Contents

[AEM (ADOBE EXPERIENCE MANAGER) 3](#_Toc82195580)

[AEM TECHNOLOGY STACK 3](#_Toc82195581)

[STARTING AEM 3](#_Toc82195582)

[AEM PROJECT – MAVEN ARCHETYPE- 23 4](#_Toc82195583)

[RUN MODES 5](#_Toc82195584)

[DEBUGING BUNDLE CODE 8](#_Toc82195585)

[CREATING TEMPLATES 9](#_Toc82195586)

[CREATING COMPONENTS 15](#_Toc82195587)

[INITIALIZING WCM 23](#_Toc82195588)

[DIALOG & DESIGN DIALOGS 24](#_Toc82195589)

[CLIENTLIBRARY 29](#_Toc82195590)

[CLIENT-LIBRARY PROPERTIES 30](#_Toc82195591)

[OSGI 32](#_Toc82195592)

[OSGI CONFIGURATION – USING R7 ANNOTATIONS 36](#_Toc82195593)

[SLING SERVLET 41](#_Toc82195594)

[SCHEDULERS 45](#_Toc82195595)

[EVENT HANDLING 48](#_Toc82195596)

[(CAAS) - SLING MODEL EXPORTER 51](#_Toc82195597)

[HTML TEMPLATE LANGUAGE (HTL) – SIGHTLY 54](#_Toc82195598)

[TAGGING 61](#_Toc82195599)

[WORKFLOW 64](#_Toc82195600)

[REPLICATION 75](#_Toc82195601)

[VLT (VAULT) TOOL 75](#_Toc82195602)

[AEM RESPONSIVE GRID SYSTEM 77](#_Toc82195603)

[SITEADMIN – IN DETAIL 78](#_Toc82195604)

[SLING SCRIPT RESOLUTION 81](#_Toc82195605)

[SECURITY 82](#_Toc82195606)

[QUERYBUILDER API 85](#_Toc82195607)

[MULTI SITE MANAGER 86](#_Toc82195608)

[INTERNATIONALIZATION 101](#_Toc82195609)

[PERSONALIZATION 103](#_Toc82195610)

[SOME USEFUL CODE 116](#_Toc82195611)

[STEPS SETTING UP MONGO DB WITH AEM 6.0 117](#_Toc82195612)

[ENABLING MOBILE EMULATOR 118](#_Toc82195613)

[DISPATCHER 119](#_Toc82195614)

[CURL 130](#_Toc82195615)

[AEM LOGS 133](#_Toc82195616)

[SLING RESOURCE MERGER 136](#_Toc82195617)

[CORE COMPONENTS 141](#_Toc82195618)

[CONTENT FRAGMENTS 145](#_Toc82195619)

[VERSIONING 151](#_Toc82195620)

[VERSION MANAGER 151](#_Toc82195621)

[CREATE MULTIPLE BUNDLES IN AN AEM MAVEN PROJECT 152](#_Toc82195622)

[FREQUENTLY USED CQ URLS 154](#_Toc82195623)

[ACS COMMONS 156](#_Toc82195624)

# 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.

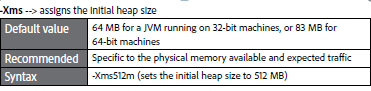
### STARTING AEM

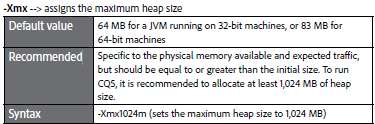
**Install or Start AEM Using a Command Line**

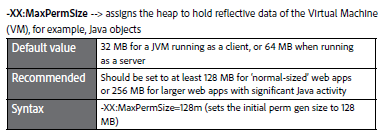
There are two ways to install **AEM: graphical and by command line**. The latter is more powerful because the user has the possibility to provide additional performance tuning parameters to the Java Virtual Machine (JVM). On Windows, MacOS X, or \*x, you can also install or start AEM from the command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

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; or 6.5.0, 6.4.4, or 6.3.3 for Adobe Managed Services or on-premise) |
| **sdkVersion** | latest | [When aemVersion=cloud an SDK version can be specified](https://docs.adobe.com/content/help/en/experience-manager-cloud-service/implementing/developing/aem-as-a-cloud-service-sdk.html) |
| **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 example site (y/n)](https://www.aemcomponents.dev/) |
| **includeErrorHandler** | n | Includes a custom 404 response page that will be global to the entire instance (y/n) |

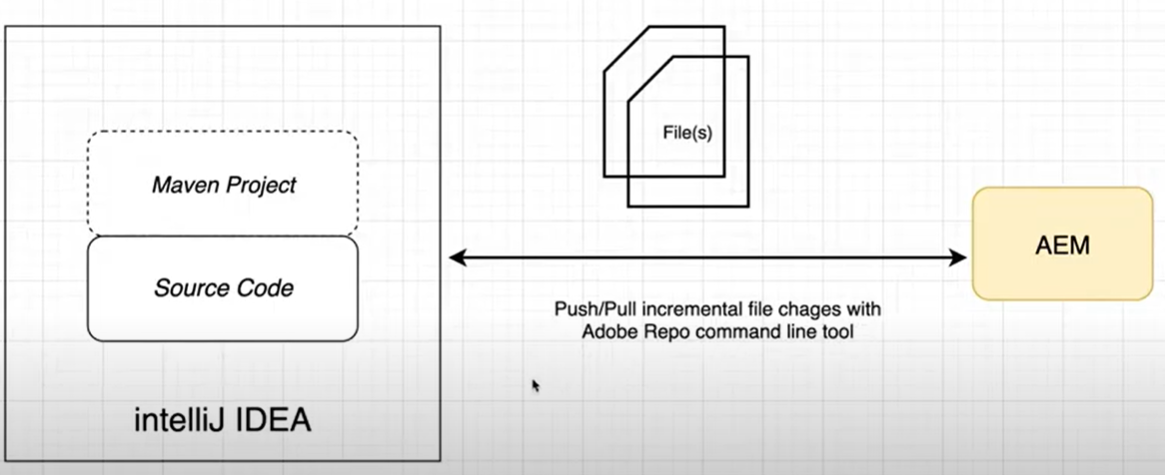
#### PROJECT STRUCTURE

|  |  |
| --- | --- |
|  | **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 DEPENDENCIES

#### 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

**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 (Installation Run mode)
2. Secondary Run Mode(Customized 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.

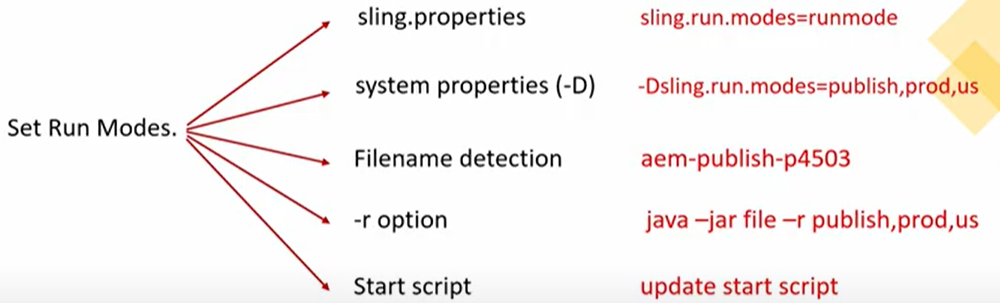
Note: Primary Run Modes can’t be changing once the aem jar is started. So, at the time of AEM instance startup only, we need to finalize which primary run mode is required.

#### SECONDARY RUN MODES

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

**CUSTOMIZED RUN MODES:** We 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.

#### SETTING RUN MODES

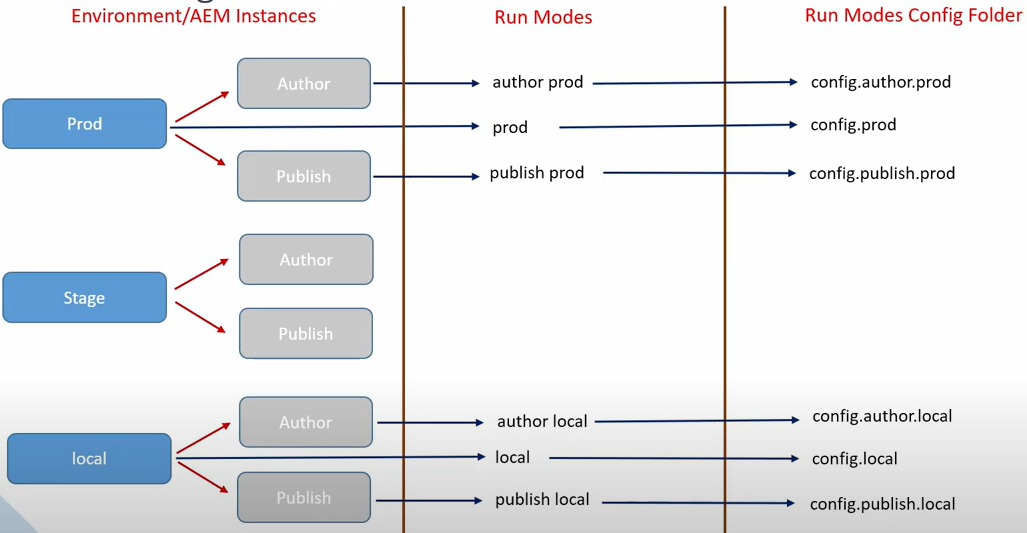


|  |  |
| --- | --- |
| **USING THE SLING.PROPERTIES** | **Step 1: Edit the configuration file:** <cq-installation-dir>/crx-quickstart/conf/sling.properties  **Step 2:** Add the following properties; the following example is for author: sling.run.modes=author |
| **USING THE -R OPTION:** | **A** custom run mode can be activated by using the -r option when launching the quickstart. To launch an AEM instance with run mode set to publish,prod and test.  **java -jar aem-author-p4503 -r publish,prod,test -p 4503** |
| **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> * For example - To set the publish run mode by naming the jar file: cq5-publish-p4503 |
| **USING A SYSTEM PROPERTY** | * 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 |
| **START SCRIPT** | * Navigate to : crx-quickstart\bin * Update the following   + CQ\_RUNMODE   + CQ\_PORT |

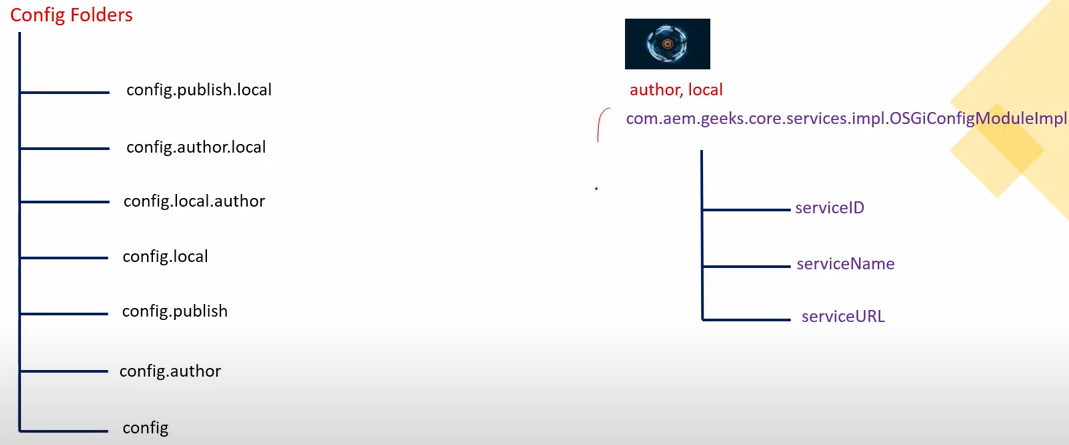
**HOW TO CHECK THE RUN MODE:** Felix Console 🡪 Status TAB 🡪 Sling Setting

#### OSGi CONFIGURATION WITH RUN MODES

|  |  |
| --- | --- |
|  | * In the below hypothetical scenario. Each environment has their specific runmodes for each instance. * If we want to create configuration of a specific run mode – we might have to create folder accordingly in our project. * The folder name convention is **config.<installation\_run\_mode>.<custom\_run\_mode> . Ex config.author.prod.T**he project created by new maven achetype project creates the config folders for different run modes. * The environment specific configuration are kept in config.<env>. For ex – config.prod (This config will be applicable to both author and publish in PROD environment) * The configurations a which are common across all the runmodes are added inside **“config”** folder |



#### CONFIG FOLDER RESOLUTION



In the above example , lets say the instance has run mode =author,local. AEM will resolve the config folder in below order

* config.author.local 🡪config.local 🡪 config
* if the configuration has only two folders config.author.local & config.local.author. The resolution order will be **config.author.local 🡪 config.local.author.** The preference is always given to the primary 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

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** |
|  |  |
|  | |

### DEBUGING BUNDLE CODE

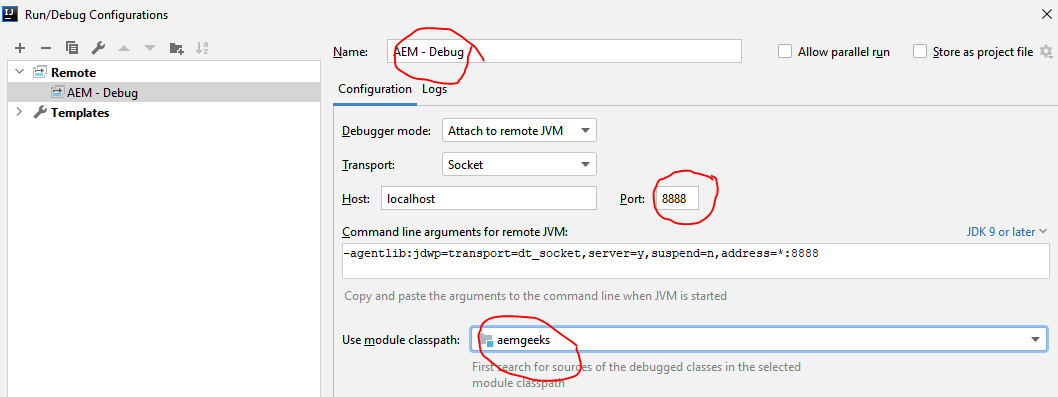
* Start AEM in Debug Mode using command

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

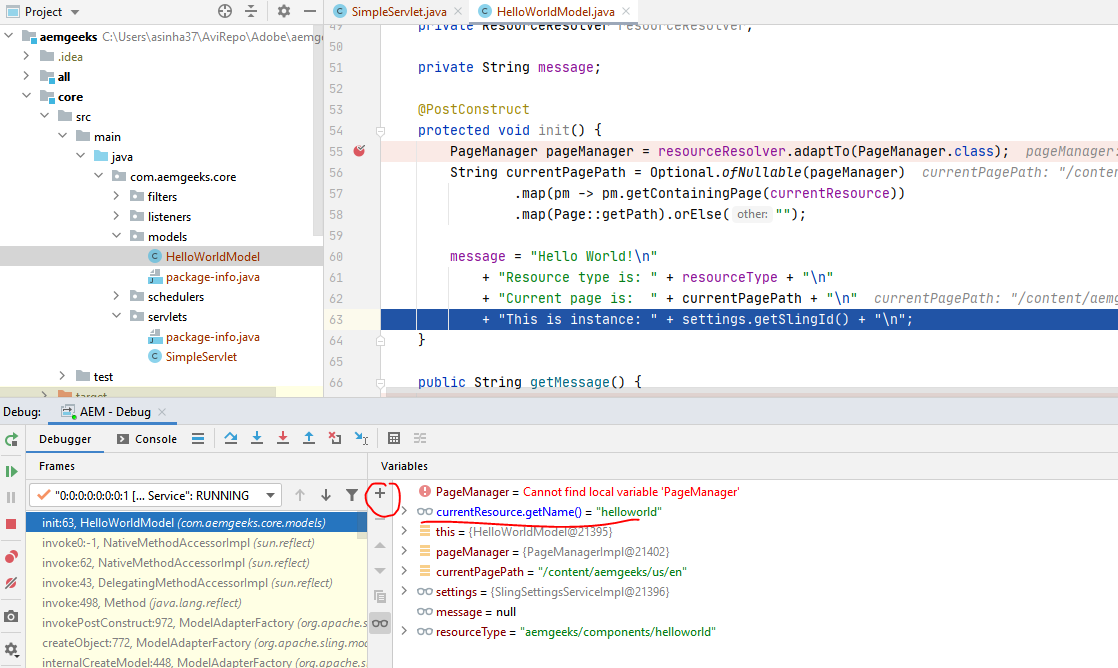
AEM will still run on 4502 port but allow another debugger port 8888 to debug

#### 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 pathd 🡪 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 |  |

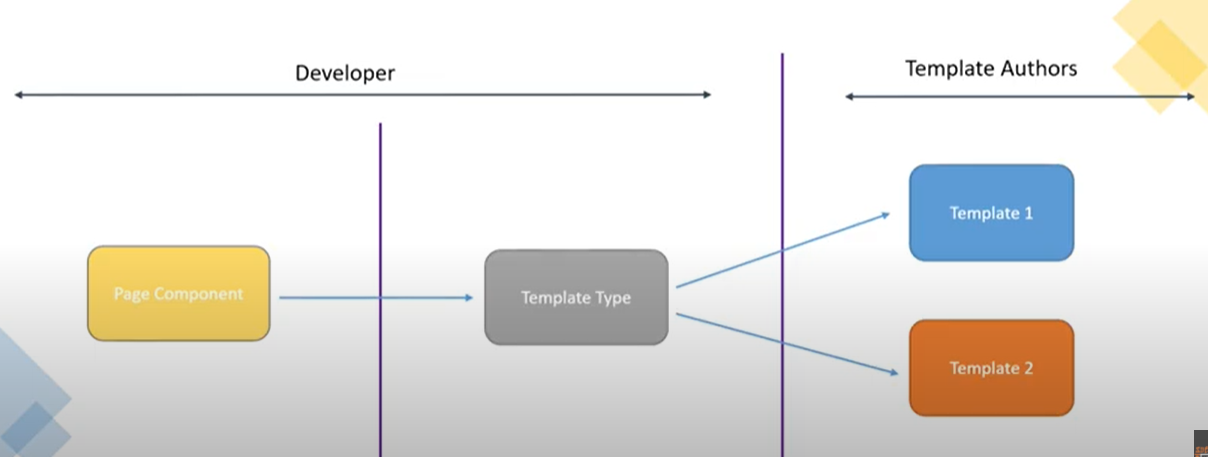
**Note🡪** While creating a template, if a custom property is added in the jcr node of the template, the same property will be added to the jcr node of page, created by using that template.

#### EDITABLE TEMPLATE

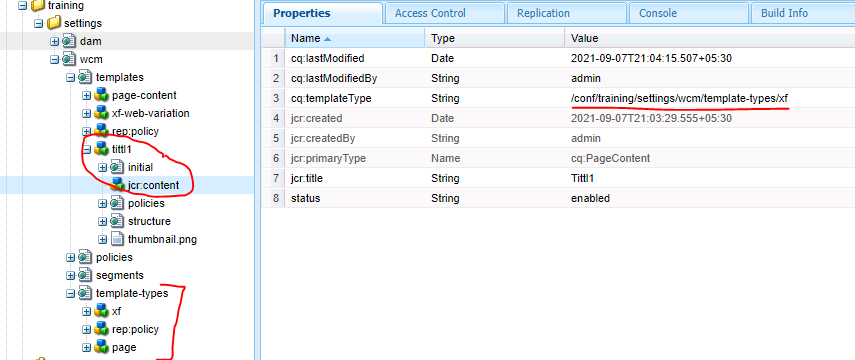
Introduced in AEM 6.2

|  |  |
| --- | --- |
| **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 |

##### CREATING EDITABLE TEMPLATES

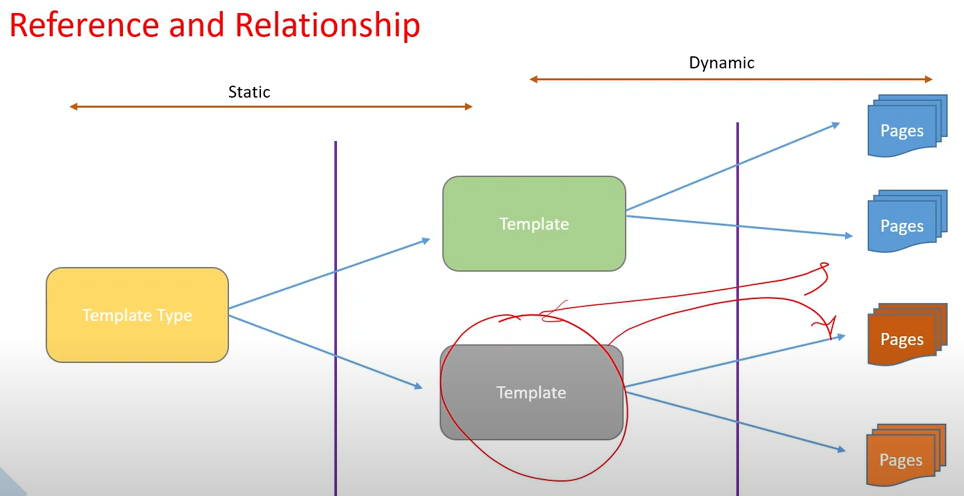


* It’s a role of template editors to create an editable table. They can define the structure of the template on it own without developer intervention
* The Editable template can be build based on some base template type. Building the template types is developer’s responsibility.
* **In the below example – “xf” and “page”are template types . “title1” is an editable template which has been on “xf” template type.**



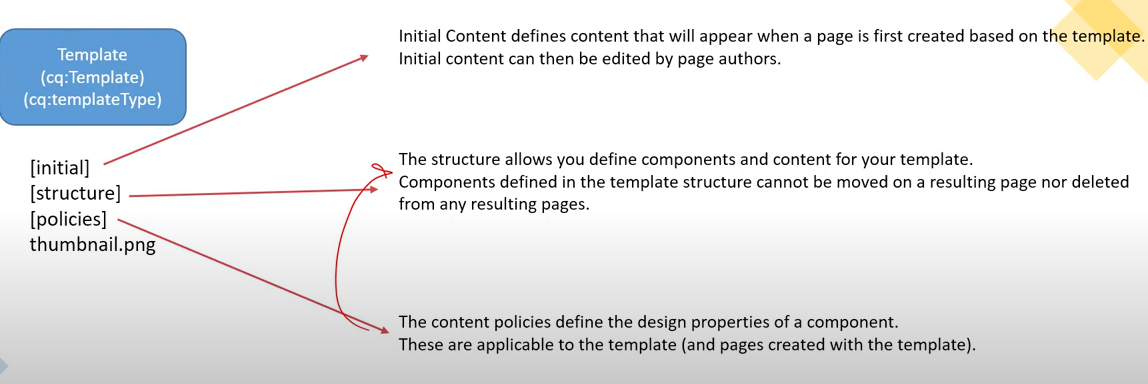
|  |  |
| --- | --- |
|  | The editable template has **cq:templateType –** which points to the template-type |

###### RELATIONSHIP BETWEEN TEMPLATE-TYPE/EDITABLE TEMPLATE AND PAGE



* Editable template can be built using template type
* **Editable template maintains the dynamic connection / reference with the page which has been build using that editable template.**
* Any change in the editable template will also update the underlining page structure .
* The relation between template-type and editable template is static. Any change in template-type will not update the undelying page.

###### EDITABLE TEMPLATE STRUCTURE



|  |  |
| --- | --- |
|  | * **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.   **Note : All AEM maven project creates this folder structure for us (like Maven Artchetye - 23)** |

###### TEMPLATE TYPE RESOLUTION

|  |  |
| --- | --- |
|  | * For building the editable template – it looks for the template-type. * AEM looks for the template type in a hierarchy in as shown. * It looks for conf 🡪 apps 🡪 libs |

###### GOTCHAS

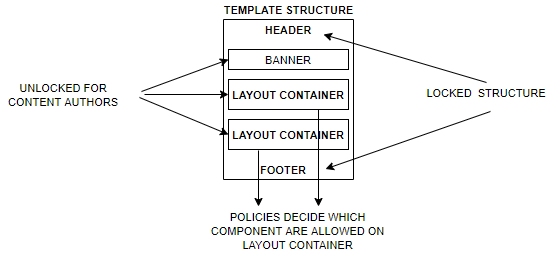
* For old projects or project created using old maven archetype – We can create the structure using below steps

|  |  |
| --- | --- |
| * 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 |

**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 annotated 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> | |

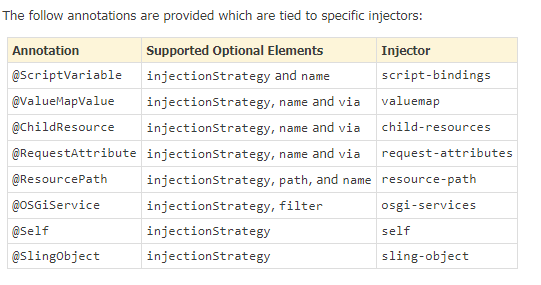
##### POM.XML

|  |  |
| --- | --- |
| 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 />** |

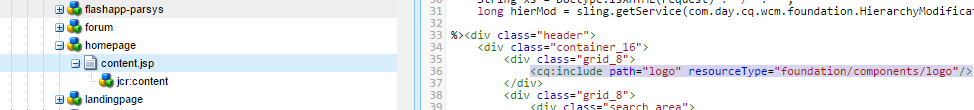
### DIALOG & DESIGN DIALOGS

#### 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. authoring.dialog** | |
|  | |

**CUSTOM VALIDATION USING GRANITE UI**

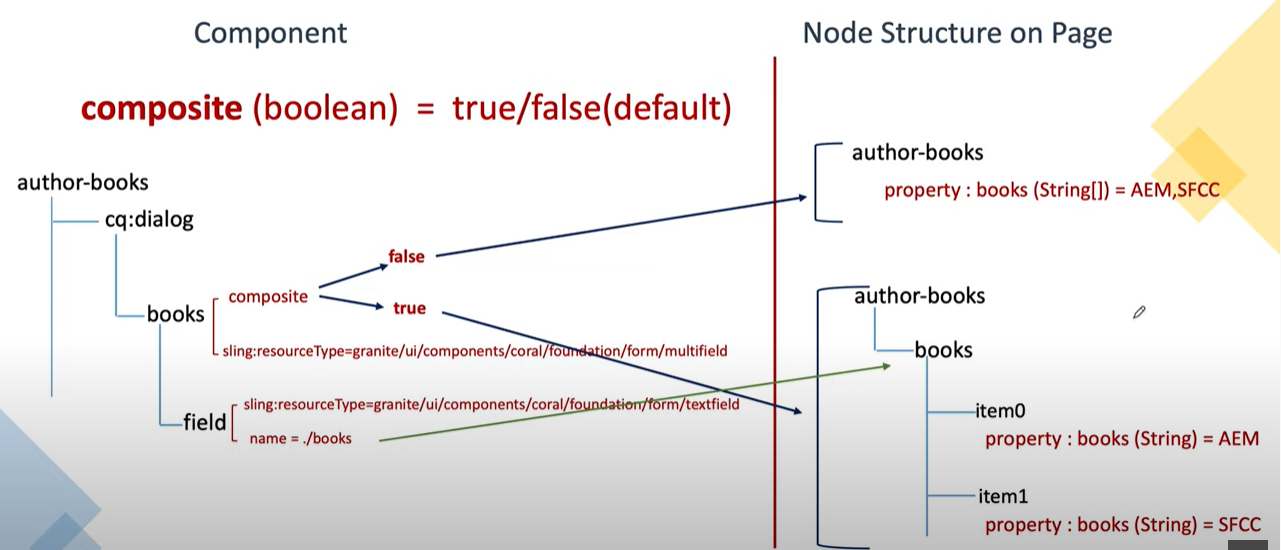
* To execute the custom validation logic in the JS files we need to load the JS file using either of the above methods.
* Previously for form validation JQuery validator is used which has been deprecated and “foundation-validation-validator”is recommended

|  |
| --- |
| **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** |

#### HANDLING MULTIFIELD USING SLING MODEL

* If composite= **false**. It will create a String [] and store all the value in the same node
* If composite= **true.** It will create child nodes for each item of multifield
* As a thumb rule – if the multifield has just one field – use composite= **false** . And use composite= **true** of multifiled has more than one field.

**MULTIFIELD WITH ONE FIELD [COMPOSITE PROPERTY = FALSE]**



**CORRESPONDING SLING MODEL**

|  |  |
| --- | --- |
| **INTERFACE** | |
| public interface EmployeeDetails {  String getEmployeeName();  List<String> getSkills(); } |  |
| **SLING MODEL** | |
| import com.aem.geeks.core.models.EmployeeDetails; import org.apache.sling.api.SlingHttpServletRequest; import org.apache.sling.models.annotations.Default; import org.apache.sling.models.annotations.DefaultInjectionStrategy; import org.apache.sling.models.annotations.Model; import org.apache.sling.models.annotations.injectorspecific.ValueMapValue;  import java.util.ArrayList; import java.util.Collections; import java.util.List;  @Model(adaptables = SlingHttpServletRequest.class, adapters = EmployeeDetails.class, defaultInjectionStrategy = DefaultInjectionStrategy.*OPTIONAL*) public class EmployeeDetailsImpl implements EmployeeDetails {   @ValueMapValue  @Default(values = "AEM Geeks")  String empName;   @ValueMapValue  List<String> skills;   @Override  public String getEmployeeName() {  return empName;  }   @Override  public List<String> getSkills() {  if (skills != null) {  return new ArrayList<>(skills);  }  return Collections.*emptyList*();  } } | |

**MULTIFIELD WITH MORE THAN ONE FIELD[COMPOSITE : TRUE]**

### 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 CLIENT 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**. |  |

##### CLIENT LIBRARY – USEFUL URLs

|  |  |
| --- | --- |
| **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. |

### 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.

### 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.

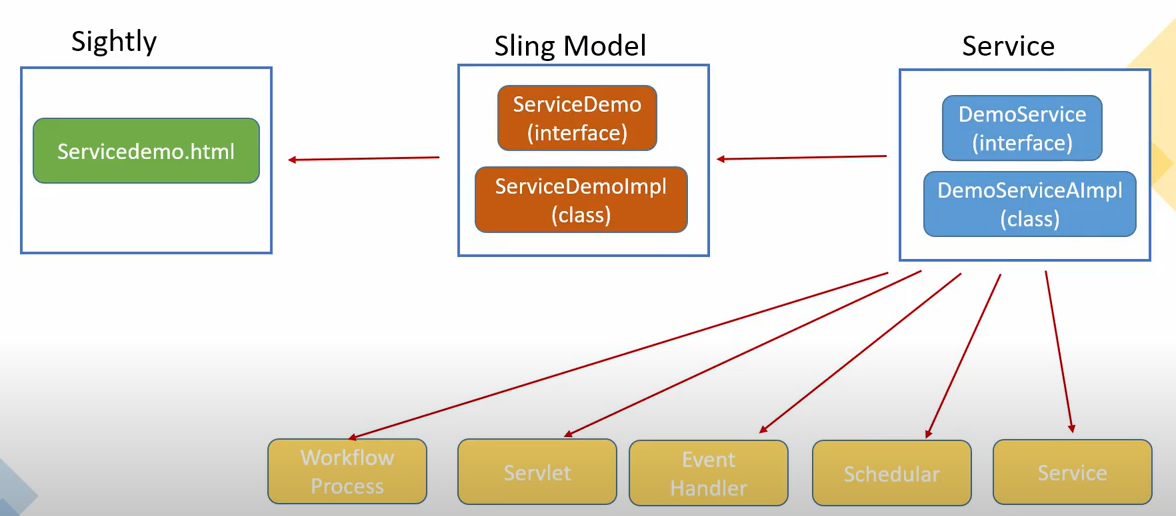
1. OSGi bundle can be installed, updated, and uninstalled without taking down the entire application.
2. OSGi bundling allows multiple versions to exist.
3. OSGi (Open Services Gateway Initiative) defines an architecture for developing and deploying modular applications and libraries.

#### OSGI SERVICES

##### 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

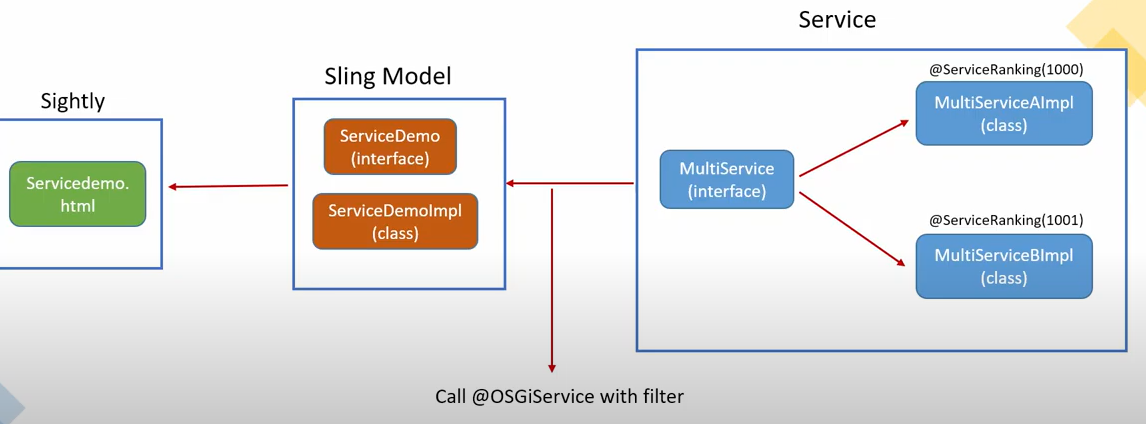
* We use **@OSGiService** annotation to call a OSGI service in a sling model

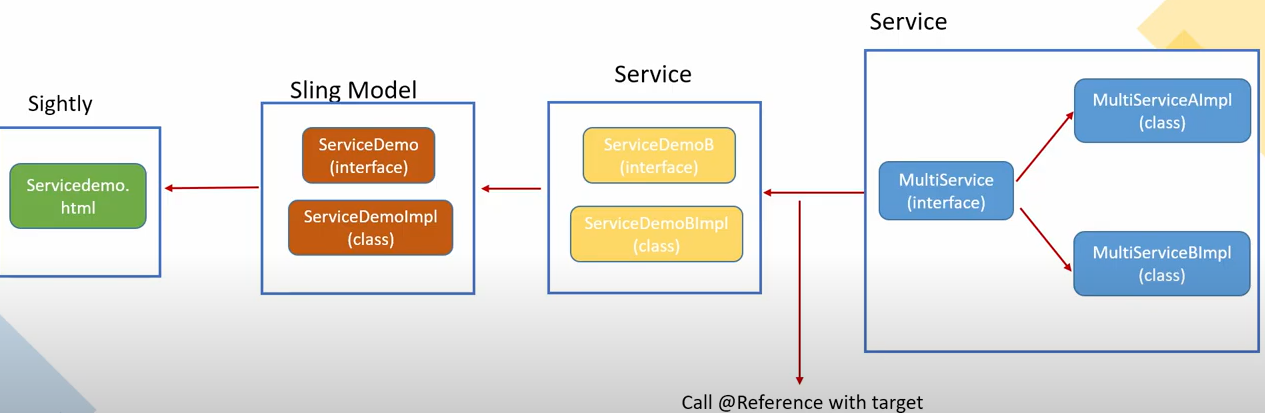
##### CALLING AN OSGI SERVICE IN ANOTHER SERVICE

* 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 using ServiceRanking.

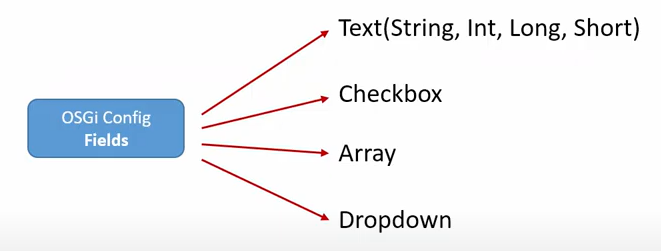




### OSGI CONFIGURATION – USING R7 ANNOTATIONS

* OSGI Configuration helps in passing the dynamic value in the Java Class like components etc.
* You can manage the configuration settings for such bundles by either:
* using the Adobe CQ Web console
* configuring content-nodes (sling:OsgiConfig) in the repository
* Either method can be used though there are subtle differences, primarily in relation to Run Modes:

#### OSGI CONFIGURATION FIELD TYPES



#### CREATING OSGI CONFIGURATION

|  |  |  |
| --- | --- | --- |
|  | Step 1: Create an Interface  Step 2: Create an Impl class which will have embeded OSGi configuration interface  Note : The OSGI interface can a separate file as well rather than embedding in a Impl Class | |
|  | | OSGI CONFIGURATION TO BE BUILT   * The OSGi Configuration values are stored in **/apps/system/config as a file** |

|  |
| --- |
| **INTERFACE**  public interface TrainingOSGiConfig {  public String getServiceName();  public int getServiceCount();  public boolean isLiveData();  public String[] getCountries();  public String getRunModes(); } |
| **IMPL CLASS (IT’S A SERVICE)**  @Component(service = TrainingOSGiConfig.class,immediate = true) @Designate(ocd = TrainingOSGiConfigImpl.ServiceConfig.class) public class TrainingOSGiConfigImpl implements TrainingOSGiConfig {  @ObjectClassDefinition(name = "Training- OSGi Configuration",description = "Training- OSGi Configuration")  public @interface ServiceConfig{  @AttributeDefinition(name = "Service Name",  description = "Name of Service",type = AttributeType.*STRING*)  public String trainingServiceName() default "Training Service";   @AttributeDefinition(name = "Service Count", description = "Service Count", type = AttributeType.*INTEGER*)  public int serviceCount() default 5;   @AttributeDefinition(name = "is Live Data",type = AttributeType.*BOOLEAN*, description = "is Live Data")  public boolean isLiveData() default false;   @AttributeDefinition(name = "Countries", type = AttributeType.*STRING*, description = "Add Country")  String[] countries() default {"de", "in"};   @AttributeDefinition(name="Run Mode", description = "Select Run Mode",options = {  @Option(label = "Author", value = "author"),  @Option(label = "Publish", value = "publish"),  @Option(label = "Both", value = "both")  })  String runMode() default "both";  }   private String trainingServiceName;  private int serviceCount;  private boolean isLiveData;  private String[] countries;  private String runMode;   @Activate  public void activate(ServiceConfig serviceConfig){  trainingServiceName = serviceConfig.trainingServiceName();  isLiveData = serviceConfig.isLiveData();  serviceCount = serviceConfig.serviceCount();  countries = serviceConfig.countries();  runMode = serviceConfig.runMode();   }   @Override  public String getServiceName() {  return trainingServiceName;  }   @Override  public int getServiceCount() {  return serviceCount;  }   @Override  public boolean isLiveData() {  return isLiveData;  }   @Override  public String[] getCountries() {  return countries;  }   @Override  public String getRunModes() {  return runMode;  } } |

#### CALLING THE OSGI CONFIGURATION

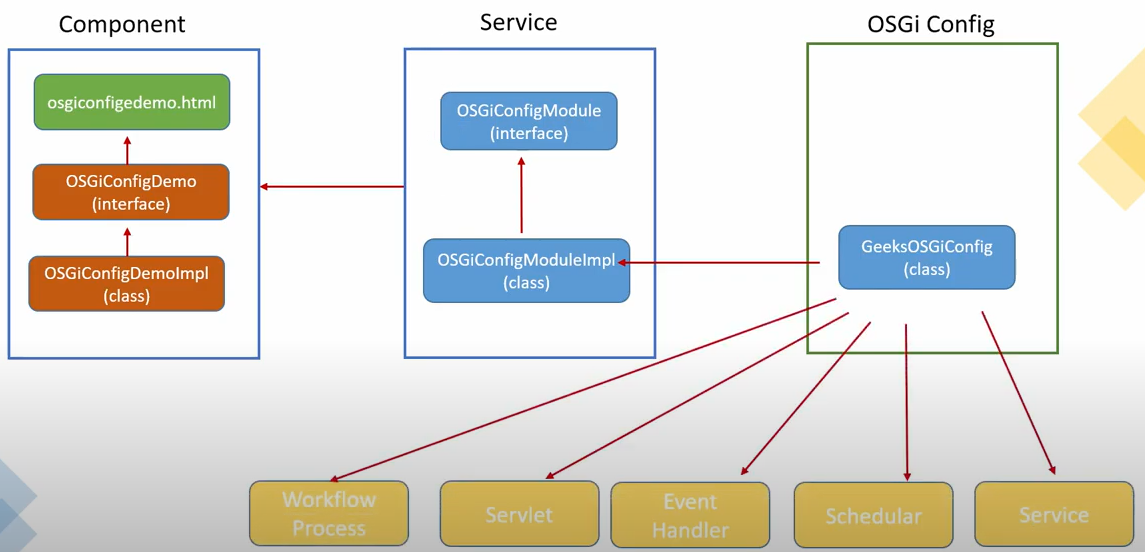
The OSgi Configuration can be called from any backend component. Like Sling Models, Schedulers,Workflow,Services,Servlet and Event Handlers

##### CALLING A OSGI CONFIGURATION IN A SLING MODEL

|  |
| --- |
| SLING MODEL  @Model(adaptables = SlingHttpServletRequest.class) public class TrainingModel {   **@OSGiService  TrainingOSGiConfig trainingOSGiConfig;**   public String getServiceName(){  return trainingOSGiConfig.getServiceName();  }    public int getServiceCount() {  return trainingOSGiConfig.getServiceCount();  }   public boolean isLiveData() {  return trainingOSGiConfig.isLiveData();  }   public String[] getCountries() {  return trainingOSGiConfig.getCountries();  }   public String getRunModes() {  return trainingOSGiConfig.getRunModes();  } } |
| SIGHLY COMPONENT  <sly **data-sly-use**.**trainingConfig**="com.training.core.models.TrainingModel"></sly> Service Name : **${**trainingConfig.serviceName**}**<br/> Service Count : **${**trainingConfig.serviceCount**}**<br/> Service RunMode : **${**trainingConfig.runModes**}**<br/> IsLivedata : **${**trainingConfig.liveData**}**<br/> <ul **data-sly-list**="**${**trainingConfig.countries**}**">  <li>**${**item**}**</li> </ul> |

#### CREATING OSGI CONFIGURATION- OSGI AS SEPARATE FILE

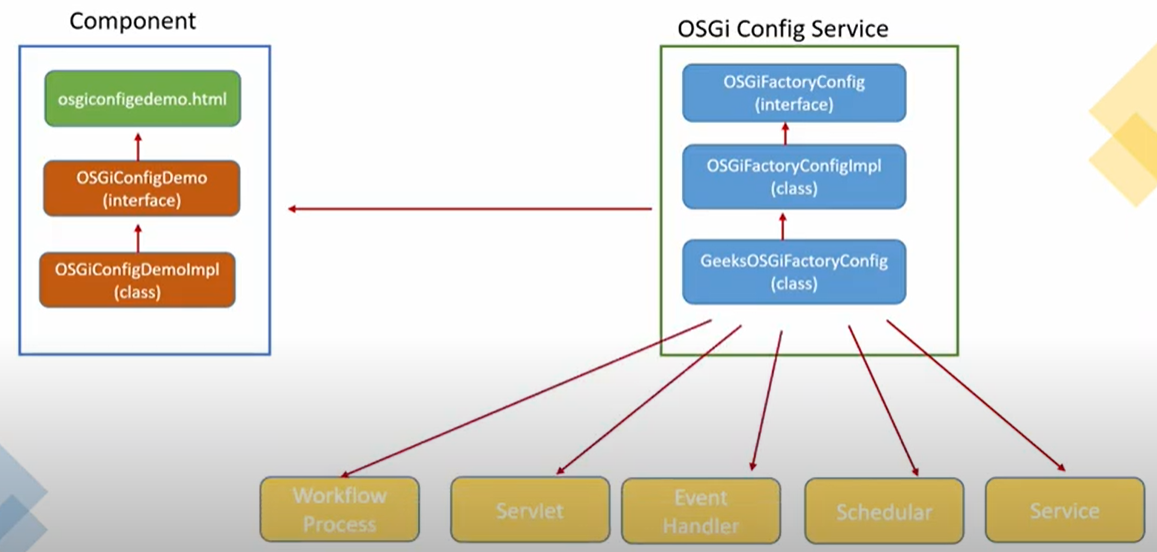
* In the above example , the OSGi Configuration is a part of a class . This defeats the purpose of reusability.
* The OSGI configuration interface can be a separate file as well.



|  |
| --- |
| INTERFACE  @ObjectClassDefinition(name = "Training - OSGI Config Module",description = "Training - OSGI Config Module") public @interface OSGiConfigModule {   @AttributeDefinition(name = "Service Name",description = "Name of Service",type = AttributeType.*STRING*)  public String serviceName();   @AttributeDefinition(name = "Service Count",description = "Count of Service",type = AttributeType.*INTEGER*)  public int serviceCount();    @AttributeDefinition(name = "Service Url",description = "URL of Service",type = AttributeType.*STRING*)  public String serviceUrl(); } |
| **USING THE OSGI CONFIGURATION IN SERVICE**  **INTERFACE**  public interface TrainingOSGiConfigModule {  public String getServiceName();  public String getServiceUrl() ;  public int getServiceCount() ; }  **IMPL**  @Component(immediate = true,name = "Training - OSGI Configuration In Interface") @Designate(ocd = OSGiConfigModule.class) public class OSGiConfigModuleImpl implements TrainingOSGiConfigModule {  private String serviceName;  private int serviceCount;  private String serviceUrl;   @Activate  public void activate(OSGiConfigModule osGiConfigModule){  serviceName = osGiConfigModule.serviceName();  serviceCount = osGiConfigModule.serviceCount();  serviceUrl= osGiConfigModule.serviceUrl();  }   public String getServiceName() {  return serviceName;  }   public String getServiceUrl() {  return serviceUrl;  }   public int getServiceCount() {  return serviceCount;  } } |
| **SLING MODEL**  @Model(adaptables = SlingHttpServletRequest.class) public class TrainingModel {   @OSGiService  TrainingOSGiConfigModule trainingOSGiConfigModule;   public String getServiceNameWithInterface(){  return trainingOSGiConfigModule.getServiceName();  }    public int getServiceCountWithInterface() {  return trainingOSGiConfigModule.getServiceCount();  }   public String getServiceUrlWithInterface() {  return trainingOSGiConfigModule.getServiceUrl();  } } |
| **HTL**  <sly **data-sly-use**.**trainingConfig**="com.training.core.models.TrainingModel"></sly> Service Name : **${**trainingConfig.serviceNameWithInterface**}**<br/> Service Count : **${**trainingConfig.serviceCountWithInterface**}**<br/> Service URL : **${**trainingConfig.serviceUrlWithInterface**}**<br/> |

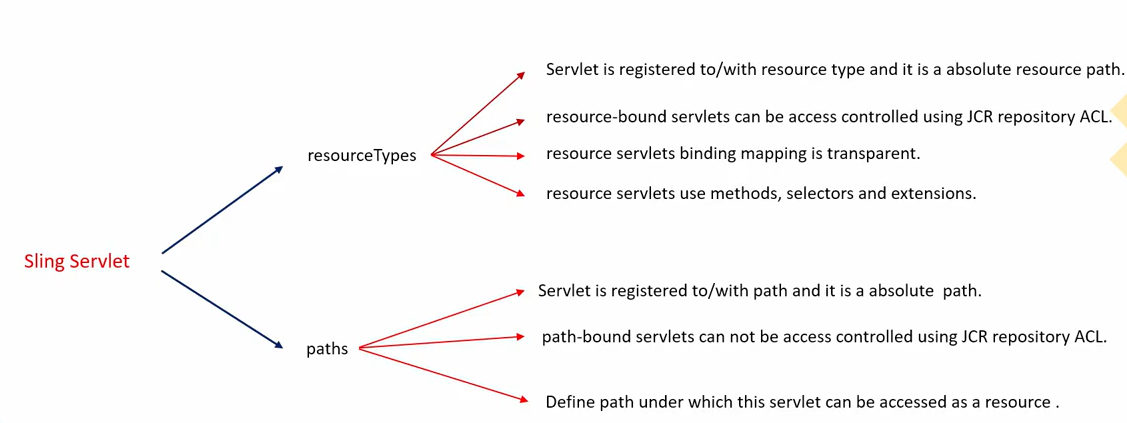
#### OSGI CONFIGURATION USING (SLING:OSGICONFIG) IN THE REPOSITORY

#### OSGI FACTORY CONFIGURATION



### SLING SERVLET

#### REGISTERING A SERVLET



**Note :**

* **Sling Servlets can be registered by both resourceType and path .In this case - Path will take the precedence . All other properties on Sling servlets are ignored.**

#### WRITING A SERVLET

Servlet can be be written in following ways

1. **OSGi DECLARATIVE SERVICE-1.4(R7) -This uses bnd-maven-plugin 4.0+**
2. **OSGi DECLARATIVE SERVICE- 1.2(R5) -This uses maven-plugin-plugin 3.0+**
3. **FELIX SCR – This uses maven-scr-plugin – (This is outdated)**

The plugin version can be validated in in parent pom.xml

#### EXTENDING A CLASS IN A SERVLET

* **SlingSafeMethodsServlet** - Helper base class for read-only Servlets used in Sling. It supports **GET, HEAD, OPTIONS** etc methods.
* **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.

#### CREATING A SERVLET USING OSGi DS 1.4 (R7)

##### REGISTERING USING RESOURCE-TYPE

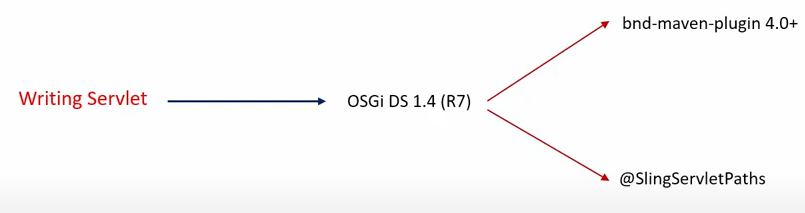
* The annotation used to mark a class as Sling Servlet - @**SlingServletResourceTypes**
* The servlet is bound to a particular resource type or resourcetype in the hierarchy

|  |
| --- |
| @Component(service = {Servlet.class}) @SlingServletResourceTypes(  resourceTypes = "training/components/page",  methods = {HttpConstants.*METHOD\_GET*,  HttpConstants.*METHOD\_POST*}, selectors ={"sample",”sample1”} ) @ServiceDescription("Home Page Servlet") public class HomePageServlet extends SlingAllMethodsServlet {  @Override  protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException {  Resource resource = request.getResource();  String title = resource.getValueMap().get(JcrConstants.*JCR\_TITLE*, StringUtils.*EMPTY*);  System.*out*.println("Title is ="+title);  response.getWriter().println(title);  }   @Override  protected void doPost(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException {  super.doPost(request, response);  } }  } }  **This servlet will be triggered for all the resources whose resourcetype is : training/components/page** |

|  |  |
| --- | --- |
| Since the servlet is mapped to a resourceType. To call the Servlet: <http://localhost:4502/content/training/us/en.sample.html> | |
| * The servlet can be called for home page we well. To call a servlet : <http://localhost:4502/content/training/us/en/homepage.sample.html>   Although the homepage has different resourceType- but servlet is getting called for homepage resourcetype as well . Why?  The Sling Servlet will eb not just get called for the resource type it is registed for , but also for those resource-type is inheriting it. | |
|  | Home page component is child of **training/components/page (resourceSuperType)** |

##### REGISTERING SERVELT WITH PATH

* The servlet can be bounded to multiple paths



|  |
| --- |
| @Component(service = Servlet.class) @SlingServletPaths(value = {  "/bin/training",  "/bin/training1" }) @ServiceDescription("Path Based Servlet") public class TrainingServlet extends SlingSafeMethodsServlet {   @Override  protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException {  JSONObject pageTitleJson =null;  JSONArray pageArr = new JSONArray();  final ResourceResolver resourceResolver = request.getResourceResolver();  Page page = resourceResolver.adaptTo(PageManager.class).getPage("/content/training/us/en");  Iterator<Page> children = page.listChildren();  while (children.hasNext()){  Page childPage = children.next();  try {  pageTitleJson = new JSONObject();  pageTitleJson.put(childPage.getName(),childPage.getTitle());  pageArr.put(pageTitleJson);  } catch (JSONException e) {  e.printStackTrace();  }  }  response.getWriter().write(String.*valueOf*(pageArr));  } } |

|  |  |
| --- | --- |
|  | * The root paths which are opened as allowed servlet path are configured in OSGi Configuration * We can add any custom path to the Execution Path multifield * Note : Add a “/” at the end to the configured path value – so that it can cater the child path too |

#### CREATING A SERVLET USING OSGi DS 1.2 (R5)

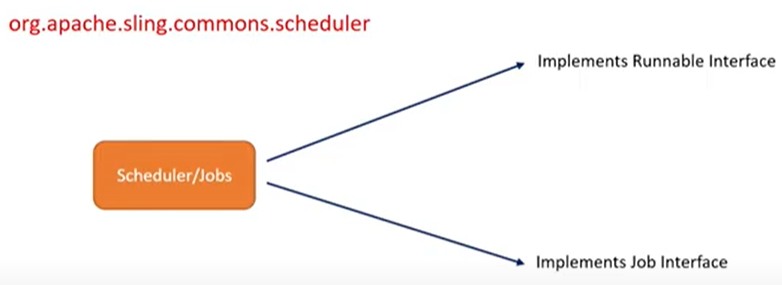
* Unlike DS1.4 – It does not have specific annotation defined – exclusively for servlet.
* All the Servelt are marked with @Component annotation.

<https://sling.apache.org/apidocs/sling9/org/apache/sling/api/servlets/ServletResolverConstants.html>

|  |
| --- |
| import org.apache.sling.api.servlets.ServletResolverConstants;  @Component(service = Servlet.class,  property = {  ServletResolverConstants.*SLING\_SERVLET\_METHODS* +"="+ HttpConstants.*METHOD\_GET*,  ServletResolverConstants.*SLING\_SERVLET\_RESOURCE\_TYPES* +"="+ "training/components/page",  ServletResolverConstants.*SLING\_SERVLET\_SELECTORS* +"="+"ds",  ServletResolverConstants.*SLING\_SERVLET\_EXTENSIONS* +"="+"xml"  }) public class TrainingServletDS extends SlingSafeMethodsServlet {   @Override  protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws ServletException, IOException {  Resource resource = request.getResource();  String title = resource.getValueMap().get(JcrConstants.*JCR\_TITLE*, StringUtils.*EMPTY*);  System.*out*.println("Title is ="+title);  response.getWriter().println(title);  } } |

### SCHEDULERS

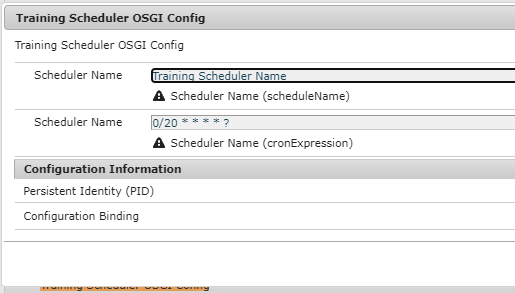
* **Scheduler executes time or cron based Jobs.**



#### IMPLEMENTING SCHEDULER USING RUNNABLE INTERFACE

* In the below example – **We are configuring the cron Expression and Scheduler name in OSGI Configuration**
* The logic which the scheduler has to run will go inside the “run()” method.

OSGI CONFIGURATIONS



|  |
| --- |
| **OSGI CONFIGURATION**  @ObjectClassDefinition(name = "Training Scheduler OSGI Config",  description = "Training Scheduler OSGI Config") public @interface TrainingSchedulerOSGiConfig {  @AttributeDefinition(name = "Scheduler Name",description = "Scheduler Name",  type = AttributeType.*STRING*)  public String scheduleName() default "Training Scheduler Name";   @AttributeDefinition(name = "Scheduler Name",description = "Scheduler Name",  type = AttributeType.*STRING*)  public String cronExpression() default "0/20 \* \* \* \* ?"; *// runs every 10 seconds* } |
| **SCHEDULER**  @Component(immediate = true,service = Runnable.class) @Designate(ocd = TrainingSchedulerOSGiConfig.class) public class TrainingScheduler implements Runnable{   @Reference  Scheduler scheduler;   private int schedulerId;   @Activate  public void activate(TrainingSchedulerOSGiConfig trainingSchedulerOSGiConfig){  schedulerId = trainingSchedulerOSGiConfig.scheduleName().hashCode();  ScheduleOptions scheduleOptions = scheduler.EXPR(trainingSchedulerOSGiConfig.cronExpression());  scheduleOptions.name(String.*valueOf*(schedulerId));  scheduleOptions.canRunConcurrently(true);  scheduler.schedule(this,scheduleOptions);  }  @Deactivate  public void deActivate(TrainingSchedulerOSGiConfig trainingSchedulerOSGiConfig){  scheduler.unschedule(String.*valueOf*(schedulerId));  }  @Override  public void run() {  System.*out*.println("This is where Logic Reside which scheduler will execute");  } } |

#### IMPLEMENTING SCHEDULER USING JOB INTERFACE

* The class must implement the Job interface and implement the execute() method
* The execute() contains the logic which the scheduler will execute.
* ***Job interface give a flexibility to execute multiple jobs***

|  |
| --- |
| package com.training.core.schedulers;  import com.training.core.config.TrainingSchedulerOSGiConfig; import org.apache.sling.commons.scheduler.ScheduleOptions; import org.apache.sling.commons.scheduler.Scheduler; 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.Reference; import org.osgi.service.metatype.annotations.Designate;  @Component(immediate = true,service = Runnable.class) @Designate(ocd = TrainingSchedulerOSGiConfig.class) public class TrainingScheduler implements Runnable{   @Reference  Scheduler scheduler;   private int schedulerId;   @Activate  public void activate(TrainingSchedulerOSGiConfig trainingSchedulerOSGiConfig){  schedulerId = trainingSchedulerOSGiConfig.scheduleName().hashCode();  ScheduleOptions scheduleOptions = scheduler.EXPR(trainingSchedulerOSGiConfig.cronExpression());  scheduleOptions.name(String.*valueOf*(schedulerId));  scheduleOptions.canRunConcurrently(true);  scheduler.schedule(this,scheduleOptions);  }  @Deactivate  public void deActivate(TrainingSchedulerOSGiConfig trainingSchedulerOSGiConfig){  scheduler.unschedule(String.*valueOf*(schedulerId));  }  @Override  public void run() {  System.*out*.println("Run method- This is where Logic Reside");  } } |

#### EXECUTING MULTIPLE JOBS IN SCHEDULER

|  |  |
| --- | --- |
|  | * Using Job interface, multiple jobs can be schedules. * Ideally – For such scenarios we keep the configuration like the cron expressions in a OSGi factory configurations |

|  |
| --- |
| @Component(immediate = true,service = Job.class) @Designate(ocd = TrainingSchedulerOSGiConfig.class) public class TrainingMultipleJobsScheduler implements Job {   @Reference  private Scheduler scheduler;   private int scheduleId;   @Activate  protected void activate(TrainingSchedulerOSGiConfig trainingSchedulerOSGiConfig){  scheduleId = trainingSchedulerOSGiConfig.scheduleName().hashCode();  registerSchedulerJob(trainingSchedulerOSGiConfig);  }   @Deactivate  protected void deActivate(TrainingSchedulerOSGiConfig trainingSchedulerOSGiConfig){  scheduler.unschedule(String.*valueOf*(scheduleId));  }   private void registerSchedulerJob(TrainingSchedulerOSGiConfig trainingSchedulerOSGiConfig){  */\* Job 1 Start\*/* ScheduleOptions in = scheduler.EXPR(trainingSchedulerOSGiConfig.cronExpression());  Map<String, Serializable> inMap = new HashMap<>();  inMap.put("country","in");  inMap.put("url","www.in.com");  in.config(inMap);  scheduler.schedule(this,in);  */\* Job 1 End\*/   /\* Job 2 Start\*/* ScheduleOptions de = scheduler.EXPR(trainingSchedulerOSGiConfig.cronExpression());  Map<String, Serializable> deMap = new HashMap<>();  deMap.put("country","de");  deMap.put("url","www.de.com");  de.config(deMap);  scheduler.schedule(this,de);  */\* Job 2 End\*/   /\* Job 3 Start\*/* ScheduleOptions es = scheduler.EXPR(trainingSchedulerOSGiConfig.cronExpression());  Map<String, Serializable> esMap = new HashMap<>();  esMap.put("country","es");  esMap.put("url","www.es.com");  es.config(esMap);  scheduler.schedule(this,es);  */\* Job 3 End\*/* }   @Override  public void execute(JobContext jobContext) {  Map<String, Serializable> configuration = jobContext.getConfiguration();  String county = configuration.get("country").toString();  switch (county){  case "in":  System.*out*.println("Hitting the URL>>>"+ configuration.get("url"));  break;  case "de":  System.*out*.println("Hitting the URL>>>"+ configuration.get("url"));  break;  case "es":  System.*out*.println("Hitting the URL>>>"+ configuration.get("url"));  break;  }   } } |

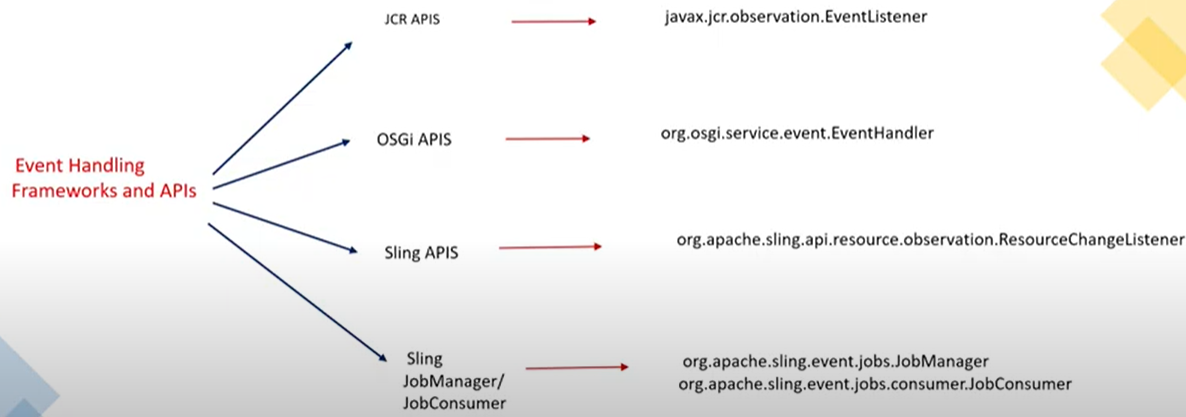
**WHAT IS APACHE JACKRABBIT?**

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 |

### EVENT HANDLING

* Event handling in AEM can characterized in two different categories . This category is based on type of event these API is listening to.
  + **JCR BASED EVENT HANDLING :** 
    - These api are used when we want to listen to the operation performed on nodes.
    - Event like - Updating , Deleting or creating nodes and properties.
  + **RESOURCE BASES EVENT HANDLING :** These handle the event of Jcr as well operation on resources like publishing the page.Types of
    - **OSGI API based Event Handling**
    - **Sling Based Event Handling**
    - These API are used to ls



* In JCR , OSGI and Sling API – The execution of event handler is not guaranteed .For example – Let say – if the event is triggered and AEM instance went down – then there is no guaranteed of completion of event handler .
* In the JobManager /JobManager API event handler will execute atleast once. Even if the AEM go down during the execution of handler – The handler might resume or atleast give the status of the handler.

#### EVENT HANDLING USING JCR API

* Event handling at JCR / Repository level
* Event handler can be registered via Observation Manager Object.
* Event listeners are notifiled asynchronously. For example if we have multiple event listener for an event. if the event occurs , the event listener will we notified in a asych way.

|  |  |
| --- | --- |
|  | Using JCR API - We can write an event handler to respond to the following JCR events:   * **A node was added, moved , deleted** * **A property was added to a node,property was changed or deleted** |

##### STEPS TO CREATE JCR API BASED EVENT HANDLER (OBSERVATION MANAGER)

1. Step 1 : Implement a EventListener Interface.( EventListener)
2. Step 2: Implement the event Handler(onEvent)
3. Step 3: Register the event handler for an event.( addEventListener)

|  |
| --- |
| @Component(service = EventListener.class,immediate = true) public class JCREventHandler implements EventListener {  private Session session = null;  @Reference  SlingRepository slingRepository;   @Activate  protected void activate() throws RepositoryException {  //String[] nodetypes={"cq:PageContent"};  session = slingRepository.loginService("training-user",null);  session.getWorkspace().getObservationManager().addEventListener(this,  Event.*NODE\_ADDED*| Event.*PROPERTY\_ADDED*,  "/content/training",true,null, nodetypes,true);  }  @Override  public void onEvent(EventIterator eventIterator) {  while(eventIterator.hasNext()){  try {  System.*out*.println("The updated path= "+eventIterator.nextEvent().getPath());  } catch (RepositoryException e) {  e.printStackTrace();  }  }  } }  }  } |

###### REGISTERING EVENT HANDLER

[**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)**

|  |  |  |
| --- | --- | --- |
| **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 |

#### EVENT HANDLING USING OSGI API

**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>



### (CAAS) - SLING MODEL EXPORTER

* The default selector of content exporter is “**model**”

#### JACKSON EXPORTER

* **JACKSON ANNOTATION**: <https://github.com/FasterXML/jackson-annotations/wiki/Jackson-Annotations>

##### SAMPLE JSON EXPORTER

|  |
| --- |
| import com.aem.geeks.core.services.ChildPagesService; import com.day.cq.wcm.api.Page; import com.fasterxml.jackson.annotation.JsonIgnore; import com.fasterxml.jackson.annotation.JsonProperty; import com.fasterxml.jackson.annotation.JsonRootName; import org.apache.sling.api.resource.Resource; import org.apache.sling.models.annotations.\*; import org.apache.sling.models.annotations.injectorspecific.OSGiService; import org.apache.sling.models.annotations.injectorspecific.Self; import com.aem.geeks.core.models.Employee;  import javax.inject.Inject; import java.util.Iterator;  @Model(adaptables = Resource.class,  adapters = Employee.class,  resourceType = "aemgeeks/components/content/employee",  defaultInjectionStrategy = DefaultInjectionStrategy.*OPTIONAL*) @Exporter(name = "jackson", extensions = "json", selector = "geeks", options = {  @ExporterOption(name = "SerializationFeature.WRAP\_ROOT\_VALUE", value = "true"),  @ExporterOption(name = "MapperFeature.SORT\_PROPERTIES\_ALPHABETICALLY", value = "true"),  }) @JsonRootName(value = "employeeData") public class EmployeeImpl implements Employee {  @Inject  @Default(values = "AEM")  String fname;   @Inject  @Required  @Default(values = "Geeks")  String lname;   @Inject  boolean permanent;   @OSGiService  ChildPagesService childPagesService;   @Self  Resource resource;   @Override  public String getFirstName() {  return fname;  }   @Override  public String getLastName() {  return lname;  }   @Override  public boolean getIsPermanent() {  return permanent;  }   @JsonProperty(value = "company")  public String companyName() {  return "COMPANY INC";  }    @JsonIgnore  public String getAddress() {  return "202 Eden Park";  }    @Override  @JsonProperty(value = "pageList")  public Iterator<Page> getPages() {  return childPagesService.getPages();  } } |

|  |  |
| --- | --- |
|  | * Jackson is the OOTB Exporter in AEM * On the class level – we have to declare the “**resource**” path – which we want to expose as JSON. * The getters in Sling Model are considered for the JSON * The key of the resulting JSON depends upon the name of the getters. For example for the getFirstName() , the key will be “firstName” |

##### JACKSON ANNOTATION

|  |  |
| --- | --- |
| **@Exporter** | * This Annotation is to define the exporter type * **name**= Name of the exporter. * **extension** = Extension to be used while accessing the content as JSON * **selector** = Selector to be used while accessing the content as JSON   **Ex:** <http://localhost:4502/content/aemgeeks/us/en/jcr:content/root/container/container/employee.geeks.json> |
| **@ExporterOption** | The defines the format of output JSON  **ADD A ROOT NODE OF RESULTING JSON**  **@ExporterOption(name = "SerializationFeature.WRAP\_ROOT\_VALUE", value = "true")**  **@JsonRootName(value = "employeeData")**  **SORT THE RESUTING JSON**  @ExporterOption(name = "MapperFeature.SORT\_PROPERTIES\_ALPHABETICALLY", value = "true"), |
| **@ JsonProperty** | * Changing the of the JSON key * The non- getter methods can be added to JSON using this annotation. |
| **@JsonIgnore** | Add this annotation on the getter- if we want to ignore the getter in the JSON response. |
| **@ JsonRootName** | Class level annotation – For the root level JSON key |

##### EXPORTER OPTIONS API

* <https://fasterxml.github.io/jackson-databind/javadoc/2.7/com/fasterxml/jackson/databind/MapperFeature.html>
* <https://fasterxml.github.io/jackson-databind/javadoc/2.6/com/fasterxml/jackson/databind/SerializationFeature.html>

#### XML EXPORTER

### HTML TEMPLATE LANGUAGE (HTL) – SIGHTLY

|  |  |
| --- | --- |
| * Sightly is the new AEM templating system * 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. |

#### GLOBAL/IMPLICT OBJECTS IN SIGHTLY

|  |  |
| --- | --- |
| properties | Component Properties |
| pageProperties | Page Properties |
| inheritedPageProperties | Inherited page properties from parent page. |
| component | The meta information of components like  Component Title:**${*component***.title**}** Component Group:**${*component***.componentGroup**}** Component Description:**${*component***.description**}** |
| componentcontext |  |
| resource | Object of current resource |
| resourcePage | Page path of the page where the resource has been dropped |
| currentDesign |  |
| currentNode | Information about current node like – node path etc. |
| currentPage | Information about current page like – page path etc. |
| resourceDesign |  |
| wcmmode | WCM Mode. It has properties which return Boolean. For example,   * ${wcmmode.edit} : “true” for edit mode * ${wcmmode.preview} : “true” for preview mode |

* Properties, pageProperties and inheritedPageProperties are Enumerable object gives a Valuemap of properties
* All other objects are called Java Backed Object.

#### SLY ELEMENT

|  |  |
| --- | --- |
| **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 |
| **EXAMPLE**  <div **data-sly-use**.**employeeDetails**="com.aem.geeks.core.models.EmployeeDetails"></div>  This will create an empty div in the Markup which is of no use . Hence in such cases it is recommended to use “sly” element  <sly **data-sly-use**.**employeeDetails**="com.aem.geeks.core.models.EmployeeDetails"></sly> | |

#### BLOCK ELEMENT

HTML Template Language (HTL) block statements are custom data attributes added directly to existing HTML.

|  |  |  |
| --- | --- | --- |
| **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 |  |

#### ITERATIONS

##### DATA-SLY-LIST

* data-sly-test is used to perform iteration. This involves the iteration of
  + Arrays
  + List
    - List of Strings
    - List of Map
    - List of bean objects
  + Map

ITERATION SYNTAX

|  |  |
| --- | --- |
| **EXPLICT ITEM VARIABLE**  <ul **data-sly-list**="**${**employeeDetails.skills**}**"> <li>**${**itemList.count**}** - **${**item**}**</li>  </ul>  **USING CUSTOM VARIABLE**  <ul  **data-sly-list.empSkills**="**${**employeeDetails.skills**}**">  <li>**${empSkills**List.count**} - ${ empSkills }**</li> </ul> | * Sightly explicitly gives a variable “item” – which has element of each iteration. * We can use a custom variable too. * ${itemList.count} – It gives the current count of iteration. |
| **ITERATING A MAP** <ul **data-sly-list**.**cmap**="**${**employeeDetails.customMap**}**">  <li>key=**${**cmap**}** | value= **${**employeeDetails.customMap[cmap]**}**</li> </ul> | |
| **ITERATING A LIST OF BEAN**  <ul **data-sly-list**.**users**="**${**employeeDetails.userList**}**">  <li>**${**users.userId**}** | **${**users.userName**}**</li> </ul> | |
| **ITERATING A LIST OF MAP** <ul **data-sly-list**.**cListMap**="**${**employeeDetails.customListOfMap**}**">  <sly **data-sly-list**="**${**cListMap**}**">  <li>**${**item**}** |**${**cListMap[item]**}** </li>  </sly> </ul> | |
| **ITERATING ITERATOR**  **SLING MODEL**  @Model(adaptables = SlingHttpServletRequest.class, adapters = EmployeeDetails.class, defaultInjectionStrategy = DefaultInjectionStrategy.OPTIONAL)  public class EmployeeDetailsImpl implements EmployeeDetails {  @ScriptVariable  PageManager pageManager;  @Override  public Iterator<Page> getPages() {  Page page = pageManager.getPage("/content/aemgeeks/us/en");  Iterator<Page> child = page.listChildren();  return child;  }  **}**  **SIGHTLY**  <div **data-sly-list**="**${**employeeDetails.pages**}**">  <p>**${'The PageName is {0} has title {1}' @***format*=[item.name, item.name]**}**</p> </div> | |
| * **${itemList.index}**: Index of iteration * **${itemList.count}**: Count of iteration * **${itemList.first}**: Returns “true” if first element of iteration else false * **${itemList.last}**: Returns “false” if first element of iteration else false * **${itemList.even}**: Returns “true” if even element of iteration else false * **${itemList.odd}**: Returns “true” if odd element of iteration else false | |

###### LIMITING LIST

|  |
| --- |
| <ul **data-sly-list**.**skills**="**${**employeeDetails.skills **@** begin = 0, end =10, step =2**}**">  <li>**${**skills**}**</li> </ul>   * It’s a zero index based iteration. It will start from 0th index and end at index= 1. * Step =2 : The iteration jumbs 2 steps in iteration. |

##### DATA-SLY-REPEAT

|  |
| --- |
| <div **data-sly-repeat**="**${**employeeDetails.skills**}**">  <p>**${**item**}**</p> </div>   * In this block element – The container element(div) also repeats not just <p> |

#### CODE REUSABILITY

##### DATA-SLY-TEMPLATE

|  |  |
| --- | --- |
|  | * Create a template (HTML file) file where the reusable template will reside * **data-sly-use:** Takes the path of the HTML file where the template is residing * **data-sly-call**: We can pass parameter to the template |

**EXAMPLE**

|  |
| --- |
| **TEMPLATE FILE**  <div **data-sly-template**.**employeeDetailsTemplate**="**${@** employeeParam**}**">  <ul **data-sly-list**.**cmap**="**${**employeeParam.customMap**}**">  <li>key=**${**cmap**}** | value= **${**employeeParam.customMap[cmap]**}**</li>  </ul> </div>  <div **data-sly-template**.**mapAttribute**="**${@** mapParam**}**">  <ul **data-sly-list**="**${**mapParam**}**">  <li>map-key=**${**item**}** | map-value= **${**mapParam[item]**}**</li>  </ul> </div> |
| **CALLING THE TEMPLATE: The template can be called from any HTML file**  <div **data-sly-use**.**employeeDetails**="com.aem.geeks.core.models.EmployeeDetails"></div>  <div **data-sly-use**.**empDetailsTemplate**="sightlyTemplate.html">  <sly **data-sly-call**="**${**empDetailsTemplate.employeeDetailsTemplate **@***employeeParam*=employeeDetails**}**"></sly>  <sly **data-sly-call**="**${**empDetailsTemplate.mapAttribute **@***mapParam*=employeeDetails.customMap**}**"></sly> </div>   * customMap is a map in EmployeeDetails sling model |

#### CONDITIONS

##### DATA-SLY-TEST

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

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

#### INCLUDING RESOURCES

##### DATA-SLY-INCLUDE

|  |  |
| --- | --- |
| **TO INCLUDE HTML** | <div data-sly-include="../showTextComponent/includeFile.html"/> |
| **TO INCLUDE JSP** | <div data-sly-include="../showTextComponent/includeFile.jsp"/> |
| **INCLUDE USING FILE ATTRIBUTE** | <div data-sly-include="${ @file = 'include.html'}"></div> |
| **INCLUDING FILE IN A RUN MODE** | <section data-sly-include="${'template.html' @ wcmmode='disabled'}"></section> |
| **APPENDING IN FILE PATH** | <section data-sly-include="${'template.html' @ prependPath='my/path'}"></section>  ***File Path: template/my/path.html*** |
| **PREPEND IN FILE PATH** | <section data-sly-include="${'my/path' @ appendPath='template.html'}"></section>  ***File Path: my/path /template.html*** |

* NOTE: This replaces the hosting element with the HTML/markup of included file. In above example the “div” element will be replaced by the HTML/markup of included file.

##### DATA-SLY-RESOURCE

|  |  |
| --- | --- |
| TO INCLUDE RESOURCE | <div data-sly-resource="${'content' @resourceType='wcm/foundation/components/parsys'}"></div> |
| INCLUDE CUSTOM RESOURCE | <div **data-sly-resource**="**${'/content/aemgeeks/us/en/author/jcr:content/parsys-8/author'}**'"></div> |
| INCLUDE CUSTOM RESOURCE USING SELECTOR | <div **data-sly-resource**="**${'/content/aemgeeks/us/en/author/jcr:content/parsys-8/author' @***addSelectors*=**'test'}**'"></div> |

#### DATA-SLY-SET

* data-sly-set defines a new identifier with a pre-defined value.

EXAMPLE

|  |
| --- |
| **INTERFACE**  public interface StudentModel {  Student getStudentInfo(); } |
| **SLING MODEL**  @Model(adapters = StudentModel.class,  adaptables = Resource.class,  defaultInjectionStrategy = DefaultInjectionStrategy.*OPTIONAL*) public class StudentModelImpl implements StudentModel {  @Inject  @Default(values = "No First name")  String sfname;  @Inject  @Default(values = "No Last name")  String slname;   @Override  public Student getStudentInfo() {  System.*out*.println(sfname +slname);  Student student = new Student(sfname,slname);  return student;  } } |
| **HTL**  <div **data-sly-use**.**student**="com.aem.geeks.core.models.StudentModel"></div>  Student Name: **${**student.studentInfo.firstName**}**,**${**student.studentInfo.lastName**}** <br/> <sly data-sly-set.studentData="**${**student.studentInfo**}**"></sly>  Student Name: **${**studentData.firstName**}**,**${**studentData.lastName**}** |
| * In the HTL – we are fetching the values into 2 different ways. In the firt way to fetch each property we need to call the “getStudentInfo()” method. * To make the fetch more optimal – We fetch the value once and set it to an identifier (studentData) using data-sly-set. The further fetching of properties happen from the identifier rather than calling the method again and again |

#### DATA-SLY-USE

|  |  |  |
| --- | --- | --- |
|  | **htlAppComponent.html**  <div data-sly-use.nav='/apps/htlApp/components/navigation.js'>  ${nav.foo}</div> | **navigation.js**  use(function () {  return {  foo: "Hello World"  };  }); |
| **INCLUDING CLIENTLIBS**  STEP 1 : <sly data-sly-use.**clientLibs**="${'/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**, | To remove the <div></div> elements.  <div data-sly-use.nav="navigation.js" data-sly-unwrap>${nav.foo}</div>  Output 🡪Hello World |

#### 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> |

#### DATA-SLY-ATTRIBUTE

**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> |

* **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.**
* If an 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> |

##### DYNAMIC ATTRIBUTES

* data-sly-attibute comes out to be very handly when an attribute on HTML or sightly elements are dynamic.
* A map can be dymanically generated to set the attributes. For example,

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

#### SIGHTLY RELATIONAL OPERATOR

##### SEARCHING USING “IN”

* Search using “in” is case sensitive

|  |  |
| --- | --- |
| **SEARCHING STRING** | **${'a'** in **'abc'}** |
| **SEARCHING IN LIST** | **${'sds'** in employeeDetails.skills**}** |
| **SEARCHING IN MAP** | **${'key3'** in employeeDetails.customMap**}**  **It search of the occourence of key in a map** |

##### FORMATING

###### FORMATTING USING format

|  |
| --- |
| <div **data-sly-use**.**employeeDetails**="com.aem.geeks.core.models.EmployeeDetails"></div>  <div **data-sly-list**="**${**employeeDetails.customMap**}**">  <p>**${'The Employee id {0} has Employee Name {1}' @***format*=[item,employeeDetails.customMap[item]]**}**</p> </div> |

###### FORMATTING USING join

* The arrays, list or map can be formatted as string separated by dilimiter

<div>${employeeDetails.skills @join=','}</div>

#### SIGHTLY COMMENTS

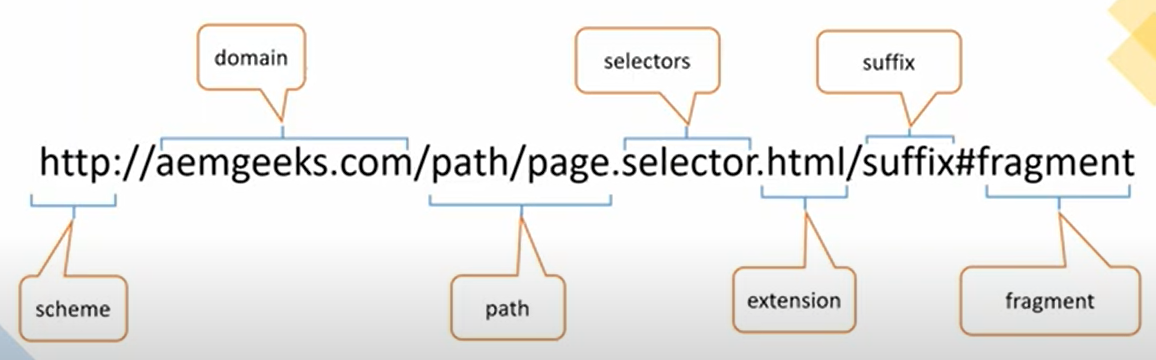
Following example shows on line 1 an HTL comment, and on line 2 an HTML comment:

<!--/\* An HTL Comment \*/-->

<!-- An HTML Comment -->

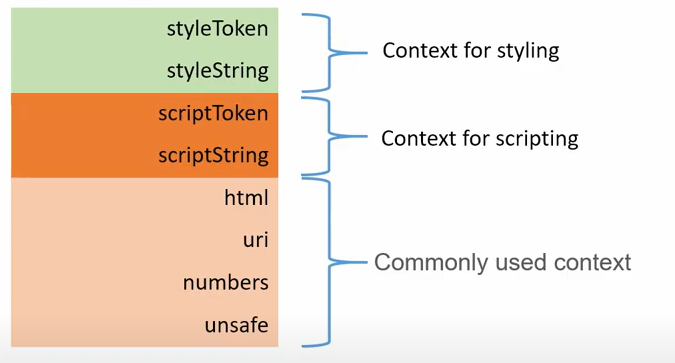
* The whole HTL comment, and anything within will be entirely ignored by the processor and removed from the markup.
* The content of standard HTML comments however will be passed through and expressions within the comment will be evaluated.
* HTML comments cannot contain HTL comments and vice versa.

#### URL MANIPULATION USING SIGHTLY



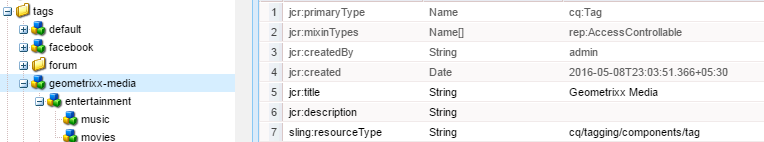
#### HTL /SIGHTLY CONTEXT

* Contexts are used to prevent cross site scripting.



### 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.

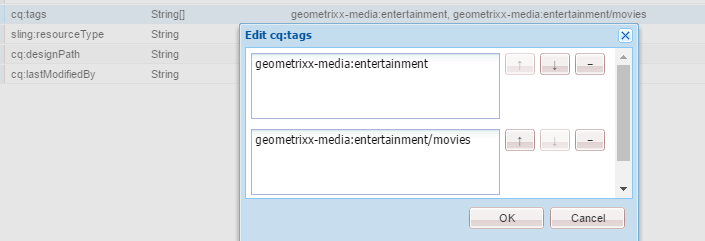


**TAG CHARACTERISTICS**

* 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.**

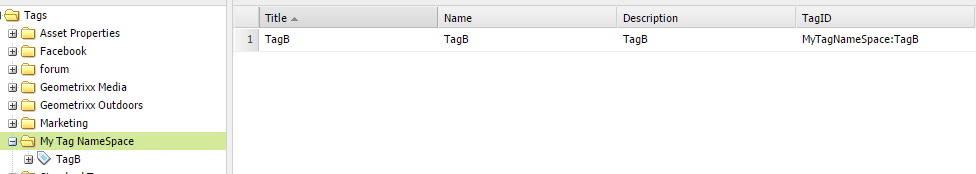
**CONTAINER TAGS**

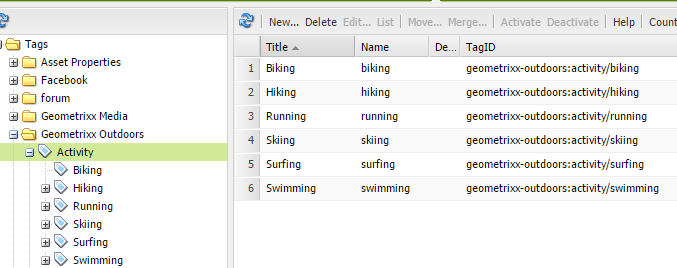
A container tag is a node of type cq:Tag containing any number and type of child nodes, which makes it possible to enhance the tag model with custom metadata.

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:

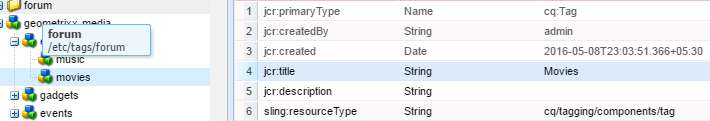
**RangeIterator<Resource> it = tagManager.find(tagID);**

**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

* Workflows enable us 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.

#### WORKFLOW STEPS

A workflow is made of steps. The workflow steps are of 4 types

|  |  |
| --- | --- |
| **CONTAINER STEP** | We use this step – when we want to trigger a child work flow. |
| **PARTICIPANT STEP** | We use this step – We we need user involvement – like approvals |
| **DECISION STEP** |  |
| **PROCESS STEP** | Used in Custom workflow. An ECMA script or Java class implements the step |

#### TRIGGERING A WORKFLOW

Workflow can be triggred by following ways

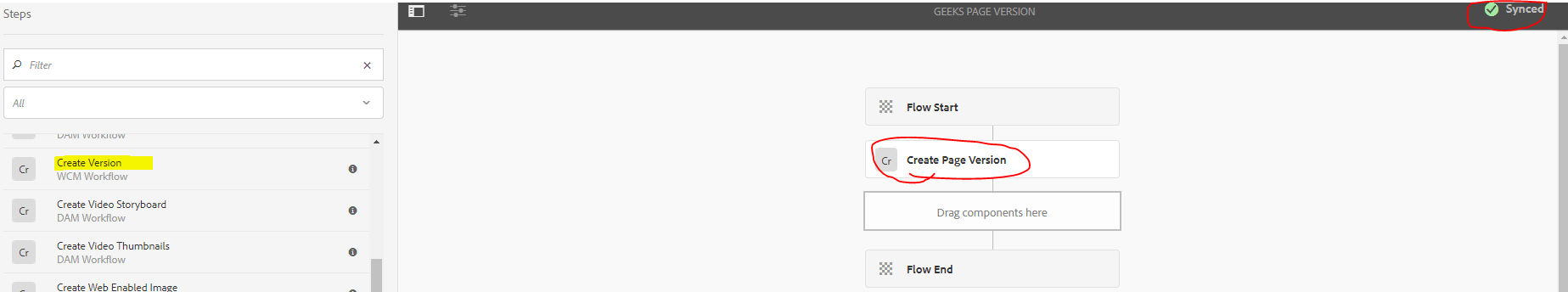
1. Using Launcher from workflow launcher console
2. API/Code – Workflow can be triggred using Servlet , Scheduler or Slimng Models
3. Manually from workflow comsole

#### WORKFLOW MODEL

* All workflow models are stored at (workflow Design)**: /conf/global/settings/workflow/models**
* A runtime model also gets created for the same workflow model at : **/var/workflow/models**

##### CREATING A WORKFLOW MODEL

* Go to Tools 🡪Workflow 🡪Models
* Create workflow with a “Title” and “Name”
* Drag and drop of the step. Note: Everytime we update the model we need to sync the workflow design with the runtime model

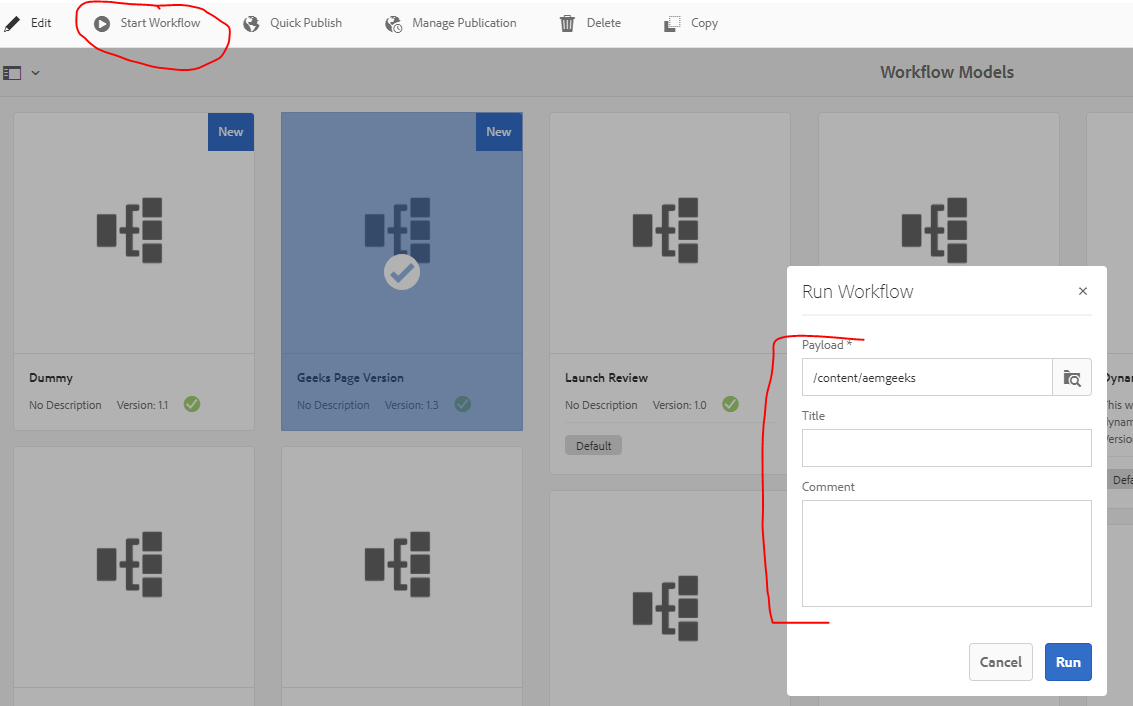


|  |  |
| --- | --- |
| **WORKFLOW TABS** | |
| **MODELS** | Lists the workflow models currently available. Here you can start, create, edit or delete workflow models. |
| **INSTANCES** | Details of workflow instances which are currently active. These instances are also version dependent |
| **ARCHIVE** | 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. |

#### TRIGGERING A WORKFLOW

##### MANUALLY TRIGGERING WORKFLOW

* To manually trigger the workflow – select the workflow with a manually entered payload(path of resource)

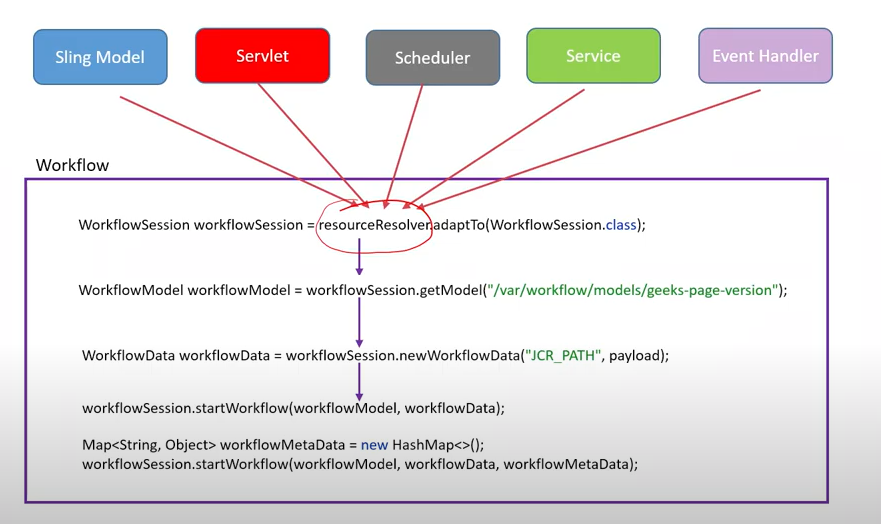


##### TRIGGERING WORKFLOW USING LAUNCHER

|  |  |
| --- | --- |
|  | **Go to Tools 🡪 Launcher 🡪 Add Launcher**  **LAUINCHER PROPERTIES**   * Event Type: Event when workflow will be triggerd * Nodetype: : Type of node for which workflow will be triggered * Path : The path of the payload * Workflow : Select the Work flow Model.   Make sure Workflow is in activated |

#### TRIGGERING A WORKFLOW USING API/CODE

* The workflow can be called called from any of the backend module



##### TRIGGERING THE WORKFLOW FROM SERVLET

|  |
| --- |
| @Component(service = Servlet.class) @SlingServletPaths(  value = {"/bin/executeworkflow","/geeks/executeworkflow"} ) public class ExecuteWorkflow extends SlingSafeMethodsServlet {  private static final Logger *LOG* = LoggerFactory.*getLogger*(ExecuteWorkflow.class);  @Override  protected void doGet(final SlingHttpServletRequest req, final SlingHttpServletResponse resp) throws ServletException, IOException {  String status="Workflow Executing";  final ResourceResolver resourceResolver = req.getResourceResolver();  String payload=req.getRequestParameter("page").getString();  try {  if(StringUtils.*isNotBlank*(payload)) {  WorkflowSession workflowSession = resourceResolver.adaptTo(WorkflowSession.class);  WorkflowModel workflowModel = workflowSession.getModel("/var/workflow/models/geeks-page-version");  WorkflowData workflowData = workflowSession.newWorkflowData("JCR\_PATH", payload);  status=workflowSession.startWorkflow(workflowModel, workflowData).getState();  }  } catch (Exception e) {  *LOG*.info("\n ERROR IN WORKFLOW {} ", e.getMessage());  }  resp.setContentType("application/json");  resp.getWriter().write(status);  } } |

|  |  |
| --- | --- |
| **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 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");

}

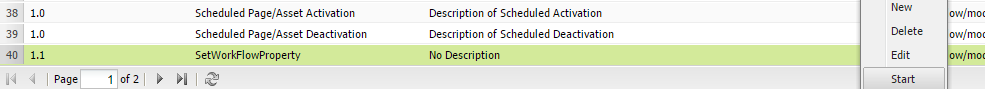
}

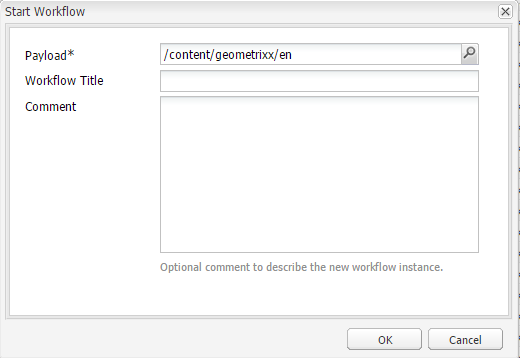
To execute of this work flow by passing the arguments:

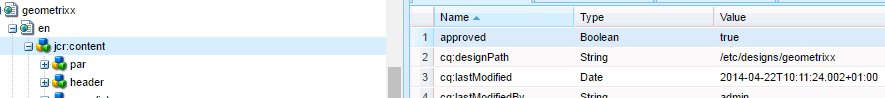
Step 1: Create a New Workflow Model

|  |  |
| --- | --- |
| 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. |
|  |  |

### VLT (VAULT) TOOL

* Tool use to pull and push the data from CRX to file system and vice versa

#### SETING UP 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>***

##### EXPORTING FILES FROM CRX [CRX 🡪 FILE STSTEM]

**To Export(CRX🡪File System) : vlt --credentials admin:admin export -v http://localhost:4506/crx /apps/training**

**vlt co http://localhost:4502/crx/-/jcr:root/apps/company company**

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

**Backup created on file Sytem**

##### IMPORTING FILES FROM CRX [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"?>*

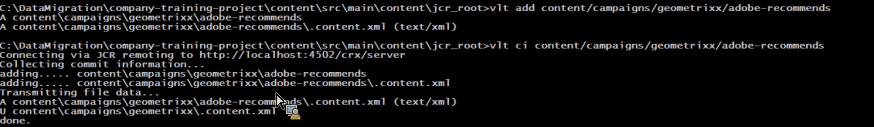
*<workspaceFilter version="1.0">*

***<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** | |
|  | |

### 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**

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

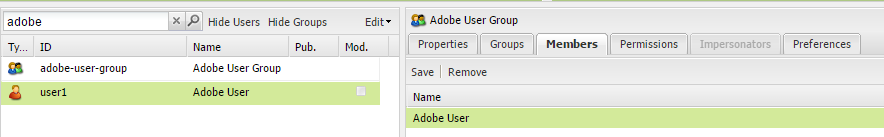
**Assigning permission to the user group**

|  |  |
| --- | --- |
|  | **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**

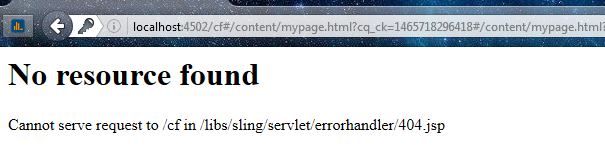
Let’s create 2 user groups

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

### 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>

### 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

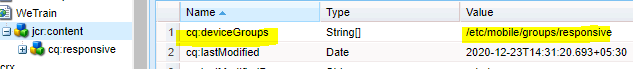
**::\* default JVM options**

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> )

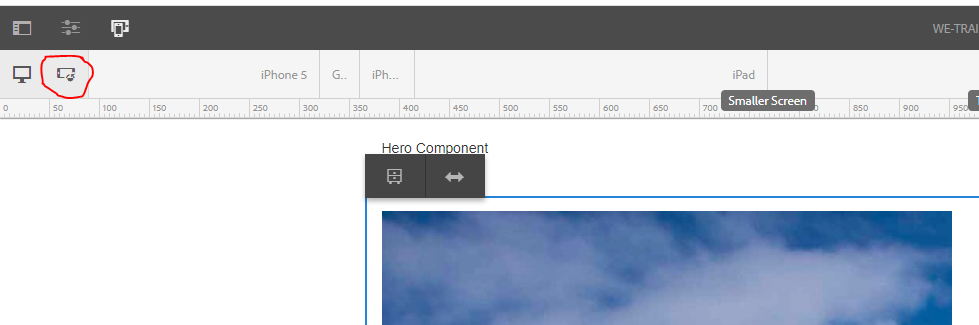
### ENABLING MOBILE EMULATOR

|  |  |
| --- | --- |
| **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



### 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

#### WRITING A CUSTOM LOGS

* We usually use any of the Logger library for logging like SLFJ4

|  |
| --- |
| @Model(adaptables = SlingHttpServletRequest.class)  public class TrainingModel {  **public static final Logger logger = LoggerFactory.getLogger(TrainingModel.class);**    @OSGiService  TrainingOSGiConfig trainingOSGiConfig;  public String getServiceName(){  logger.info("Service Name:"+trainingOSGiConfig.getServiceName());  return trainingOSGiConfig.getServiceName();  }  } |

#### CONFIGURING PROJECT SPECIFIC CUSTOM LOGS

* Configuration Name : **Apache Sling Logger Configuration**
* ***It’s a OSGi Factory Configuration***



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**

1. **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.

1. **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 |

1. **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 |

1. **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.

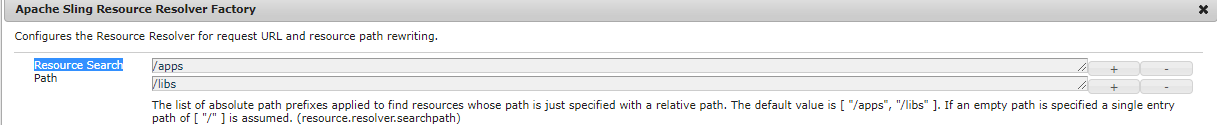
But as adobe is promoting to use its new touch optimized ui for granite components , it have changed the definition of overlay for Touch ui (Granite) components where you need to create only similar skeleton structure (where nodes can be of type nt:unstructured) and you can add, remove or modify existing 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.



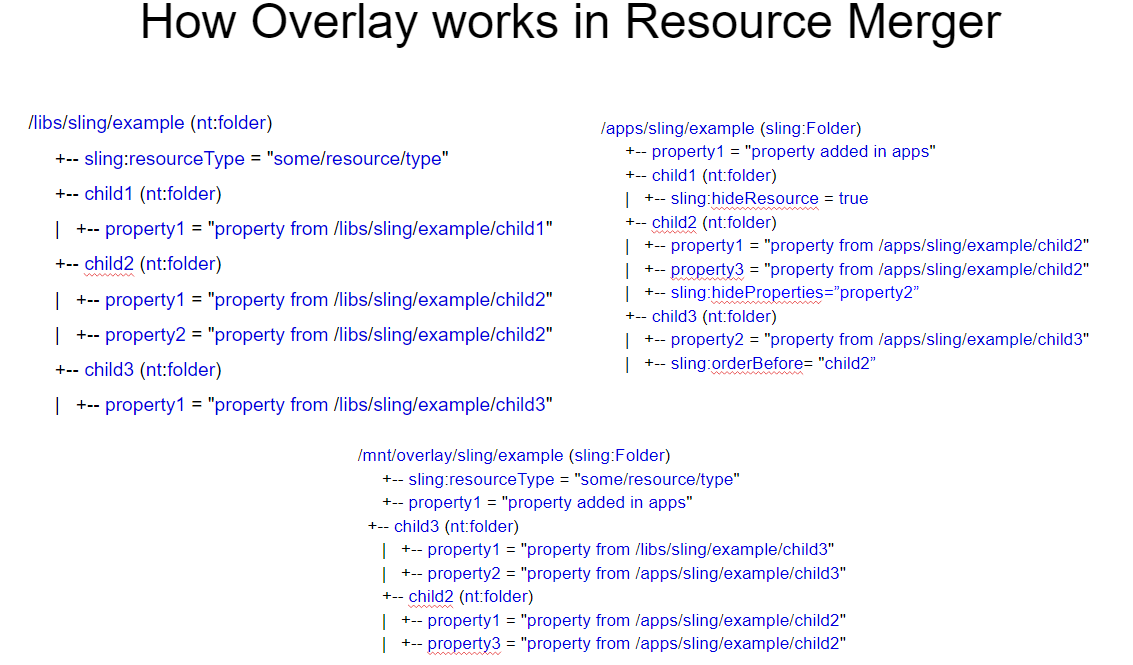
**WITH AEM 6.0 ONWARDS, AFTER INTRODUCTION OF TOUCH UI FOR GRANITE RELATED OVERLAY WE ARE USING SLING RESOURCE MERGER**

* 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.

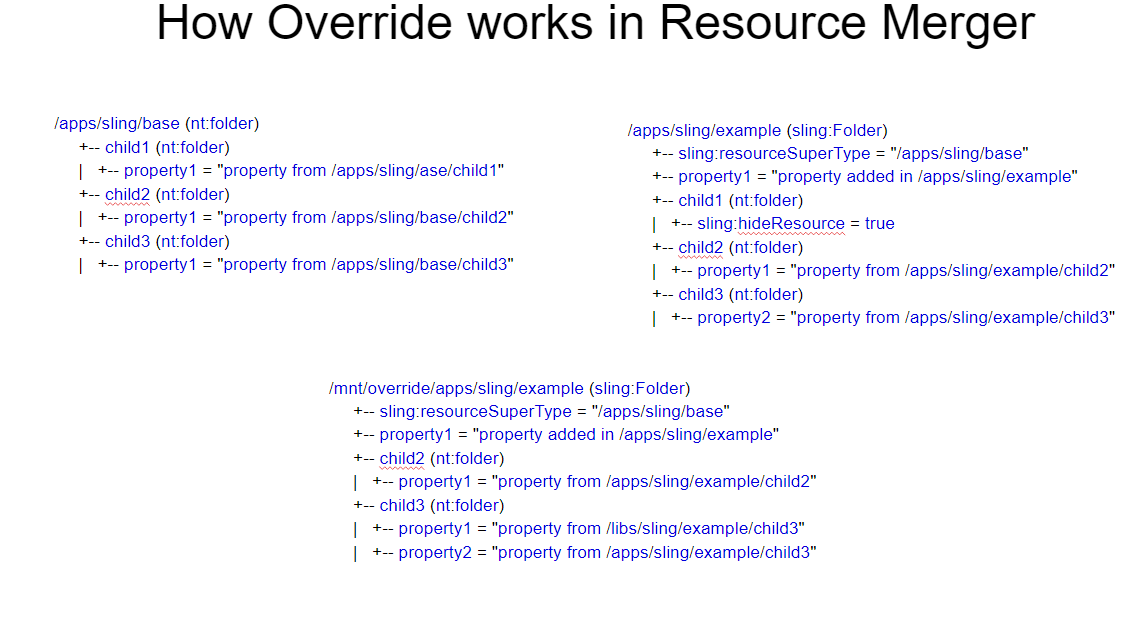
|  |  |  |  |
| --- | --- | --- | --- |
| **EXAMPLE**  Lets say , we want to modify navigation option(say project) in the navigation by overlaying using Sling resource merger  The navigation is generated from **/libs/cq/core/content/nav.** Let overlay this node for our requirement | |  | |
|  |  | | |
| 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.

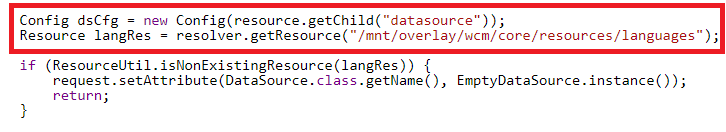
If I check the language.jsp under /libs/cq/gui/components/common/datasources/languages, you can see that all the languages are coming from /libs/wcm/core/resources/languages.

|  |
| --- |
| 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.

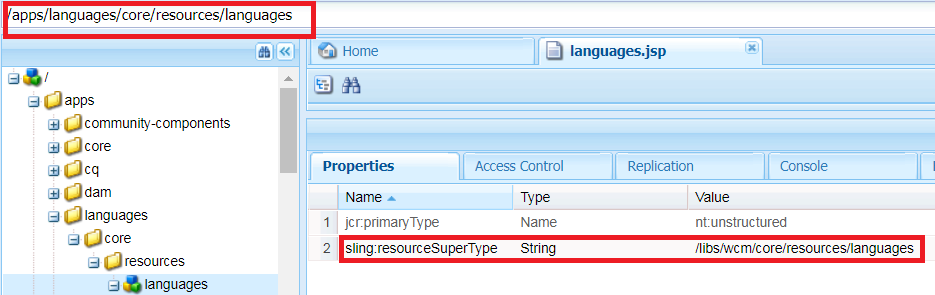


Fig - Overriding the languages which are available under libs

Overlay the language.jsp with the following modifications.

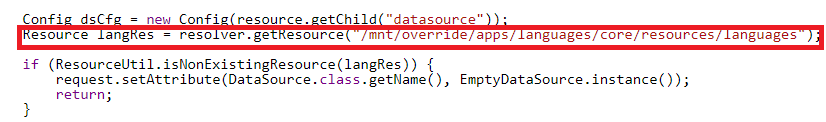
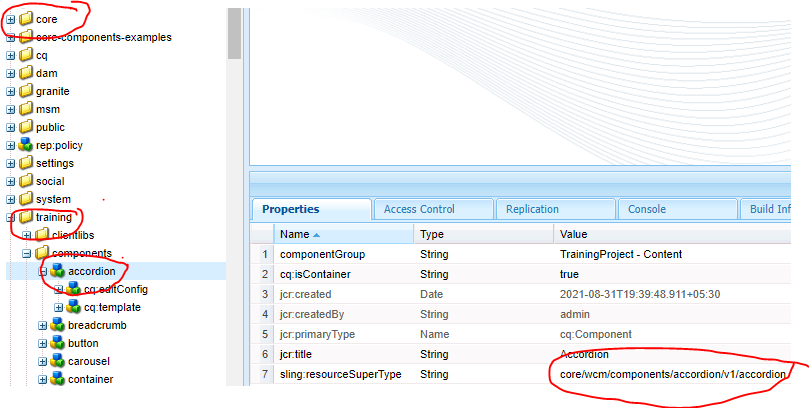


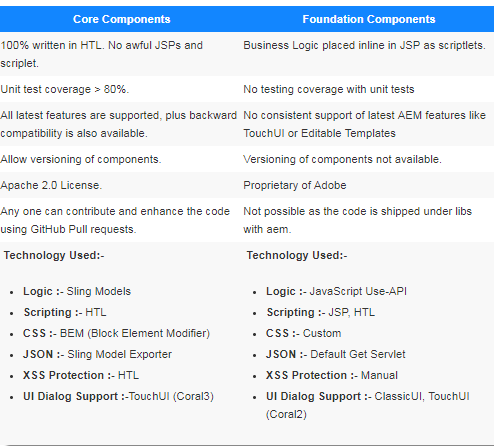
Fig - Merging the new locale and existing locale using mnt override

### CORE COMPONENTS

|  |  |
| --- | --- |
| * AEM core components are introduced in aem 6.2 but are strongly recommended to use from aem 6.3 * Core Components a versioned * Note: The new maven archetype has core components are part of maven dependencies. It creates proxy component on the project level itself – which then can be extended by the custom components. | wcm-components-hierarchy |



#### ADVANTAGE OF CORE COMPONENTS

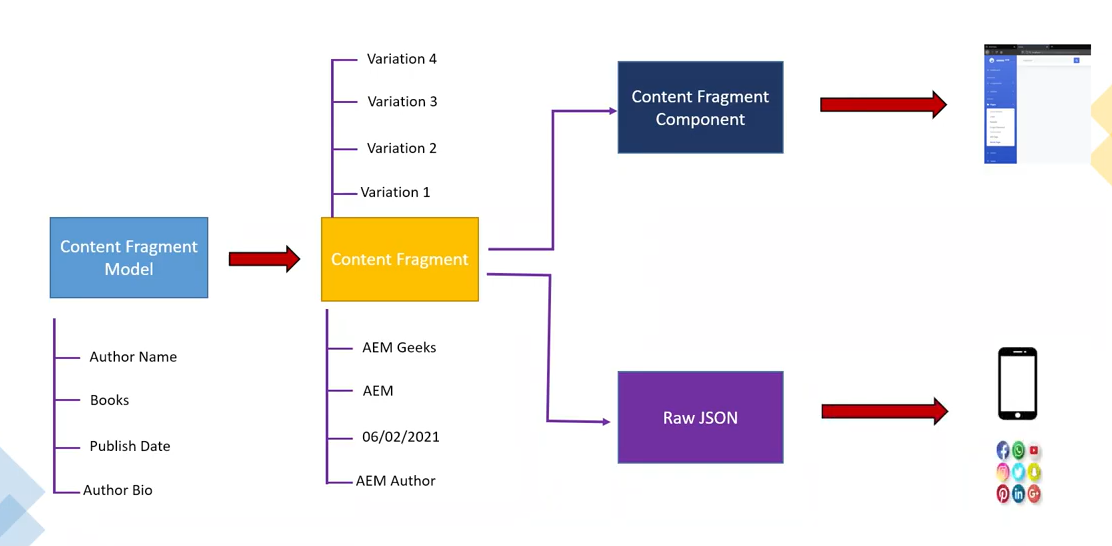


#### PROXY COMPONENT

|  |  |
| --- | --- |
| * **To use core components, we need to create proxy components in aem.** * 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 thatwe can switch to any version by updating the sling:resourceSuperType. | core-component-binding-structure |

### 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.



* 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 MODEL

1. Introduce in AEM 6.4
2. Using Content Fragment Model, we can define a desired structure of the content.
3. **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 |

###### 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.   **NOTE : To use drop down – Use “Enumeration” datatype. The options of drop down can be provided using comma seperated values.** | |

###### CRX LOCATION OF CONTENT FRAGMENT MODELS

|  |  |
| --- | --- |
| * The Content fragments models are saved in conf folder in the project specific folder . For example – for training project the content fragment will be stored in: **/conf/training/settings/dam/cfm/models/trainingcfmodel** * These content fragment model act as template for content fragments |  |

#### USING THE CONTENT FRAGMENT MODEL TO CREATE CONTENT FRAGMENT

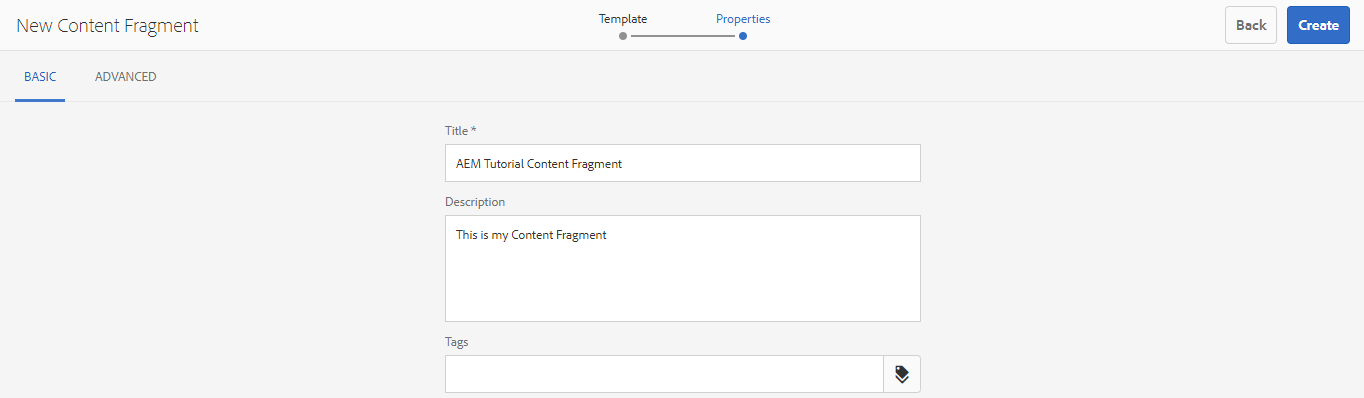
* The content fragment models act as templates – while creating the content fragments.
* The content fragments stored as an asset in CRX.

##### STEPS TO CREATE CONTENT FRAGMENT

* Go to Assets 🡪Files
* Create a folder to store all the content fragment.

|  |
| --- |
| **GOTCHAS**   * If the content fragment folder is not visible inside the newly created folder – Select the folder 🡪 Properties **🡪** Cloud Services. * Make sure the conf/ <projectFolder> is added to the cloud configuration. This is the location where the CFM are stored in CRX. |

* **Create a “language folder” - to enable translation.**
* Select the content fragment model to create a content fragment



###### CONTENT FRAGMENT CREATED (MASTER COPY)

|  |  |
| --- | --- |
|  |  |
| **MASTER**  Enter the content in the RTE of Master 🡪Save | |

###### CREATING VARIATIONS

|  |  |
| --- | --- |
|  | 1. Click of Create Variation 2. Update the content (as per requirement) in the variation |

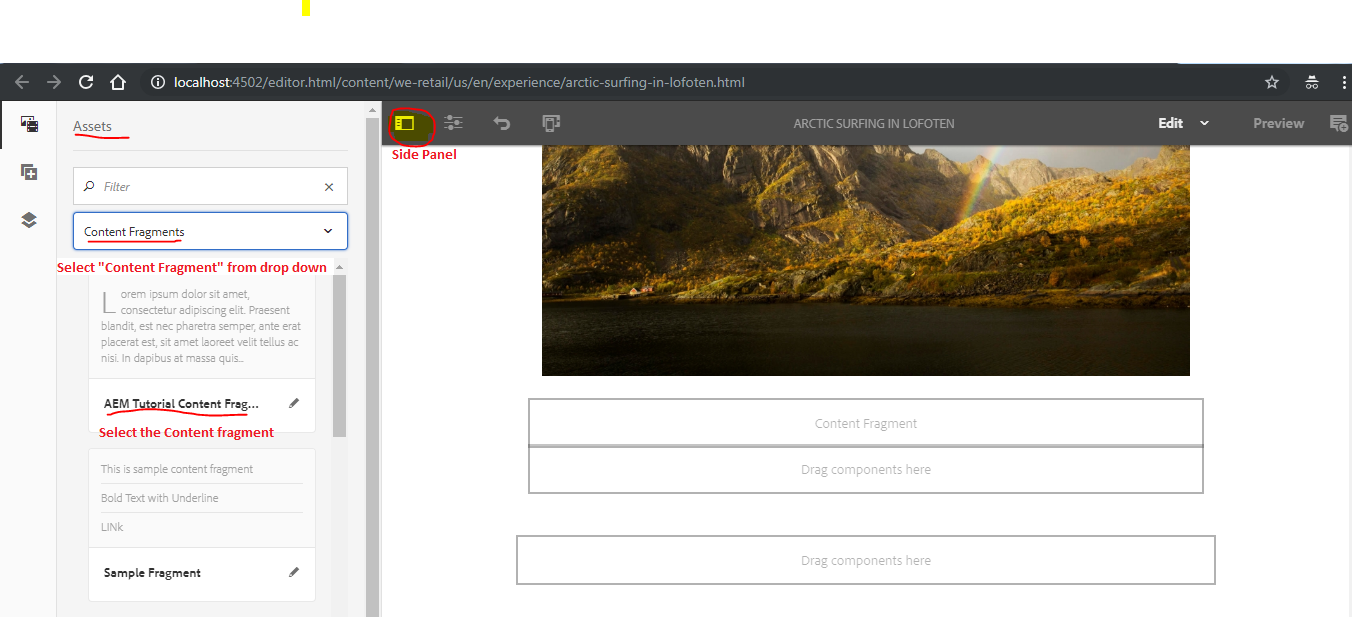
###### CREATING PROXY COMPONENT FROM CONTENT FRAGMENTS

* The content fragments created can be used on the page using Content fragment component. It is core components.
* Ideally – We should not use core component directly. Create a proxy component by extending core components.



AUTHORING CONTENT FRAGMENT

1. Open the page where we want to drop the content fragment (Proxy Component)
2. Drag and drop the desired content fragment. This will show the Master copy on the page

****

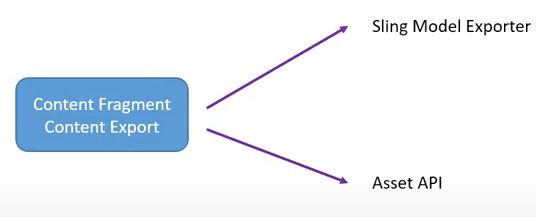
|  |  |
| --- | --- |
| **SELECTING VARIATION**   1. Edit the content fragment and select the variation e.g “Mobile App” variation |  |

DISPLAY MODE IN CONTENT FRAGMENT COMPONENT

|  |  |
| --- | --- |
|  |  |
| The contnet fragment can be displayed in different modes   1. **SINGLE TEXT ELEMENT** : Any one field can be selected from the cotent fragment field 2. **MULTIPLE ELEMENTS** : It’s a multi field component – Multiple fields can be selected from the content fragments. | |

#### EXPORTING CONTENT FRAGMENTS

* Contnet fragment can be exported in JSON format. There are 2 ways – the content can be exported



##### EXPORTING CONTENT FRAGMENTS USING SLING MODEL EXPORTER

* When we use Sling model Export – It outputs only the variation which has been used on the page.
* <http://localhost:4502/content/training/us/en/jcr:content/root/container/container/trainingcontentfragm.model.json>

##### EXPORTING CONTENT FRAGMENTS USING ASSERT API

* When we use Asset API – It outputs all the variations(as variation property in the output JSON) + master variation of the content fragment.
* Access the content fragment – which has to be exposed : Go to Navigation 🡪Assets 🡪 Files
* Edit the content fragment: [http://localhost:4502/~~editor.html/content/dam~~/training/content-fragments/en/author-details](http://localhost:4502/editor.html/content/dam/training/content-fragments/en/author-details)

<http://localhost:4502/api/assets/training/content-fragments/en/author-details.json>

### DISPATCHER

* Dispatcher is a module which is loaded/configured in the Apache webserver. Role of dispatcher module includes
  + **SECURITY**
  + **CACHING**
  + **LOAD BALANCING.**

The dispatcher set-up in a Apache web-server need 2 important files

* **dispatcher.any** : This contain the dispatcher configuration files like caching configuration , connection info with the publish instance etc.
* **httpd.conf**: In this file we configure the dispatcher module in apache.

#### DISPATCHER SET UP

* Download the Apache webserver
* Download the dispatcher module(dll file)
* Copy the dispatcher dll in the module folder of Apache.
* Update the **httpd.conf** to load the module to load the dispatcher module



|  |  |
| --- | --- |
| * Configure the path of dispatcher.any file & dispatcher.log file in httpd.conf file. |  |
| * Copy and paste the directory tag in the httpd.cong file. | <Directory />  <IfModule disp\_apache2.c>  ModMimeUsePathInfo On  # enable dispatcher for ALL request. if this is too restrictive,  # move it to another location  SetHandler dispatcher-handler  </IfModule>    Options FollowSymLinks  AllowOverride None  </Directory> |

* Copy the dispatcher.any file in apache’s “conf” folder
* Configure the dispathcher.any file
* Restart the Apache Server.

#### CONFIGURING dispathcher.any FILE

<https://experienceleague.adobe.com/docs/experience-manager-dispatcher/using/configuring/dispatcher-configuration.html?lang=en>

SAMPLE dispatcher.any

|  |
| --- |
| # 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** | * Configure farms for each site * Parent Node of eitire configuration for each side |
| **clientheaders** | List of all HTTP headers that is allowed |
| **virtualhost** | Host name of the site (e.g - "\*.uhcmedicaresolutions.com") |
| **renders** | We configure the AEM publish instance . For example  /renders  {  /publish1  {  /hostname "apsrp11277.uhc.com"  /port "8080"  }  } |
| **filter** | * The configures the valid Urls. * As a bsest practice – we deny all the Urls and then allow the specific urls * We set rules of each url/urls types designated with a unique number. E.g   # Deny everything first and then allow specific entries  /0001 { /type "allow" /glob "\*" } |
| **cache** | * **/docroot** - Path of cache directory .e.g - **/ "/ebiz/cache/publish"** * /statfileslevel – * / **serveStaleOnError :** if set to 1. This wil render the cached content even if publish is down * **/rules :** This includes the caching rules . i.e what files needs to be cached * **/** **invalidate –** This include the rules – which all files will be flushed out when flush agent will execute. |
| **allowClient** | * This can be used to allow or block specific IP |
| **statistics** | The statistics define, how the load should be balanced among the renders(publish instance) according to the media-type.  /statistics  {  /categories  {  /html  { /glob "\*.html" }  /others  { /glob "\*" }  }  } |

#### METHOD OF 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.

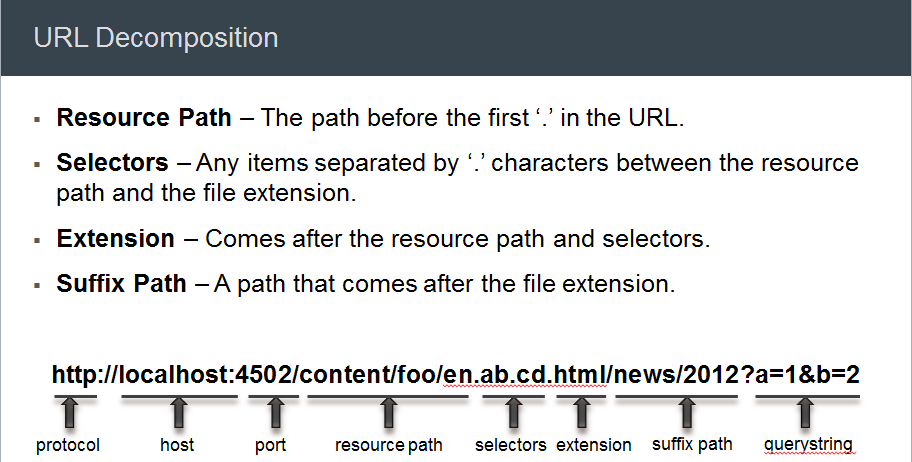
**CONTENT UPDATES :** In a content update, one or more AEM documents change. AEM sends a syndication request to the Dispatcher, which updates the cache accordingly:

**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.

#### WHAT DISPATCHER CAN CACHE ?



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

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

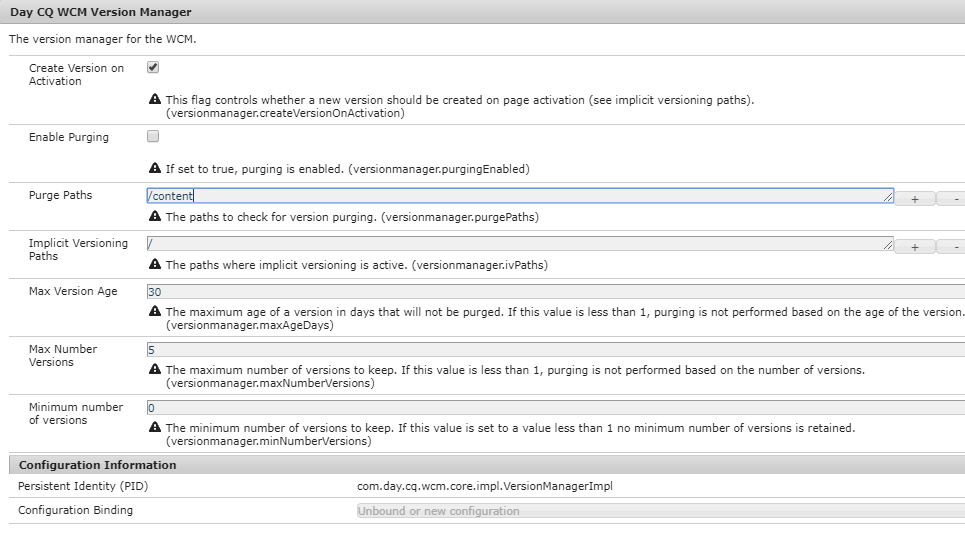
### VERSIONING

|  |  |  |
| --- | --- | --- |
|  |  | * 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. |

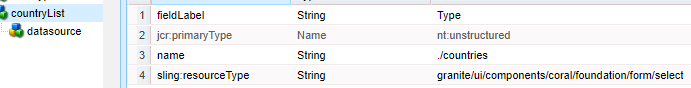
**CHANGE THE AUTHORING MODE**

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

### [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**

|  |
| --- |
|  |