* default JVM options**

1. **if not defined CQ\_JVM\_OPTS set CQ\_JVM\_OPTS=-Xmx4096m -XX:MaxPermSize=512M -Djava.awt.headless=true -Doak.mongo.uri=mongo://localhost:27017**
2. Start the AEM server **using start.bat file from bin directory**
3. The AEM will startup (<http://localhost:4502/crx/de/index.jsp> )

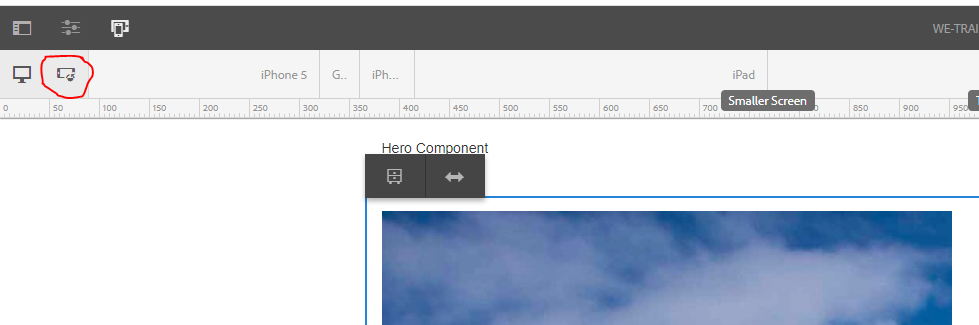
### ENABLING MOBILE EMULATOR

|  |  |
| --- | --- |
| **ENABLING MOBILE EMULATOR** | This allows you to create and edit responsive websites that rearrange the layout according to device/window size by resizing components interactively. The user can then see how the content will be rendered using the emulator. |
| **STEP 1 : CREATE MOBILE EMULATOR CONFIGURATION**   |  |  | | --- | --- | | **CONFIGURATION NAME** | **com.day.cq.wcm.mobile.core.impl.MobileEmulatorProvider**~**training**  FORMAT : **<*PID\_OF\_MOBILE\_EMULATOR*>~*<PROJECT\_NAME*>** | | **ADD PROPERTY TO CONFIG** |  | | |
| **EXAMPLE** | |

**STEP 2 : ADD DEVICE GROUP PROPERTY TO JCR CONTENT OF THE CONTENT PAGE**



#### STEP 3 : MOBILE EMULATOR IN ACTION



### DISPATCHER

* Dispatcher is Adobe Experience Manager's caching and/or load balancing tool. Using AEM's Dispatcher also helps to protect your AEM server from attack. Therefore, you can increase the security of your AEM instance by using the **Dispatcher in conjunction with an enterprise-class web server.**
* Dispather is a module which is deployed to a webserver like Apache. So the content is exposed to end user via webserver.

**STEPS TO CONFIGURE A DISPATCHER**

* + Install the Apache webserver
  + We need below files to configure the dispatcher
* **dispatcher.any**
  + The file has the configuration like caching mechanism, Publish instance which will eb exposed via dispatcher,the allowed headers/ file type etc.
  + The dispatcher.any file has to be copied in **conf** folder of apache
* **httpd.conf** – This file has configuration to load the dispatcher module as shown below

**LoadModule dispatcher\_module modules/disp\_apache2.2.dll**

* **disp\_apache2.2.dll** – This is the dispatcher module. This has to copied in the Apache webserver in - Apache Software Foundation\Apache2.2\modules**LoadModule dispatcher\_module modules/disp\_apache2.2.dll**

**HOW DISPATCHER WORKS?**

|  |  |
| --- | --- |
|  | **Here is a diagram demonstrating how the dispatcher typically fits in as part of your CQ architecture.**   1. Visitors request a file from your site via the dispatcher 2. If the dispatcher has not already cached the file this causes the dispatcher to connect to configured CQ instance to retrieve the file 3. CQ responds with the file 4. The dispatcher caches the file and serves it back to the visitor. 5. Sub sequent requests for the file would then be served from the cache. |

**WHY WE NEED A DISPATCHER?**

1. LOAD BALANCING
2. IMPROVE PERFORMANCE
3. FILTER OUT UNWANTED TRAFFIC FROM HITTING PUBLISH INSTANCES.

#### Dispatcher Configuration Files – dispatcher.any

The configuration file contains a series of single-valued or multi-valued properties that control the behavior of Dispatcher:

* Property names are prefixed with a forward slash ("/").
* Multi-valued properties enclose child items using braces ("{ }").

An example configuration is structured as follows:

|  |
| --- |
| # name of the dispatcher  **/name** "internet-server"  # each farm configures a set off (loadbalanced) renders  **/farms**  {  # first farm entry (label is not important, just for your convenience)  **/website**  {  **/clientheaders**  { # List of headers that are passed on }  **/virtualhosts**  { # List of URLs for this Web site }  **/sessionmanagement**  { # settings for user authentification }  **/renders**  { # List of AEM instances that render the documents }  **/filter**  { # List of filters }  **/vanity-urls**  { # List of vanity URLs }  **/cache**  {  # Cache configuration  **/rules**  { # List of cachable documents }  **/invalidate**  { # List of auto-invalidated documents }  }  **/statistics**  {  **/categories**  { # The document categories that are used for load balancing estimates }  }  **/stickyConnectionsFor** "/myFolder"  **/health\_check**  { # Page gets contacted when an instance returns a 500  }  **/retryDelay** "1"  **/numberOfRetries** "5"  **/unavailablePenalty** "1"  **/failover** "1"  }  } |

**FARMS**

* The /farms property is a top-level property in the configuration structure. For each site we have one farm.The /farms property defines one or more sets of Dispatcher behaviors, **where each set is associated with different web sites or URLs**.
* The /farms property can include a single farm or multiple farms- as shown below
  1. If the configuration file is large we can split it into several smaller files (that are easier to manage) then include these.
  2. Use a single farm when you want Dispatcher to handle all of your web pages or web sites in the same way.
  3. Create multiple farms when different areas of your web site or different web sites require different Dispatcher behavior.
* **Naming the Dispatcher Instance (refer above)- /name**
  1. Use the **/name** property to specify a unique name to identify your Dispatcher instance. The **/name** property is a top-level property in the configuration structure.

|  |  |
| --- | --- |
| For example, to include the file myFarm.any in the /farms configuration  ***/farms***  ***{***  ***$include "myFarm.any"***  ***}*** | Use the asterisk ("\*") as a wildcard to specify a range of files to include. For example, if the files **farm\_1.any** through to **farm\_5.any** contain the configuration of farms one to five, you can include them as follows:  ***/farms***  ***{***  ***$include "farm\_\*.any"***  ***}*** |
| The following example shows the skeleton definition for two farms named /daycom and /docsdaycom: | #name of dispatcher  /name "day sites"   #farms section defines a list of farms or sites  /farms  {     /daycom     {         ...     }     /docdaycom     {        ...     }  } |

**CHILDS OF FARM PROPERTY**

|  |  |
| --- | --- |
| **Property name** | **Description** |
| /homepage | Default homepage ((optional)(IIS only)) |
| [/clientheaders](https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html#SpecifyingtheHTTPHeaderstoPassThroughclientheaders) | The headers from the client HTTP request to pass through. |
| [/virtualhosts](https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html#IdentifyingVirtualHostsvirtualhosts) | The virtual hosts for this farm. |
| [/sessionmanagement](https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html#EnablingSecureSessionssessionmanagement) | Support for session management and authentication. |
| [/renders](https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html#DefiningPageRenderersrenders) | The servers that provide rendered pages (typically AEM publish instances). |
| [/filter](https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html#ConfiguringAccesstoContentfilter) | Defines the URLs to which Dispatcher enables access. |
| [/vanity-urls](https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html#EnablingAccesstoVanityURLsvanityurls) | Configures access to vanity URLs. |
| [/propagateSyndPost](https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html#ForwardingSyndicationRequestspropagateSyndPost) | Support for the forwarding of syndication requests. |
| [/cache](https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html#ConfiguringtheDispatcherCachecache) | Configures caching behavior. |
| [/statistics](https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html#ConfiguringLoadBalancingstatistics) | Defining statistic categories for load-balancing calculations. |
| [/stickyConnectionsFor](https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html#IdentifyingaStickyConnectionFolderstickyConnectionsFor) | The folder that contains sticky documents. |
| [/health\_check](https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html#SpecifyingaHealthCheckPage) | The URL to use to determine server availability. |
| [/retryDelay](https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html#SpecifyingthePageRetryDelay) | The delay before retrying a failed connection. |
| [/unavailablePenalty](https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html#ReflectingServerUnavailabilityinDispatcherStatistics) | Penalties that affect statistics for load-balancing calculations. |
| [/failover](https://helpx.adobe.com/experience-manager/dispatcher/using/dispatcher-configuration.html#UsingtheFailoverMechanism) | Resend requests to different renders when the original request fails. |

|  |  |  |
| --- | --- | --- |
| **/homepage**   * The optional /homepage parameter specifies the page that dispatcher returns when a client requests an undeterminable page or file. * Typically this situation occurs when a user specifies an URL for which neither IIS nor AEM provides an automatic redirection target | **#name of dispatcher**  **/name "day sites"**  **#farms section defines a list of farms or sites**  **/farms**  **{**  **/daycom**  **{**  **/homepage "/index.html"**  **...**  **}**  **/docdaycom**  **{**  **...**  **}**  **}** | |
| **/clientheaders**   * The /clientheaders property defines a list of HTTP headers that Dispatcher passes from the client HTTP request to the renderer (AEM instance). * By default Dispatcher forwards the standard HTTP headers to the AEM instance. | **/clientheaders**  **{**  **"user-agent"**  **"content-type"**  **"content-length"**  **"accept"**  **"host"**  **….**  **}** | |
| **/virtualhosts**   * The **/virtualhosts** property **defines a list of all hostname/URI combinations that Dispatcher accepts for this farm**. You can use the asterisk ("\*") character as a wildcard. * Values for the /**virtualhosts** property use the following format: ***[scheme]host[uri][\*] ex https://*www.mycompany.com/products"** * **scheme**: (Optional) Either **http://** or **https://.** * **host**: The name or IP address of the host computer and the port number if necessary [**www.mycompany.com**] * **uri**: (Optional) The path to the resources. [***products***] | **EXAMPLE 1**  The following example configuration handles requests for the .com and .ch domains of myCompany, and all domains of mySubDivision  /virtualhosts   {   "www.myCompany.com"   "www.myCompany.ch"   "www.mySubDivison.\*"   }  **EXAMPLE 2 : The following configuration handles *all* requests:**  **/virtualhosts**  **{**  **"\*"**  **}**  **EXAMPLE 3**  /virtualhosts  {  "www.awe-\*-uhcmedicaresolutions.uhc.com"  "www.awe-\*-aarpmedicareplans.uhc.com"  "www.awe-\*-uhcretiree.uhc.com"  "www.awe-\*-uhcmedicarerxforgroups.uhc.com"  "localhost"  } | |
| **RESOLVING THE VIRTUAL HOST**  When Dispatcher receives an HTTP or HTTPS request, it finds the virtual host value that best-matches the **host,** **uri**, and **scheme** headers of the request. Dispatcher evaluates the values in the **virtualhosts** properties in the following order:   1. Dispatcher begins at the lowest farm and progresses upward in the dispatcher.any file.(order of $include files) 2. For each farm, Dispatcher begins with the topmost value in the **virtualhosts** property and progresses down the list of values. 3. Dispatcher finds the best-matching virtual host value in the following manner:    1. The first-encountered virtual host that matches all three of the **host**, the **scheme**, and the **uri** of the request is used.    2. If no **virtualhosts** values has **scheme** and **uri** parts that both match the **scheme** and **uri** of the request, the first-encountered virtual host that matches the **host** of the request is used.    3. If no **virtualhosts** values have a **host** part that matches the host of the request, the topmost virtual host of the topmost farm is used. Therefore, you should place your default virtual host at the top of the **virtualhosts** property in the topmost farm of dispatcher.any file.   **Example Virtual Host Resolution**  Using this example, the following table shows the virtual hosts that are resolved for the given HTTP requests:   |  |  | | --- | --- | | **Request URL** | **Resolved virtual host** | | http://www.mycompany.com/products/gloves.html | www.mycompany.com/products/\* | | <http://www.mycompany.com/about.html> | www.mycompany.com | | | EXAMPLE  /farms    {    /myProducts      {  **/virtualhosts**  **{**  **"www.mycompany.com"**  **}**      /renders        {        /hostname "server1.myCompany.com"        /port "80"        }      }    /myCompany      {  **/virtualhosts**  **{**  **"www.mycompany.com/products/\*"**  **}**      /renders        {        /hostname "server2.myCompany.com"        /port "80"        }      }  } |

**/renders** - Defining Page Renderers - The /renders property defines the URL to which Dispatcher sends requests to render a document

|  |  |  |
| --- | --- | --- |
| The following example **/renders** section identifies a single AEM instance for rendering:  **/renders**  **{**  **/myRenderer**  **{**  **/hostname "aem.myCompany.com"**  **/port "4503"**  **# connection timeout in milliseconds, "0" (default) waits indefinitely**  **/timeout "0"**  **}**  **}** | The following example /renders section identifies an AEM instance that runs on the same computer as Dispatcher:  **/renders**  **{**  **/myRenderer**  **{**  **/hostname "127.0.0.1"**  **/port "4503"**  **}**  **}** | The following example /renders section distributes render requests equally among two AEM instances:  **/renders**  **{**  **/myFirstRenderer**  **{**  **/hostname "aem.myCompany.com"**  **/port "4503"**  **}**  **/mySecondRenderer**  **{**  **/hostname "127.0.0.1"**  **/port "4503"**  **}**  **}** |

**DEFINING A FILTER**

Each item in the **/filter** section includes a type and a pattern that is matched with a specific element of the request line or the entire request line. Each filter can contain the following items:

* **Type:**The **/type** indicates whether to allow or deny access for the requests that match the pattern. The value can be either **allow** or **deny**.
* **Element of the Request Line:** Include **/method**, **/url**, **/query**, or **/protocol** and a pattern for filtering requests according to these specific parts of the request-line part of the HTTP request. Filtering on elements of the request line (rather than on the entire request line) is the preferred filter method.
* **glob Property:**The **/glob** property is used to match with the entire request-line of the HTTP request.

|  |  |
| --- | --- |
| * You should deny access to everything, then allow access to specific (limited) elements: | **/filter**  **{**  **# Deny everything first and then allow specific entries**  **/0001 { /type "deny" /glob "\*" }**  **# Allow non-public content directories**  **/0023 { /type "allow" /url "/content\*" }**  **}** |

**FILTER EXAMPLES**

|  |  |  |
| --- | --- | --- |
| **DENY ALL**  Deny requests for all files.  **/0001  {/type "deny" /glob "\*" }** | **DENY ACCESS TO SPECIFIC AREAS**  The following filter denies access to ASP pages:  **/0002  { /type "deny" /url "\*.asp"  }** | **ENABLE POST REQUESTS**  The following example filter allows submitting form data by the POST method  **/filter {**  **/0001  { /glob "\*" /type "deny" }**  **/0002 { /type "allow" /method "POST" /url "/content/[.]\*.form.html" }**  **}** |
| ALLOW ACCESS TO THE WORKFLOW CONSOLE  **/filter {**  **/0001  { /glob "\*" /type "deny" }**  **/0002  {  /type "allow"  /url "/libs/cq/workflow/content/console\*"  }**  **}** | **USING REGULAR EXPRESSIONS**  This filter enables extensions in non-public content directories using a regular expression, defined here between single quotes:  **/005  {  /type "allow" /extension '(css|gif|ico|js|png|swf|jpe?g)' }** | |

**/CACHE - CONFIGURING THE DISPATCHER CACHE**

The **/cache** section controls how dispatcher caches documents. :

|  |  |
| --- | --- |
| Configure several sub-properties to implement your caching strategies   * /docroot * /statfile * /serveStaleOnError * /allowAuthorized * /rules * /statfileslevel * /invalidate * /invalidateHandler * /allowedClients * /ignoreUrlParams * /headers * /mode | An example cache section might look as follows  **/cache**  **{**  **/docroot "/opt/dispatcher/cache"**  **/statfile  "/tmp/dispatcher-website.stat"**  **/allowAuthorized "0"**  **/rules**  **{**  **# List of files that are cached**  **}**  **/invalidate**  **{**  **# List of files that are auto-invalidated**  **}**  **}** |

|  |  |  |
| --- | --- | --- |
| /docroot | identifies the directory where cached files are stored | |
| **/statfile** | This property identifies the file to use as the statfile. Dispatcher uses this file to register the time of the most recent content update. The statfile can be any file on the web server.The statfile has no content. When content is updated, Dispatcher updates the timestamp. The default statfile is named .stat and is stored in the docroot. Dispatcher blocks access to the statfile.  **Note:**  If**/statfileslevel** is configured, Dispatcher ignores the **/statfile** property and uses .stat as the name. | |
| **/serveStaleOnError** | **Serving Stale Documents When Errors Occur**  This property controls whether Dispatcher returns invalidated documents when the render server returns an error. By default, when a statfile is touched and invalidates cached content, Dispatcher deletes the cached content the next time it is requested.  If **/serveStaleOnError** is set to "1", Dispatcher does not delete invalidated content from the cache unless the render server returns a successful response. A 5xx response from AEM or a connection timeout causes Dispatcher to serve the outdated content and respond with and HTTP Status of 111 (Revalidation Failed). | |
| /allowAuthorized | The **/allowAuthorized** property controls whether requests that contain any of the following authentication information are cached:  The **authorization** header.  A cookie named **authorization**.  A cookie named **login-token**.  By default, requests that include this authentication information are not cached because authentication is not performed when a cached document is returned to the client. This configuration prevents Dispatcher from serving cached documents to users who do not have the necessary rights.  However, if your requirements permit the caching of authenticated documents, set /allowAuthorized to one:  **/allowAuthorized "1"** | |
| /rules | The /rules property controls which documents are cached according to the document path   * If the request URI contains a question mark ("?"). * This usually indicates a dynamic page, such as a search result that does not need to be cached. * The file extension is missing. * The web server needs the extension to determine the document type (the MIME-type). * The authentication header is set (this can be configured) * If the AEM instance responds with the following headers:   1. no-cache   2. no-store   3. must-revalidate * The GET or HEAD (for the HTTP header) methods are cacheable by the Dispatcher   **Each item in the /rules property includes a glob pattern and a type:**   * The glob pattern is used to match the path of the document. * The type indicates whether to cache the documents that matches the glob pattern. The value can be either allow (to cache the document) or deny (to always render the document).   Example:  If you do not have dynamic pages we can configure Dispatcher to cache everything.  /rules  {  /0000 { /glob "\*" /type "allow" }  }  If there are some sections of your page that are dynamic ,for example a news application  /rules  {  /0000 { /glob "\*" /type "allow" }  /0001 { /glob "/en/news/\*" /type "deny" }  } | |
| /statfileslevel |  | |
| /invaidate | * The /invalidate property defines the documents that are automatically invalidated when content is updated. * With automatic invalidation, Dispatcher does not delete cached files after a content update, but checks their validity when they are next requested. Documents in the cache that are not auto-invalidated will remain in the cache until a content update explicitly deletes them. * Automatic invalidation is typically used for HTML pages. HTML pages often contain links to other pages, making it difficult to determine whether a content update affects a page. To make sure that all relevent pages are invalidated when content is updated, automatically invalidate all HTML pages.   Example 1 : The following configuration invalidates all HTML pages:  /invalidate  {  /0000 { /glob "\*" /type "deny" }  /0001 { /glob "\*.html" /type "allow" }  }  This configuration causes the following activiy when the /content/geometrixx/en.html file is activated:   * All the files with pattern en.\* are removed from the /content/geometrixx/ folder. * The /content/geometrixx/en/jcr\_content folder is removed. * All the other files that match the /invalidate configuration are not immediately deleted. These files are deleted when the next request occurs. In our example /content/geometrixx.html is not deleted, it will be deleted when /content/geometrixx.html is requested. | |
| /invaidateHandler | The /invalidateHandler property allows you to define a script which is called for each invalidation request received by Dispatcher.  It is called with the following arguments:   1. Handle 2. The content path that is invalidated 3. Action | curl -H "CQ-Action:Delete" -H "CQ-Handle:/" -H "CQ-Path:/" <http://dispatcher-publish-team-t.ocp-elr-core-nonprod.optum.com/dispatcher/invalidate.cache> |
| /allowedClients | The /allowedClients property defines specific clients that are allowed to flush the cache. The globbing patterns are matched against the IP. The following example:denies access to any client and explicitly allows access to the localhost  /allowedClients  {  /0001 { /glob "\*.\*.\*.\*" /type "deny" }  /0002 { /glob "127.0.0.1" /type "allow" }  } | |
| /IgnoreURlParams | The ignoreUrlParams section defines which URL parameters are ignored when determining whether a page is cached or delivered from cache:   * When a request URL contains parameters that are all ignored, the page is cached. * When a request URL contains one or more parameters that are not ignored, the page is not cached. * When a parameter is ignored for a page, the page is cached the first time that the page is requested. Subsequent requests for the page are served the cached page, regardless of the value of the parameter in the request. * To specify which parameters are ignored, add glob rules to the ignoreUrlParams property: * To ignore a parameter, create a glob property that allows the parameter. * To prevent the page to be cached, create a glob property that denies the parameter.   The following example causes Dispatcher to ignores the "q" parameter, so that request URLs that include the q parameter are cached:  /ignoreUrlParams  {  /0001 { /glob "\*" /type "deny" }  /0002 { /glob "q" /type "allow" }  }  HTTP request causes the page to be cached because the q parameter is ignored: **GET /mypage.html?q=5**  HTTP request causes the page to not be cached because the p parameter is not ignored: **GET /mypage.html?q=5&p=4** | |
| /headers | The /headers property allows you to define the HTTP header types that are going to be cached by the Dispatcher. On the first request to an uncached resource, all headers matching one of the configured values (see the configuration sample below) are stored in a separate file, next to the cache file. On subsequent requests to the cached resource, the stored headers are added to the response.  Presented below is a sample from the default configuration:  /cache {  ...  /headers {  "Cache-Control"  "Content-Disposition"  "Content-Type"  "Expires"  "Last-Modified"  "X-Content-Type-Options"  "Last-Modified"  }  } | |
| /mode | The mode property specifies what file permissions are applied to new directories and files in the cache. This setting is restricted by the umask of the calling process. It is an octal number constructed from the sum of one or more of the following values:   1. 0400 Allow read by owner. 2. 0200 Allow write by owner. 3. 0100 Allow the owner to search in directories. 4. 0040 Allow read by group members. 5. 0020 Allow write by group members. 6. 0010 Allow group members to search in the directory. 7. 0004 Allow read by others. 8. 0002 Allow write by others. 9. 0001 Allow others to search in the directory.   The default value is 0755 which allows the owner to read, write or search and the group and others to read or search. | |
| /statistics | The /statistics section defines categories of files for which Dispatcher scores the responsiveness of each render. Dispatcher uses the scores to determine which render to send a request.  Each category that you create defines a glob pattern. Dispatcher compares the URI of the requested content to these patterns to determine the category of the requested content:  The order of the categories determines the order in which they are compared to the URI.  The first category pattern that matches the URI is the category of the file. No more category patterns are evaluated.  Dispatcher supports a maximum of 8 statistics categories. If you define more than 8 categories, only the first 8 are used.  Render Selection  Each time Dispatcher requires a rendered page, it uses the following algorithm to select the render:  If the request contains the render name in a renderid cookie, Dispatcher uses that render.  If the request includes no renderid cookie, Dispatcher compares the render statistics:  Dispatcher determines the cateogry of the request URI.  Dispatcher determines which render has the lowest response score for that category, and selects that render.  If no render is selected yet, use the first render in the list.  The score for a render's category is based on previous response times, as well as previous failed and successful connections that Dispatcher attempts. For each attempt, the score for the category of the requested URI is updated.  Note:  If you do not use load balancing, you can omit this section.  Defining Statistics Categories  Define a category for each type of document for which you want to keep statistics for render selection. The /statistics section contains a /categories section. To define a category, add a line below the /categories section that has the following format:  /name { /glob "pattern"}  The category name must be unique to the farm. The pattern is described in the Designing Patterns for glob Properties section.  To determine the category of a URI, Dispatcher compares the URI with each category pattern until a match is found. Dispatcher begins with the first category in the list and cointinues in order. Therefore, place categories with more specific patterns first.  For example, Dispatcher the default dispatcher.any file defines an HTML category and an others category. The HTML category is more specific and so it appears first:  /statistics  {  /categories  {  /html { /glob "\*.html" }  /others { /glob "\*" }  }  }  The following example also includes a category for search pages:  /statistics  {  /categories  {  /search { /glob "\*search.html" }  /html { /glob "\*.html" }  /others { /glob "\*" }  }  } | |

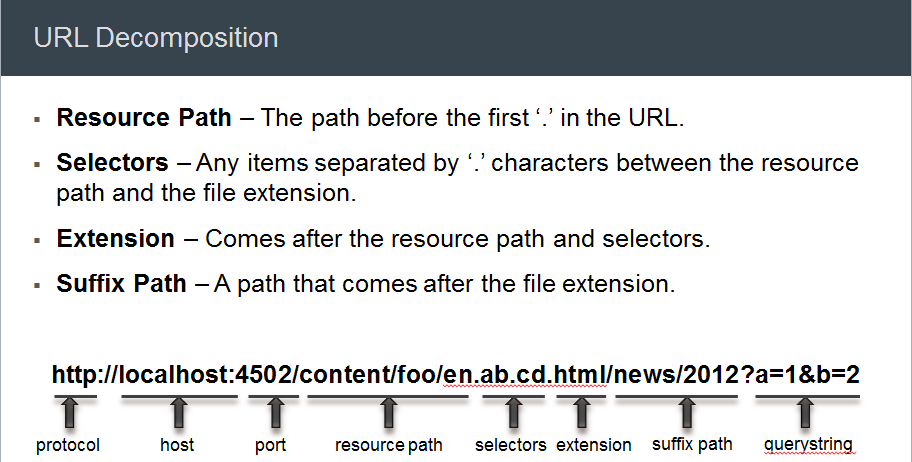
**DESIGNING PATTERNS FOR GLOB PROPERTIES**

Several sections in the Dispatcher configuration file use **glob** properties as selection criteria for client requests. The values of glob properties are patterns that Dispatcher compares to an aspect of the request, such as the path of the requested resource, or the IP address of the client. For example, the items in the **/filter** section use glob patterns to identify the paths of the pages that Dispatcher acts on or rejects.

The glob values can include wildcard characters and alphanumeric characters to define the pattern. The following table describes the wildcard characters.

|  |  |  |
| --- | --- | --- |
| Wildcard character | Description | Examples |
| \* | Matches zero or more contiguous instances of any character in the string. The final character of the match is determined by either of the following situations:  A character in the string matches the next character in the pattern, and the pattern character has the following characteristics:  Not a \*  Not a ?  A literal character (including a space) or a character class.  The end of the pattern is reached.  Inside a character class, the character is interpreted literally. | \*/geo\*  Matches any page below the /content/geometrixx node and the /content/geometrixx-outdoors node. The following HTTP requests match the glob pattern:  "GET /content/geometrixx/en.html"  "GET /content/geometrixx-outdoors/en.html"  \*outdoors/\*  Matches any page below the /content/geometrixx-outdoors node. For example, the following HTTP request matches the glob pattern:  "GET /content/geometrixx-outdoors/en.html" |
| ? | Matches any single character. Use outside character classes.  Inside a character class, this character is interpreted literally. | \*outdoors/??/\*  Matches the pages for any language in the geometrixx-outdoors site. For example, the following HTTP request matches the glob pattern:  "GET /content/geometrixx-outdoors/en/men.html"  The following request does not match the glob pattern:  "GET /content/geometrixx-outdoors/en.html" |
| [ and ] | Demarks the beginning and end of a character class.  Character classes can include one or more character ranges and single characters.  A match occurs if the target character matches any of the characters in the character class, or within a defined range.  If the closing bracket is not included, the pattern produces no matches. | \*[o]men.html\*  Matches the following HTTP request:  "GET /content/geometrixx-outdoors/en/women.html"  Does not match the following HTTP request:  "GET /content/geometrixx-outdoors/en/men.html"  \*[o/]men.html\*  Matches the following HTTP requests:  "GET /content/geometrixx-outdoors/en/women.html"   "GET /content/geometrixx-outdoors/en/men.html" |
| - | Denotes a range of characters. For use in character classes.  Outside of a character class, this character is interpreted literally. | \*[m-p]men.html\*  Matches the following HTTP request:  "GET /content/geometrixx-outdoors/en/women.html"  Does not match the following HTTP request:  "GET /content/geometrixx-outdoors/en/men.html" |
| ! | Negates the character or character class that follows. Use only for negating characters and character ranges inside character classes. Equivalent to the ^ wildcard.  Outside of a character class, this character is interpreted literally. | \*[!o]men.html\*  Matches the following HTTP request:  "GET /content/geometrixx-outdoors/en/men.html"  Does not match the following HTTP request  "GET /content/geometrixx-outdoors/en/women.html"  \*[!o!/]men.html\*  Does not match the following HTTP request:  "GET /content/geometrixx-outdoors/en/women.html" or "GET /content/geometrixx-outdoors/en/men. html" |
| ^ | Negates the character or character range that follows. Use for negating only characters and character ranges inside character classes. Equivalent to the ! wildcard character.  Outside of a character class, this charcter is interpreted literally. | The examples for the ! wildcard character apply, substituting the ! characters in the example patterns with ^ characters. |

#### URL Decomposition



**THE DISPATCHER WILL ONLY CACHE FILES THAT MEET THE FOLLOWING CRITERIA**

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

#### Methods for Caching

The Dispatcher has two primary methods for updating the cache content when changes are made to the website.

1. Content Updates remove the pages that have changed, as well as files that are directly associated with them.
2. Auto-Invalidation automatically invalidates those parts of the cache that may be out of date after an update. I.e. it effectively flags relevant pages as being out of date, without deleting anything.

**CONTENT UPDATES**

In a content update, one or more AEM documents change. AEM sends a syndication request to the Dispatcher, which updates the cache accordingly:

**AUTO-INVALIDATION**

* Auto-invalidation automatically invalidates parts of the cache - without physically deleting any files. At every content update, the so-called statfile is touched, so its timestamp reflects the last content update.
* The Dispatcher has a list of files that are subject to auto-invalidation. When a document from that list is requested, the Dispatcher compares the date of the cached document with the timestamp of the statfile:

1. If the cached document is newer, the Dispatcher returns it.
2. If it is older, the Dispatcher retrieves the current version from the AEM instance.

### CURL

**What is cURL?**

cURL is a command line tool for doing all sorts of URL manipulations and transfers. It is used for transferring data using various protocols, HTTP,FTP, Gopher, TELNET, etc. It comes in handy for automation, and if you know how script you can create powerful tools that will facilitate your work process. The name stands for Curl URL Request Library.

To view Curl List : **http://localhost:4502/crx/packmgr/service.jsp**

|  |  |
| --- | --- |
| **Arguments** | **Comment** |
| cmd=help | print this help |
| cmd=ls | print a list of all packages |
| cmd=rm name [group] | remove a package package name group name (optional) |
| cmd=build name [group] | build a package package name group name (optional) |
| cmd=install name [strict] [group] | install a package package name group name (optional) true or fail in error |
| cmd=uninstall name [group] | install a package package name group name (optional) |
| GET - download a package | |
| cmd=get name [group] | download a package package name group name (optional) |
| POST - upload a package | |
| cmd=get file [group] [strict] [install] | upload a package package name group name (optional) true or fail on install error automatically install package if 'true' |

|  |  |  |  |
| --- | --- | --- | --- |
| **Package Management Commands** | | | |
| **DESCRIPTION** | | | **COMMAND** |
| **Help Menu for useful AEM commands** | | | curl -u admin:admin http://localhost:4502/crx/packmgr/service.jsp?**cmd=help** |
| **List of all the packages in your AEM instance** | | | curl -u admin:admin http://localhost:4502/crx/packmgr/service.jsp?**cmd=ls** |
| **Build an existing package** | | | curl -u admin:admin -X POST http://localhost:4502/crx/packmgr/service/.json/etc/packages/my\_packages/samplepackage.zip?**cmd=build** |
| **Delete an existing package** | | | curl -u admin:admin -X POST http://localhost:4502/crx/packmgr/service/.json/etc/packages/my\_packages/samplepackage.zip?**cmd=delete** |
| **Install an existing package** | | | curl -u admin:admin -X POST http://localhost:4502/crx/packmgr/service/.json/etc/packages/my\_packages/samplepackage.zip?**cmd=install** |
| **Uninstall an existing package** | | | curl -u admin:admin -X POST http://localhost:4502/crx/packmgr/service/.json/etc/packages/my\_packages/samplepackage.zip?**cmd=uninstall** |
| **Download an existing package into filesystem** | | | curl -u admin:admin http://localhost:4502/etc/packages/my\_packages/samplepackage.zip > <local filepath> |
| **Upload and don’t install an existing package from File system** | | | curl -u admin:admin -F file=@"C:\sample\samplepackage.zip" -F name="samplepackage" -F force=true -F **install=false** http://localhost:4502/crx/packmgr/service.jsp |
| **Upload and Install an existing package from File system** | | | curl -u admin:admin -F file=@"C:\sample\samplepackage.zip" -F name="samplepackage" -F force=true -F **install=true** http://localhost:4502/crx/packmgr/service.jsp |
| **JCR Node Management Commands** | | | |
| **Delete a Node** | | | curl -X DELETE http://localhost:4502/content/geometrixx/en/products/jcr:content/par/flash -u admin:admin |
| **Create or Add a Node** | | | curl --data jcr:primaryType=nt:unstructured --user admin:admin http://localhost:4502/content/geometrixx/en/toolbar/test3 |
| **Create a Page** | | | curl -u admin:admin -F "jcr:primaryType=cq:Page" -F "jcr:content/jcr:primaryType=cq:PageContent" -F "jcr:content/jcr:title=Curl Page" -F "jcr:content/sling:resourceType=geometrixx/components/contentpage" http://localhost:4502/content/geometrixx/en/curlPage |
| **JCR Query API Commands** | | | |
| **Find an Asset from the JCR** | | | curl -s -u admin:admin GET "http://localhost:4502/bin/querybuilder.json?path=%2fcontent%2fgeometrixx%2fen&property=fileReference&property.value=%2fcontent%2fdam%2fgeometrixx%2fshapes%2ftri\_equilateral.png&type=nt%3aunstructured" |
| **Page Management Commands** | | | |
| **Lock a Page** | | | curl -u admin:admin -X POST -F cmd="lockPage" -F path="/content/geometrixx/en/toolbar/contacts" -F "\_charset\_"="utf-8" http://localhost:4502/bin/wcmcommand |
| **Unlock a Page** | | | curl -u admin:admin -X POST -F cmd="unlockPage" -F path="/content/geometrixx/en/toolbar/contacts" -F "\_charset\_"="utf-8" http://localhost:4502/bin/wcmcommand |
| **Copy/Move a Page** | | | curl -u admin:admin -F:operation=copy -F:dest=/content/geometrixx/en/products/contacts http://localhost:4502/content/geometrixx/en/toolbar/contacts |
| **AEM Replication Commands** | | | |
| **Activate** | curl -u admin:admin -X POST -F path="/content/geometrixx/en/toolbar/contacts" -F **cmd="activate"** http://localhost:4502/bin/replicate.json | | |
| **Deactivate** | curl -u admin:admin -X POST -F path="/content/geometrixx/en/toolbar/contacts" -F **cmd="deactivate"** http://localhost:4502/bin/replicate.json | | |
| **Tree Activation** | curl -u admin:admin -F cmd=activate -F ignoredeactivated=true -F onlymodified=true -F path=/content/geometrixx/en/community http://localhost:4502/etc/replication/treeactivation.html | | |
| **OSGi Bundle Management Commands** | | | |
| **Stop a Bundle** | | curl -u admin:admin http://localhost:4502/system/console/bundles/com.adobe.aemds.guide.aemds-guide-core -Faction=stop | |
| **Start a Bundle** | | curl -u admin:admin http://localhost:4502/system/console/bundles/com.adobe.aemds.guide.aemds-guide-core -Faction=start | |
| **Install a Bundle from File system** | | curl -u admin:admin -F action=install -F bundlestartlevel=20 -F bundlefile=@"<path of samplejar.jar>" http://localhost:4502/system/console/bundles | |
| **User Management Commands** | | | |
| **Create a new User** | | | curl -u admin:admin -FcreateUser= -FauthorizableId=hashim -Frep:password=hashim http://localhost:4502/libs/granite/security/post/authorizables |
| **Create a new Group[Creates a Group Name -testGroup1]** | | | curl -u admin:admin -FcreateGroup=group1 -FauthorizableId=**testGroup1** http://localhost:4502/libs/granite/security/post/authorizables |
| **Create a User with a profile** | | | curl -u admin:admin -FcreateUser=testuser -FauthorizableId=hashimkhan -Frep:password=hashimkhan -Fprofile/gender=male http://localhost:4502/libs/granite/security/post/authorizables |
| **Create a new User as a member of a Group** | | | curl -u admin:admin -FcreateUser=testuser -FauthorizableId=hashimkhan -Frep:password=hashimkhan -Fprofile/gender=male |
| **Add a Property to an existing User** | | | curl -u admin:admin -Fprofile/age=25 http://localhost:4502/home/users/h/hashim.rw.html |
| **Add User to a Group** | | | curl -u admin:admin -FaddMembers=testuser1 http://localhost:4502/home/groups/t/testGroup.rw.html |
| **Remove a User from a Group** | | | curl -u admin:admin -FremoveMembers=testuser1 http://localhost:4502/home/groups/t/testGroup.rw.html |
| **Set a User’s Group Memberships** | | | curl -u admin:admin -Fmembership=contributor -Fmembership=testgroup http://localhost:4502/home/users/t/testuser.rw.html |
| **Delete user and Group** | | | curl -u admin:admin -FdeleteAuthorizable= http://localhost:4502/home/users/t/testuser curl -u admin:admin -FdeleteAuthorizable= http://localhost:4502/home/groups/t/testGroup |
| **Change a user password** | | | curl -u testuser:OLD\_PWD -F rep:password=”NEW\_PWD” http://localhost:4502/home/users/t/testuser.rw.html curl rep:password=”test” –user admin:admin http://localhost:4502/home/users/a/alister@geometrixx.com |
| **Backup Commands** | | | |
| **Initiate a Backup to a Target folder** | | | http://localhost:4502/system/console/jmx/com.adobe.granite%3Atype%3DRepository/op/startBackup/java.lang.String?target=C:\sample\backupTest.zip |
| **Stop a running Backup** | | | curl -u admin:admin -X POST http://localhost:4502/libs/granite/backup/content/admin/backups.cancel.html |

### AEM LOGS

AEM provides out of the box log files for different purposes:

1. **REQUEST.LOG**
2. **ACCESS.LOG**
3. **STDOUT.LOG**
4. **STDERR.LOG**
5. **AUDIT.LOG**
6. **HISTORY.LOG**
7. **ERROR.LOG**
8. **UPGRADE.LOG**

**LOG FILE IN DETAIL**

* **REQUEST.LOG**: This log file contains each request together with response related to AEM instance. By this log file we can easily monitor the performance of AEM instance.

|  |  |
| --- | --- |
| Each request contains the following information:   * Method (GET, POST,HEAD,PUT,DELETE) * Resource Path (/content/geometrixx-outdoors/en.html) * Protocol (HTTP/1.1 or HTTPS) | Each response contains the following information:   * Status Code (404,200,401 etc) * MIME Type: (text/html, image/jpeg etc) * Response Time (518 ms) |

|  |
| --- |
| request.PNG |
| Fig- Request.log |

By analyzing, we can easily determine that which request and response is taking too much time. By this we can increase overall site performance.

* **ACCESS.LOG**: This log file is almost similar to request.log. It helps us to know the following things:
* Who is accessing
* Which resource is being accessed
* At what time resource is being accessed

The requests in access.log are exactly half of request.log because the access.log defines all the request and response in just one line.  
Each line in Access.log contains the following information:

* Access IP - The IP of that system from where the request is coming.
* Access User - The user who is sending the request.
* Access Time - At what time resource is being accessed.
* Request Method- The method of the request like GET, POST, PUT etc.
* Access Resource - The resource which is accessed by user.
* Protocol like HTTP/1.1 or HTTPS
* Response Status Code like - 404, 200, 401 etc.
* Content Length - It determines the length of response content.
* Referrer URL - The referrer is the web page that sends visitors to your site using a link. In other words, it’s the web page that a person was on right before they landed on your page.
* User Agent- It refers to a web browser telling website information about the browser and operating system.

|  |
| --- |
| accesslog.jpg |
| Fig -  Access Log |

* **STDOUT.LOG**:
* This log file basically contains the startup log. Capture logs whenever any AEM instance created or started then it will hold the information about AEM instance.
* Stdout.log is also useful to see the message given in java classes.This can be done by using System.out.println();

|  |
| --- |
| stdout.PNG |
| Fig - Stdout log |

* **STDERR.LOG** - This log file contains error messages of varying levels of severity. Some AEM instance related messages generated during startup Like: The JVM reports a heap size of 3618 MB, meets our expectation of 1024 MB +/- 20 .

**Note:  By default the log level is set to Warning (WARN).**

|  |
| --- |
| stderr.PNG |
| Fig-  Stderr.log |

|  |
| --- |
| 5**. AUDIT.LOG**- This log file is used by Jackrabbit to log changes to the repository. By default this log file is set to level info. There is not much logging going on at this level because of which audit.log seems to be empty. If you set this log file to log level debug you will see log messages every time you make changes to the repository. For example if you add a component to a page, delete a page etc.  audit.png |
| Fig - Audit.log |

6**. HISTORY.LOG**:

This log file contains the information about which action is performed on the content.

It contains high level logs about the actions performed by editors (actions can be edit, view, delete etc).This log is probably only relevant on author instances.

Each line contains the following information:

* TimeStamp
* Action
* User
* Content path
* Content type(cq:page, dam:asset)

7. **UPGRADE.LOG** 🡪 Provides a log of all upgrade operations that runs from **com.day.compat.code upgrade and com.adobe.cq.upgradesexecutor package**.By default, this log level is set to info.

8. **ERROR.LOG** 🡪 Error messages (of varying levels of severity like INFO, ERROR, DEBUG, etc.) are registered here.

Hierarchy of log4j logging levels are as follows in Highest to Lowest order

TRACE 🡪 DEBUG 🡪 INFO 🡪 WARN 🡪 ERROR

Note: The high level logs contains all the logs of lower levels like in case of Trace log level, it will include all the logs of debug, info, warn and error as well.

### SLING RESOURCE MERGER

Sling Resource merger in aem is one of the most commonly and frequently used feature of sling after aem 6.0. Due to limited functionality of touch ui components, we often are required to overlay/ override a component from /libs to /apps. Earlier (before aem 6.0) in case of overlay we need to copy paste entire component structure with node type and properties and then we make our changes which indirectly increases the overhead on /apps folder.

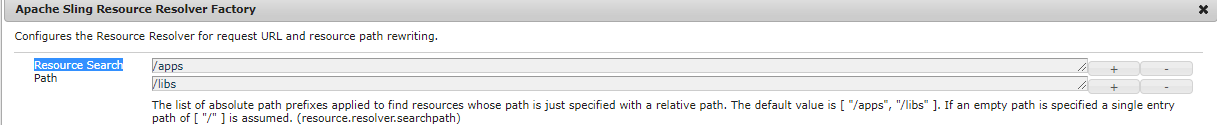
But as adobe is promoting to use its new touch optimized ui for granite components , it have changed the definition of overlay for Touch ui (Granite) components where you need to create only similar skeleton structure (where nodes can be of type nt:unstructured) and you can add, remove or modify existing node.

**SLING RESOURCE MERGER:** The Sling Resource Merger provides services to access and merge resources.

**OVERLAYING**

Usually when we overlay a component in AEM, then we copy component from /libs/ folder to /apps/ folder. And we can write our own customization on the newly copied components under /apps/.

* As per the default OSGI preferences AEM uses a search path to find a resource, searching first the /apps/ branch and then the /libs branch so our newly copied components under /apps/ gets priority over /libs/. But we change this preference from Felix console by modifying **Apache Sling Resource Resolver Factory** configuration (not recommended).
* When we overlay a component your both component that is apps and libs will be displayed in side kick.



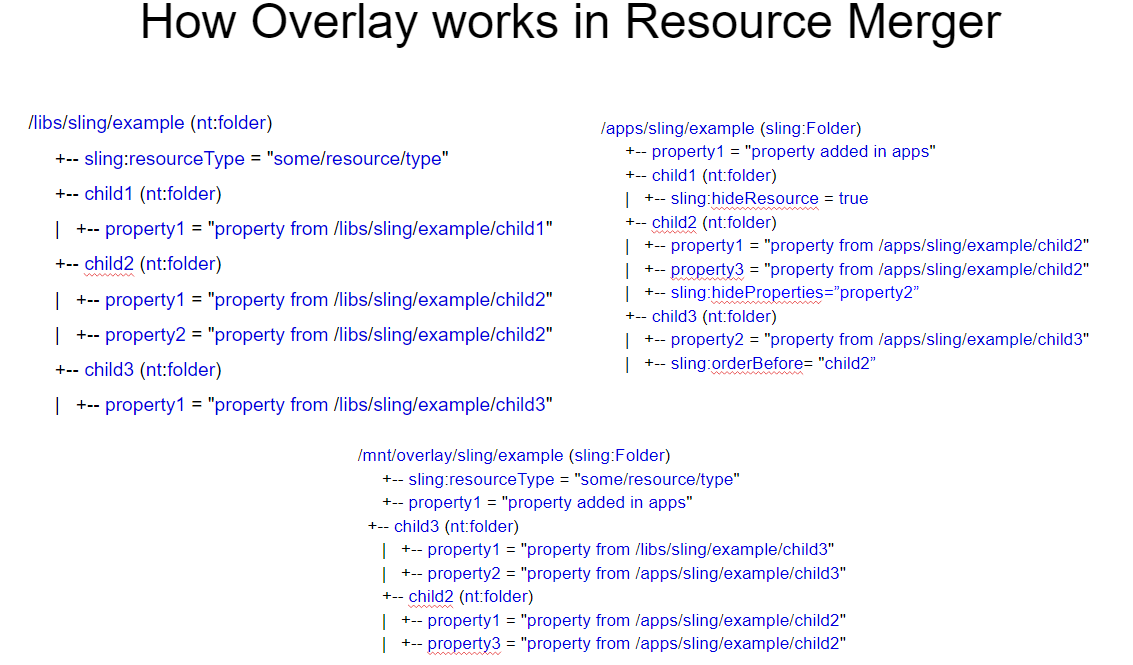
**WITH AEM 6.0 ONWARDS, AFTER INTRODUCTION OF TOUCH UI FOR GRANITE RELATED OVERLAY WE ARE USING SLING RESOURCE MERGER**

* If we have to overlay a component or use sling resource merger for a component then we need not required to copy paste entire component structure with node properties from libs to apps.
* We only need to recreate the skeleton structure. **To simplify the recreation of the structure all intermediary nodes can be of type nt:unstructured** (they do not have to reflect the original node type; for example, in /libs).
* Sling Resource Merger and its related methods can only be used with granite (touch ui) components.

**ADVANTAGES OF SLING RESOURCE MERGER**

* Ensure that customization changes are not made in /libs.
* Reduce the structure that is replicated from /libs.

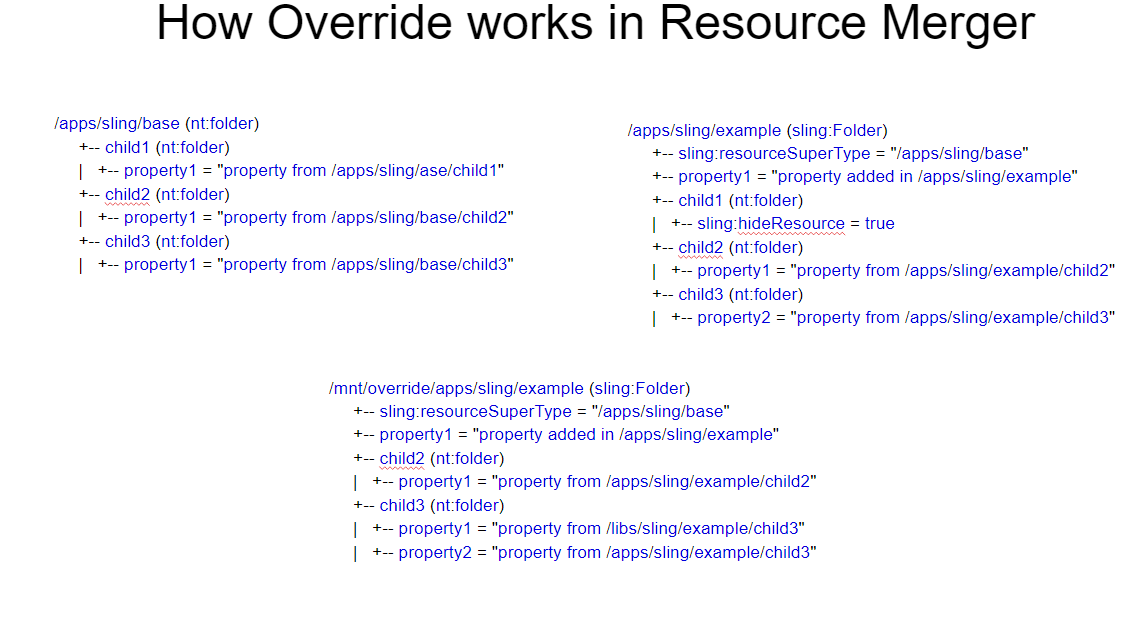
|  |  |  |  |
| --- | --- | --- | --- |
| **EXAMPLE**  Lets say , we want to modify navigation option(say project) in the navigation by overlaying using Sling resource merger  The navigation is generated from **/libs/cq/core/content/nav.** Let overlay this node for our requirement | |  | |
|  |  | | |
| project node in libs (/libs/cq/core/content/nav/projects) | | | Project node in apps (/apps/cq/core/content/nav/projects) |
| * Right click the node (in libs) 🡪Overlay Node * Similar structure gets created in apps folder. Note , the node created in the apps directory is nt:unstructured type * Now if want ro modify the jcr:title property – just add it to project node of apps * project node is gets the remaining property from libs (project node). So this is how resource is getting merged (libs+ apps) * only those property are modified , which we want to cutomize | | |  |



**OVERRIDE**

* Overriding a component is basically extending or inheriting the component using sling:resourceSuperType property. You can override a component from /libs by creating a custom component under apps manually and adding all necessary nodes and setting value of sling:superResourceType property to that component will inherit all the feature from /libs/ component, even after upgrade you still inherit the features of image component.
* Here we can use the sling:superResourceType for any component that you want to inherit functionality
* When to use overlay/sling resource merger and when to use override totally depends upon your requirement.

|  |  |  |
| --- | --- | --- |
| Lets say we want to modify the page properties dialog  I have my page component which is inherits page component of core (/apps/core)  (**using sling:resourceSuperType**) /apps/core/wcm/components/page/v1/page | | overriding.PNG |
| Since it inheriting the page component , it will inherit is cq:dialog too.  Lets say we want to modify the property of this dialog using overriding the cq:dialog node  Step1**: Copy the cq:dialog node from /apps/core/wcm/components/page/v1/page/cq:dialog to our project’s page component e.g. /apps/company/components/pagecomponent/home** | | |
| **LETS SAY WE WANT TO MODIFY THE TITLE OF THE BASIC TAB** | | |
|  | * Since the changes has to be done in basic tab , we have removed all the other nodes * We have to just mofify the jcr:title property in the basic node. * Note – basic node is getting all other properties from libs. * This is the ways we are merging the resource of apps + libs * The same rule holds good for cq:dialog node as it is getting all other nodes from libs(like advanced etc.) . Only basic node is kept at the apps level , which has been modified at apps level. | |
|  | | |



**PROPERTIES OF RESOURCE MERGER**

|  |  |  |
| --- | --- | --- |
| sling:hideProperties | String or String[] | Hide the properties,The wildcard(\*)hides all. |
| sling:hideResource | Boolean | Indicates that the resource should be completely hidden with its children |
| sling:hideChildren | String or String[] | Hide the list of children of a particular resource. The wildcard(\*)hides all the children. |
| sling:orderBefore | String | Contains the name of the preceding sibling. |

**USE CASES OF SLING RESOURCE MERGER**

* Add a property
* Redefine a property(not auto-created properties)
* Redefined an auto-created property
* Redefine a node and its children
* Hide a property
* Hide a node and its children
* Hide Children of a node (while keeping the properties of the node)
* Reorder nodes

**INVOKING THE SLING RESOURCE MERGER FROM CODE**

The Sling Resource Merger includes two custom resource providers - one for overlays and another for overrides. Each of these can be invoked within your code by using a mount point:

**Overlay**

* purpose: merge resources based on their search path
* mount point: /mnt/overlay
* Usage: mount point + relative path
* Example: getResource('/mnt/overlay/' + '<relative-path-to-resource>');

**Override**

* purpose: merge resources based on their super type
* mount point: /mnt/override
* Usage: mount point + absolute path
* Example: getResource('/mnt/override' + '<absolute-path-to-resource>');

Use Case: Suppose we want to add some more locales in the drop-down of page properties.

To know from where languages are getting listed out, I check the language widget in the page properties dialog.

|  |
| --- |
| https://lh6.googleusercontent.com/GZcTr1w1ETz-iMPkEDVtYLB0qvQZZ9UgikUF_MA1eB7MquzSK7Ck6W8sEpcrcG9DJfrK2q9vlADossuXbvt7fhNXM-LNfHrVMIm-e65039H0ENcc4LvO9NHnMuKzkdAFbz8uvI84 |
| Fig - language widget in the page properties dialog |

So to populate languages, cq/gui/components/common/datasources/languages are playing an important role.

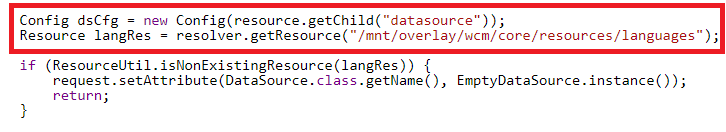
If I check the language.jsp under /libs/cq/gui/components/common/datasources/languages, you can see that all the languages are coming from /libs/wcm/core/resources/languages.

|  |
| --- |
| https://lh6.googleusercontent.com/xUjSQF_pQlWkHhNwjKJa5tpTeN4tHMQ15A75LnKAephbSHTQhedj5oNppDTTxLCeotKYYJKrKeLSed6RIjG2YKvb9T3-URvfCDsDIxqeASaB1N_TBqfOh8-3BYSSQlyTkchjfgCT |
| Fig - Source of language population in the dropdown |

Now if I add some more languages in /apps/wcm/core/resources/languages, then the jsp will start taking the values from /apps not /libs, because /apps is having preference over /libs.

But I don’t want to duplicate all the locales in apps as well. So what to do?

* 1. Overlaying the languages hierarchy: Just add all the new locales in /apps/wcm/core/resources/languages and now overlay the language.jsp with the following modifications.

Fig - Merging the new locale and existing locale using mnt overlay 

2. Overriding the languages hierarchy: Add new locales in any hierarchy example I have taken it “/apps/languages/core/resources/languages” and add a property sling:resourceSuperType to /libs/wcm/core/resources/languages.

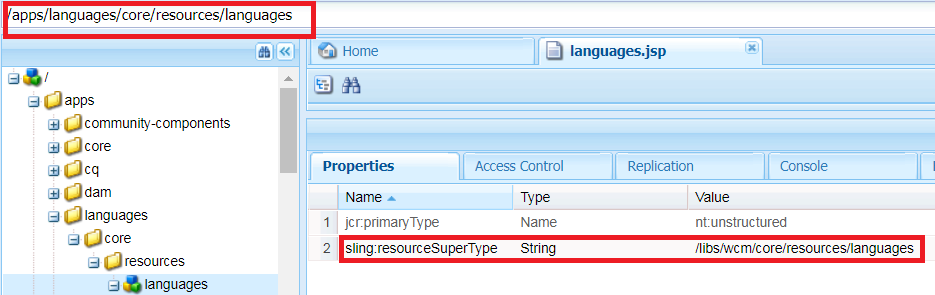


Fig - Overriding the languages which are available under libs

Overlay the language.jsp with the following modifications.

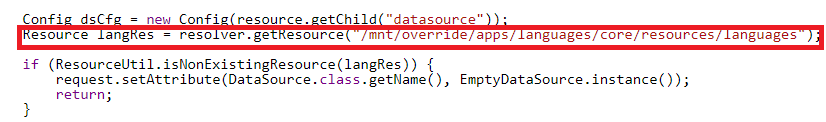


Fig - Merging the new locale and existing locale using mnt override

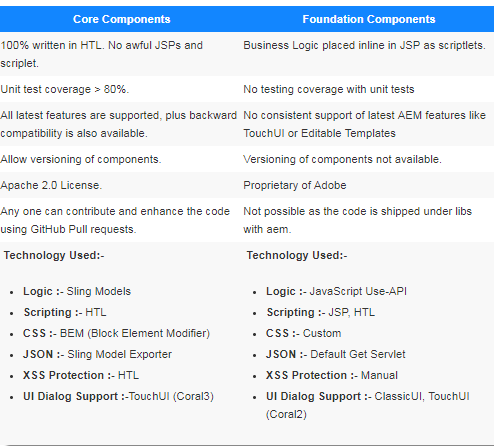
### CORE COMPONENTS

|  |  |
| --- | --- |
| * AEM core components are introduced in aem 6.2 but are strongly recommended to use from aem 6.3 because by overlaying or extending them - speed up the over all component development process * Help make extensible components in aem that are secure, robust, version-able.   **version-able** means when new version of aem comes it will not affect your current functionality. And if you want to use latest features available in new version you have to just change reference from old to new version of core components in your custom component. E.g **v1** denotes the version number of text component and if a new version comes in it will be named as **“v2”**  Note: - Core Components require **AEM 6.3 and Java 8**. Core Components do not work with the Classic UI.   * Path - /apps/core or /apps/core/wcm/components |  |

**CORE COMP HIERARCHY**

|  |  |
| --- | --- |
| There are   * Around 70 Foundation components * Around 12 core components out of which   + 7 are page authorable components   + 5 are form components   **Page Authorable core components:-**   1. Page 2. Breadcrumb 3. Title 4. Text (RTE) 5. Image 6. List 7. Sharing   **Form core components:-**   1. Container 2. Text field 3. Options field 4. Hidden field 5. Button | wcm-components-hierarchy |

**DIFFERENCE BETWEEN CORE AND FOUNDATION COMPONENTS /ADVANTAGE OF CORE COMP**

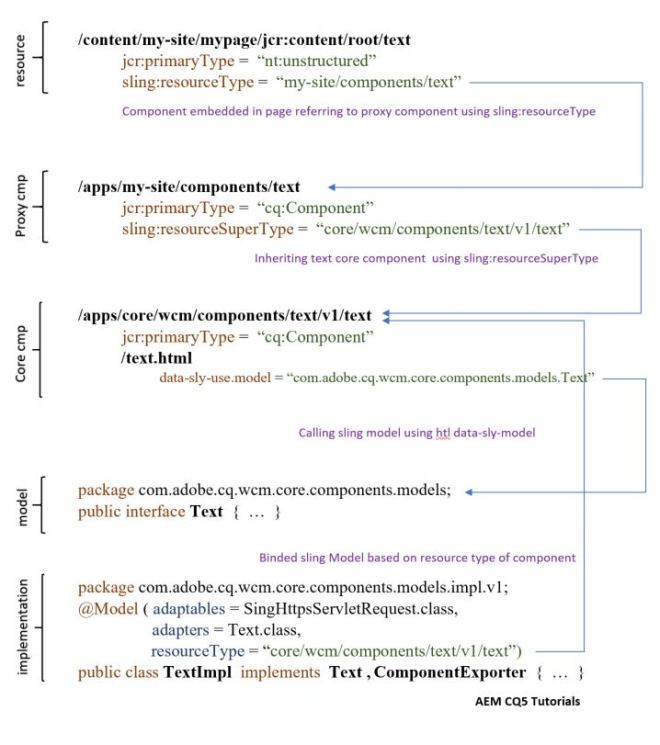


**Note : To use core components we need to create proxy components in aem.**

**WHAT ARE PROXY COMPONENTS OR PROXY PATTERNS? WHY I SHOULD USE THEM? CAN’T I DIRECTLY USE COMPONENTS LIKE I USED TO USE FOUNDATION COMPONENTS EARLIER?**

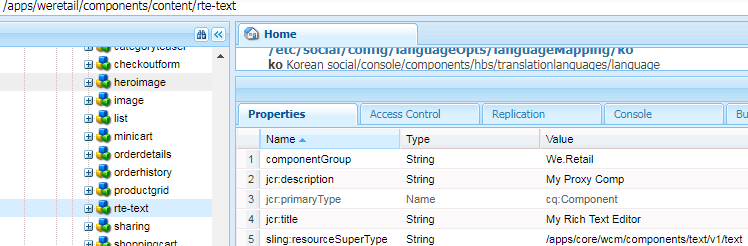
Answer - Core components are hidden by default, you cannot use them directly. Core components are designed in such a way that it should be accessed using proxy components only. The advantage of adding one more layer of proxy components is that your core components will be as it is and you can refer to any version of core component just by changing sling:resourceSuperType to new version.

The below fig. explains how the component implementation is broken down in AEM



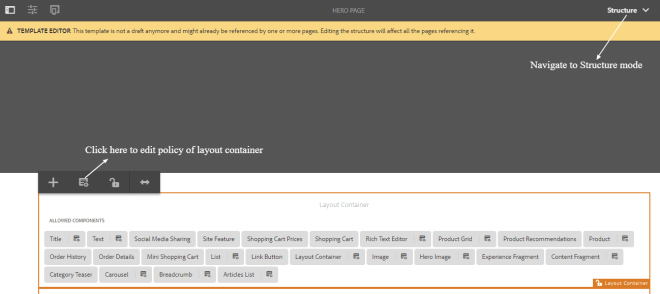
**EXAMPLE TO CREATE CORE COMPONENT**

* Navigate to /apps/weretail/components/content/.
* Create a new node of type cq:Component. (This component is known as proxy component). Click on Save All.

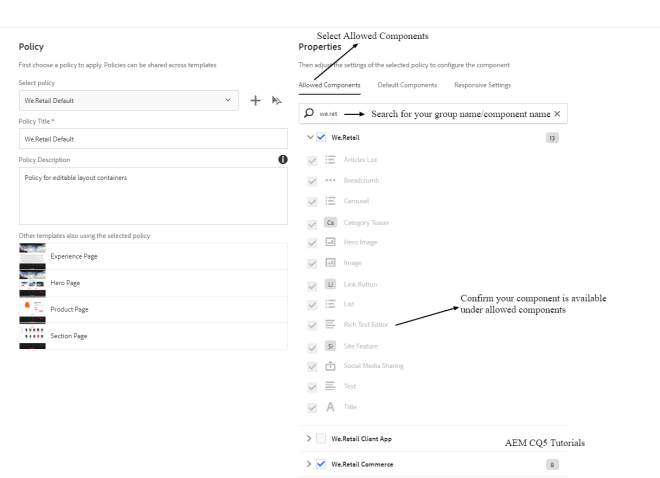


You can use this component now on both editable template and static template. For using it on we retail editable template. Follow below steps.

* Edit the template in <http://localhost:4502/editor.html/conf/we-retail/settings/wcm/templates/hero-page/structure.html>
* Edit the policy for the Layout Container



* Add your component to the list of allowed components. Click save on top right corner after making your changes.
* Go to <http://localhost:4502/editor.html/content/we-retail/ca/en/men.html> .
* Add your component into the parsys.



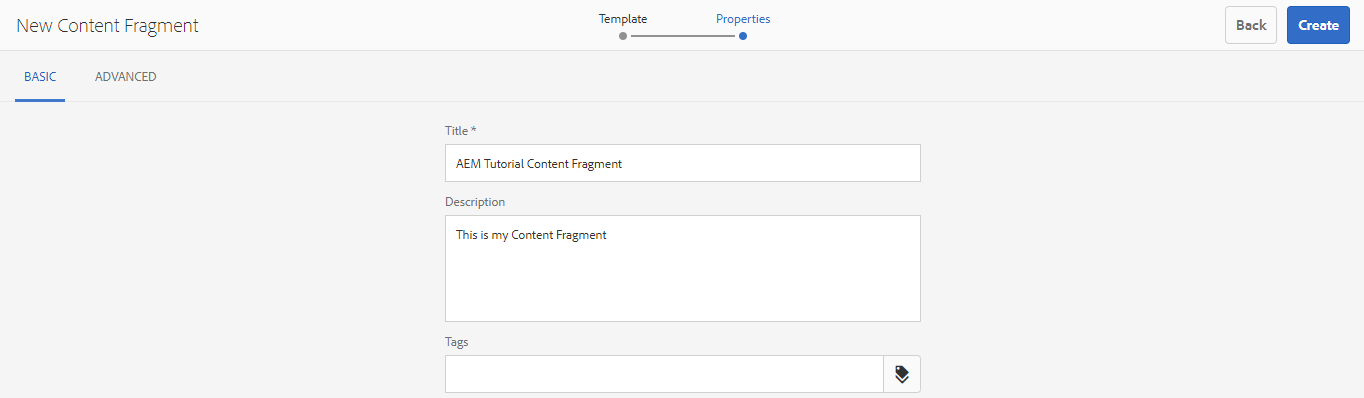
### CONTENT FRAGMENTS

* Intoduced in AEM-6.2 version.
* Content Fragments help us to create page independent content. So advantage here we get is - author don’t have to wait for the pages to be created.
* They are stored as asset and can be created and maintained from the AEM Asset Console.
* Content Fragment contains text elements and might include references to other asset like image or other content fragments.

|  |  |
| --- | --- |
| Image result for content fragments | * In Content Fragment we create Master content of the Content. * From Master content we can create multiple variations that can be used in different channels like website, mobile App and campaign.   **Content Fragment Components**   * 1. Fragment Element   2. Variations   3. Fragment Paragraph   4. Fragment metadata   5. Associated content   6. Fragment template   7. Content Fragment Components |

#### Creating Content Fragment

* Go to the AEM console <http://localhost:4502/aem/start>
* Assets 🡪Files 🡪Create 🡪Content Fragment
* Select the Simple Fragment Teamplate 🡪Next (Simple Fragment is a OOTB teamplate – We can create new teamplate as per our requirement)
* Enter the title 🡪Create



**CONTENT FRAGMENT CREATED (MASTER COPY)**

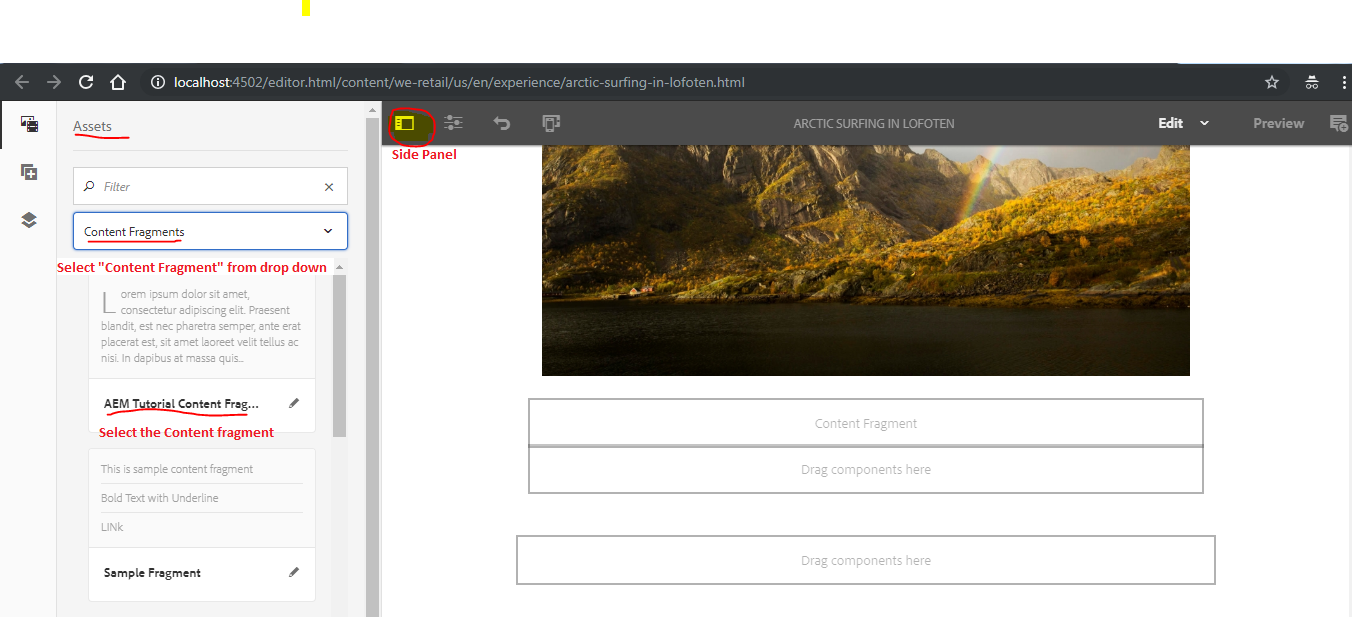
|  |  |
| --- | --- |
|  |  |
| **MASTER**  Enter the content in the RTE of Master 🡪Save | |

**CREATING VARIATIONS**

|  |  |
| --- | --- |
|  | * Click of Create Variation * Update the content (as per requirement) in the variation |

**AUTHORING CONTENT FRAGMENT**

* Open the page where we want to drop the content fragment
* Click on a responsive grid 🡪 Click in (+) 🡪Select Content Fragment
* Toggle The Side Panel
* Select ”Content fragment” from drop down
* Drag and drop the desired content fragment. This will show the Master copy on the page

****

|  |  |
| --- | --- |
| **SELECTING VARIATION**   * Edit the content fragment and select the variation e.g “Mobile App” variation |  |

#### Content Fragment Model / Content Fragment Template

* Introduce in AEM 6.4
* Using Content Fragment Model we can define a desired structure of the content.
* **Content Fragment Model works as template, while creating content fragments.**

**STEPS OF CREATING CONTENT FRAGMENT MODEL**

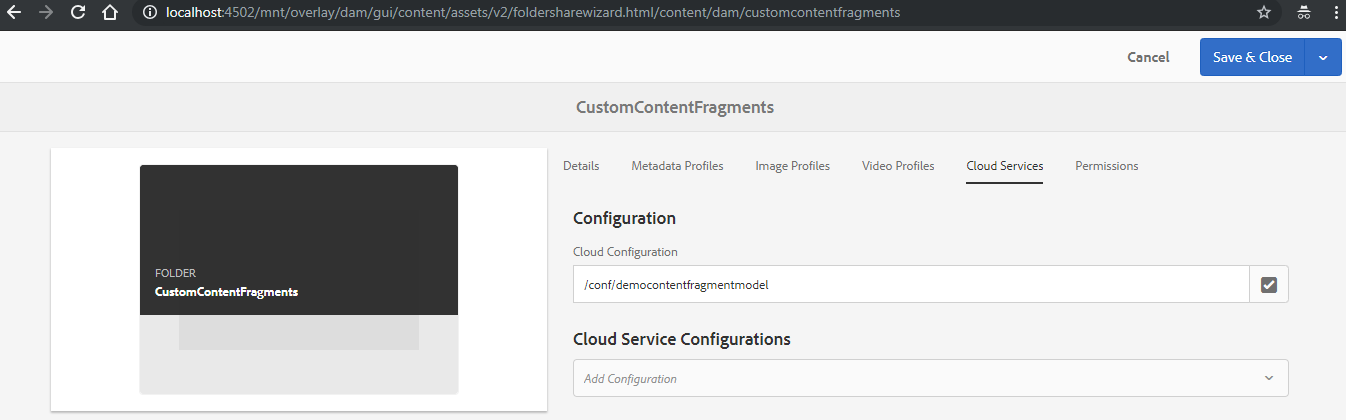
1. **Enable content Fragment Model from Configuration Manager**
2. **Create Content Fragment Model**
3. **Apply the configuration to your asset folder (Keep all you project specific content fragment in a specific folder)**

**ENABLE CONTENT FRAGMENT MODEL FROM CONFIGURATION MANAGER**

|  |  |
| --- | --- |
| * Go to Tools 🡪 General 🡪 Configuration Browser | * Click on Create to create configuration |

**APPLY THE CONFIGURATION TO YOUR ASSET FOLDER**

* Go to Asset from Navigation 🡪Files 🡪 Create a folder where you want to store you Content Fragment Model
* Select the folder and Click on “**Properties**”
* Go to Clould Service Tab and configure the Content Configuraation Model Configuration created on Step 1 🡪 Save & Close

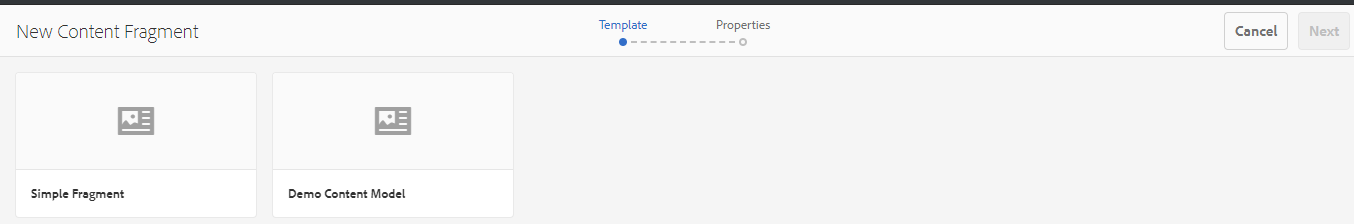


**CREATE CONTENT FRAGMENT MODEL**

|  |  |
| --- | --- |
|  | * After creating the configuration Go to Tools 🡪Asset 🡪Content Fragment Model 🡪 Select the Configuration created in Step 1. * Click on Create and Enter the Title of the content fragment model. * Then Click on Create 🡪 Open * This will open Content Fragment Model Editor |
|  | |
| * Once we open the Content Fragment Model Editor we can design the template as per our requirement * For example - Drag & drop the datetypes (field types) to create the template as below * The properties of each data types can be set from the **Properties Tab** * Save the template. | |

**USING THE CONTENT FRAGMENT MODEL**

* The content fragment model can be used in the similar steps as we followed in Creating a Content Fragment
* Apart from Simple Content Fragment we can see one more teamplate we have created in above steps.
* Create the content fragment by following the simiar steps and author the same on the page



### EXPERIENCE FRAGMENTS

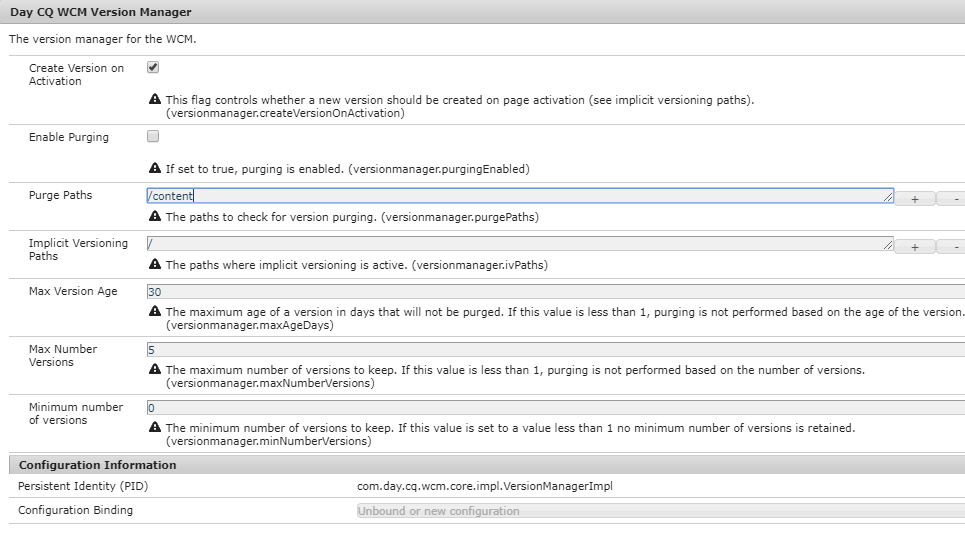
### VERSIONING

|  |  |  |
| --- | --- | --- |
|  |  | * Go to site console and select the page for which we want to check the version * Select the page * In the left pane. Select Timeline 🡪Versions. This will show the different versions created for a page * This functionality can be controlled / customized using a OSGI configuration “**Day CQ WCM Version Manager**” |

### VERSION MANAGER

AEM creates a version of the page, whenever the content is updated and activated.

**HOW TO CHECK THE VERSION OF PAGE**



|  |  |
| --- | --- |
| Create Version on Activation | Check if we want to create a version when pages are activated. |
| Enable Purging | Whether to enable purging when new versions are created |
| Purge Paths | Paths on which versions are implicitly created on activation if “Create Version on Activation” is true. |
| Implicit Versioning Paths | Paths on which versions are implicitly created on activation if “Create Version on Activation” is true. |
| Max Version Age | On purge, any version older than this value will be removed. If this value is less than 1, purging is not performed based on the age of the version |
| Max Number Versions | On purge, any version older than the n-th newest version will be removed. If this value is less than 1, purging is not performed based on the number of versions |
| Minimum number of versions | The minimum number of versions to keep regardless of the age. If this value is set to a value less than 1 no minimum number of versions is retained. |

### CREATE MULTIPLE BUNDLES IN AN AEM MAVEN PROJECT

In AEM, Maven provides multi-module architecture.It means that we can have different modules like bundle and content (core, ui.apps) in a single project. So let’s see how we can create multiple bundles in the same project.

|  |  |  |
| --- | --- | --- |
| Initially, when we create an AEM Project, we always see two modules:   * Bundle * Content | modules.PNG | These modules are defined under parent pom.xml.  modules.PNG |

**PROBLEM STATEMENT: I WANT TO CREATE MULTIPLE BUNDLES IN THE SAME AEM PROJECT. THE USE CASES OF DOING THIS CAN BE:**

* Need a bundle to run in publish instance only or need to deploy a bundle on the basis of specific locations.
* Want to create multiple bundles in a single project on the basis of big functionalities like Login Module, Payment Module etc.

Solution: It is very easy, you just need to follow the below Steps:

|  |  |
| --- | --- |
| * Create the new bundle by just copying the existing bundle of a project. | add a new bundle.PNG |
| * Rename the bundle as test * Now In Pom.xml of test bundle and change the ArtifactID (Because artifact should be unique for each module of the project). | newbundle.jpg |
| Change the “Bundle-SymbolicName” in the Test Module Pom.xml. | change-symbolic-name.jpg |
| Add a new module in parent Pom.xml | https://lh6.googleusercontent.com/t-nC_vANbPWjLcNYET7rbw4dtVh9MaUux0qID2PxjjbLbadAv-r1erIyeC8NXs6UzUQ_F3vV42_Hj_-_Lo-uUhEtH9N9DzcmceXTJhsV33E8t4rIftUhocyFveO5OZHhS2DegQ11 |
| Content Pom.xml  contains the dependency of all the bundles, so we need to add the dependency of the new bundle(test-module) in this Pom.xml | add dependency.PNG |
| For deploying the bundles at any specific location we need to specify the location in the "target" (like install) and bundle‘s groupId and artifactid in the embedded section. | deploy.PNG |
| Now you just need to build the code and you can find the bundle’s jar at expected location as you configured in content Pom.xml in embedded section. | crx.PNG |

### [FREQUENTLY USED CQ URLS](http://cq-ops.tumblr.com/post/21045033313/frequently-used-cq-urls)

|  |  |
| --- | --- |
| Component | URL |
| Main (Welcome) Page | /libs/cq/core/content/welcome.html |
| Website Administrator | /siteadmin |
| Digital Asset Manager | /damadmin |
| Notification InBox | /libs/cq/workflow/content/inbox.html |
| **SEARCH** |  |
| Search UI: | /crx/explorer/ui/search.jsp?Path=&Query= |
| Query Debugger | /libs/cq/search/content/querydebug.html |
| **CLIENT CONTEXT** | /etc/clientcontext/default/content.html |
| **TRANSLATOR SETTINGS** | /libs/cq/i18n/translator.html |
| **ADMINISTRATION** |  |
| Administration Tools: | /miscadmin |
| Backup: | /libs/granite/backup/content/admin.html |
| Mobile Administration: | /miscadmin#/etc/mobile |
| Site Templates (Blueprints): | /miscadmin#/etc/blueprints |
| Site Designs: | /miscadmin#/etc/designs |
| Tag Management: | /libs/cq/tagging/content/tagadmin.html |
| User Segment Management: | /miscadmin#/etc/segmentation |
| Multi-Site Manager Rollout Configurations: | /miscadmin#/etc/msm/rolloutconfigs |
| Digital Asset Manager (DAM): | /damadmin#/content/dam |
| Feed Import Management: | /miscadmin#/etc/importers |
| Cloud Services: | /etc/cloudservices.html |
| **PACKAGES** |  |
| Local Package Repository | /crx/packmgr |
| Cloud-based Package Repository | /crx/packageshare |
| **DEVELOPMENT** |  |
| CRX DE Lite: | /crx/de |
| **TROUBLESHOOTING** |  |
| Code Profiling: | /system/console/profiler |
| Disk Performance Benchmarking | /system/console/diskbenchmark |
| **WORKFLOW** |  |
| Workflow Models: | /libs/cq/workflow/content/console.html |
| Notification InBox: | /libs/cq/workflow/content/inbox.html |
| **REPLICATION** |  |
| Replication Management: | /etc/replication.html |
| Activate (Replicate) a tree: | /etc/replication/treeactivation.html |
| Agents on Author: | /etc/replication/agents.author.html |
| Agents on Publish: | /etc/replication/agents.publish.html |
| Dispatcher Flush Agent | /etc/replication/agents.publish/flush.html |
| **REPORTS** |  |
| Display Client Libraries in Use by Component: | /libs/cq/ui/content/dumplibs.html |
| Author Activity Audit: | /etc/reports/auditreport.html |
| Disk Usage Report: | /etc/reports/diskusage.html |
| Disk Storage by DAM Assets: | /etc/reports/diskusage.html?path=/content/dam |
| User Report: | /etc/reports/userreport.html |
| **REPOSITORY** |  |
| Content Explorer | /crx/explorer/browser/index.jsp |
| Node Type Administration | /crx/explorer/nodetypes/index.jsp |
| Maintenance Tools: | /system/console/jmx/com.adobe.granite%3Atype%3DRepository |
| **SYSTEM ADMINISTRATION** |  |
| Clustering Administration: | /libs/granite/cluster/content/admin.html |
| Apache Felix Administration Console: | /system/console |
| OSGi Bundle Configuration: | /system/console/configMgr |
| JVM Runtime Properties: | /system/console/jmx/java.lang%3Atype%3DRuntime |
| JVM Memory usage: | /system/console/memoryusage |
| System Information: | /system/console/vmstat |
| Product Version and License Information: | /system/console/productinfo |
| Performance Profiler: | /system/console/profiler |
| Disk Performance Benchmark Tool: | /system/console/diskbenchmark |
| List of Backups: | /libs/granite/backup/content/admin.html |
| MIME Type List: | /system/console/mimetypes |
| Components, Licences, EULAs: | /system/console/licenses |

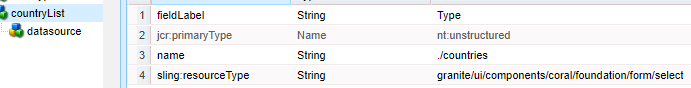
**CHANGE THE AUTHORING MODE**

|  |  |
| --- | --- |
|  |  |

### [ACS](http://cq-ops.tumblr.com/post/21045033313/frequently-used-cq-urls) COMMONS

#### DYNAMICALLY POPULATE DROP DOWN VALUES OF DIALOG

1. Download the compartible version of ACS commons package for your AEM version.
2. Create a nt:unstructured node for select box
3. Create a node “datasource” and set the **sling:resourceType= acs-commons/components/utilities/genericlist/datasource**



|  |  |
| --- | --- |
| Open a misadmin console and create a page under Generic List |  |
| Add the key value pair in the page |  |

1. Add the path to datasource node . **path= /etc/acs-commons/lists/Counties**

|  |
| --- |
|  |