# Day 01: Introduction to AEM

Suppose three friends, Tom, Dick, and Harry decided to open a confectionary that makes muffins. They have a secret recipe for cooking muffins 😋 which makes them tastier and cheaper than the other muffin brands in the market.

Thus they have a good product and now they are ready to make money. They waited and waited and waited for the customers to come to their shop but no one came. They were perplexed 😕 and pondered upon the cause. Then they came to know that very few people know about their confectionary.

But why? Then they realized they put all of their efforts into making their product better but very little time was devoted to the marketing. Thus, due to the lack of proper marketing, even a good product suffered. Proper and targeted marketing is very important in today's world.

One of the major means of marketing is the internet i.e. **online presence** of an organization is very important. If you ask top companies that what is more important for them than their website and online presence, the answer will most probably be - **NOTHING.**You do want to tell the world - who you are and what you do.

To facilitate this, Adobe Marketing Cloud (or Adobe Experience Cloud) is a good platform. Adobe's marketing suite takes care of various aspects of digital marketing. It has tools like Adobe Campaign, Adobe Analytics, Adobe Target and Adobe Experience Manager (AEM).

In this article, we will dive deep into AEM only. We will discuss what it is and why do enterprises need it.

**Adobe Experience Manager (AEM)**

At very high-level AEM is a content management system that is capable of building websites along with mobile apps, forms, and online communities. It provides the maximum abstraction which allows the non-technical digital marketers to give the desired look and feel to their websites as per business needs. Along with this, all of the digital assets of an organization can be managed in one place.

AEM provides the capability of the real-time preview of the content on the website by incorporating WYSIWYG (what you see is what you get) system. By storing an organization's digital assets in one place, it becomes easy to manage them. AEM can tag, catalogue and organize all digital assets so they are ready to use and available as and when needed.

AEM is made up of mainly following features -

**Sites**

* Here websites of an organization are created.
* AEM Sites has the capacity to build responsive, personalized sites that encourage customer engagement and retention.
* Authors and developers can easily manage and build the website. It is as easy as dragging and dropping content, images, and other media.

For more information on sites, see [this](https://helpx.adobe.com/experience-manager/kt/sites/index/aem-6-4-sites.html).

**Assets**

* It is a Digital Asset Management (DAM) tool that helps an organization to manage all their assets (images, videos, documents etc) easily.
* We can "tag", "annotate", and "catalog" assets.
* Metadata of assets can be created/modified easily.

For more information on assets, see [this](https://helpx.adobe.com/in/experience-manager/6-4/assets/user-guide.html).

**Projects**

* It lets us group resources into one entity.
* Resources can be assets, workflows, team information and many more.

To know more about projects, see [this](https://helpx.adobe.com/experience-manager/6-4/sites/authoring/using/projects.html)

**Communities**

* It helps an organization to bond with their users by creating blogs, forums etc
* Members of a community can use social logins such as Facebook and Twitter

See [this](https://helpx.adobe.com/experience-manager/6-4/communities/using/overview.html), to know more about the features AEM communities provide.

**AEM Forms**

* Document management system within the AEM which makes creating and managing forms and documents paperless, efficient and automated.
* Create interactive, personalized statements that can be accessed anytime from anywhere.
* Develop automated workflows, and merge form data and documents with your existing systems.

Also, see [this](https://helpx.adobe.com/in/aem-forms/6/introduction-aem-forms.html) for more information.

**Why AEM?**

After knowing, what is AEM in the above section, we will be discussing **why do we need it**?

**Content Storage**

AEM lets the organization store their content in the cloud which can be accessed by its members anywhere in the world at any time. This increases productivity and efficiency.

**Tags and Metadata management**

This lets the users tag their assets and modify the metadata of the assets which gives easy access.

**Searching**

This enables the users to search anywhere in the repository in no time. The users can search using various filters which are easily configurable.

**Multi-Site Management (MSM)**

This feature provides the maximum content reusability and allows the common content to be replicated in the multiple sites in no time. Configuring them is quite easy.

**User Management**

This allows an organization to give appropriate permissions to create, read, update, delete in the AEM to different users.

**Integrations**

It is very easy to integrate AEM with other Adobe Experience Cloud solutions such as Campaign, Analytics, Target etc. This allows taking full advantage of various Digital Marketing solutions and increases productivity.

These are few features that make AEM as a favourite choice for Content Management System based applications. For more reasons to use AEM, see [here](https://blog.3sharecorp.com/9-top-benefits-of-outsourcing-your-adobe-experience-manager-aem).

# Day 02: AEM Architecture

In the previous [post](https://aem.redquark.org/2018/10/day-01-introduction-to-aem.html), we discussed the basics of AEM and the reason behind its hype in the Digital Marketing space. In this post, we will go a bit more technical and will try to understand the architecture or the basic building blocks of AEM.  
  
Hence, without wasting more time, let's dive into the AEM architecture.

|  |
| --- |
|  |
| image source: <https://prezi.com/i97-ptdowxfi/aemcq-architecture/> |

#### Java Runtime Environment (JRE)

AEM is a Java based web application hence it requires server-side Java Runtime Environment (JRE).

#### Granite Platform

It is Adobe's open web stack and it forms the technical base on which AEM is built. It also provides the foundation UI framework and its major goals are to -

* provide granular UI widgets
* implement best UI practices
* provide an extensible UI

|  |
| --- |
|  |
| image source: [aemstack](https://aemstack.wordpress.com/2015/03/09/granite-ui-osgi-servlet-engine/) |

#### OSGi Framework

|  |
| --- |
|  |
| image source: [OSGi org](https://www.osgi.org/developer/architecture/layering-osgi/) |

It is a set of specifications. Its core specification defines a component and service model for Java. A practical advantage of OSGi is that every software component can define its API via a set of exported Java packages and that every component can specify its required dependencies.

The components and services can be dynamically installed, activated, de-activated, updated and uninstalled.

The OSGi specification has several implementations, for example, Eclipse Equinox, Knopflerfish OSGi or Apache Felix. AEM uses Apache Felix in its tech stack.

For more details see this [post](https://aem.redquark.org/2018/10/osgi-deep-dive-part-1.html).

#### Java Content Repository (JCR)

This combines the attributes of file systems and RDBMS and tries to provide the best of both worlds. According to [JSR 283](http://jcp.org/en/jsr/detail?id=283), "the Java Content Repository API defines an abstract model and a Java API for data storage and related services commonly used by content-oriented applications."

The JCR storage model is a tree of nodes and properties: nodes are used to organize the content and named properties store the actual data, either as simple types (string, boolean, number, etc.) or as binary streams for storing files of arbitrary size.

AEM 6.x uses ***Apache Oak*** as the JCR implementation.

#### Apache Sling

AEM is built using Sling, a Web application framework based on REST principles that provide easy development of content-oriented applications.

Sling uses a JCR repository, such as Apache Jackrabbit, or in the case of AEM, the CRX Content Repository, as its data store.

From Apache Sling's official documentation, ***Sling maps HTTP request URLs to content resources based on the request's path, extension and selectors. Using convention over configuration, requests are processed by scripts and servlets, dynamically selected based on the current resource. This fosters meaningful URLs and resource-driven request processing, while the modular nature of Sling allows for specialized server instances that include only what is needed.***

Thus, anything present in the JCR can be accessed in a RESTful way using HTTP requests.

#### AEM Modules

On top of the above technology stack, there are AEM specific modules that run. These modules are AEM Sites, AEM Assets, Workflows etc.

#### Custom Modules/Code

On top of everything, the organization-specific code runs which is according to their specific needs. In the upcoming posts, we will be learning to do this only - creating custom code on top of AEM.

# Day 04: Developing First OSGi Bundle

In this post, we will be creating and deploying our custom bundle into AEM.  
While doing so, we will be learning concepts of **OSGi Components** and **OSGi Services**.

**Software Modularity**

In modern times, complex software can be thought of as a collection of various modules or components. These modules are normally independent of each other and modification in one module does not affect the other modules.

These modules interact with each other via an API. The API is defined as a set of classes and methods which can be used from other components.

If a module uses an API from another module, it has a dependency on the other module, i.e., it requires the other module exists and works correctly.

A module which is used by other components should try to keep its API stable. This avoids that a change affects other modules. But it should be free to change its internal implementation.

**What is OSGi?**

OSGi stands for ***Open Service Gateway initiative.***It is a Java framework for developing and deploying modular software programs and libraries.

OSGi has two parts.

* The first part is a specification for modular components called bundles, which are commonly referred to as plug-ins. The specification defines an infrastructure for a bundle's life cycle and determines how bundles will interact.
* The second part of OSGi is a Java Virtual Machine (JVM)-level service registry that bundles can use to publish, discover and bind to services in a service-oriented architecture (SOA).

The components and services can be dynamically installed, activated, de-activated, updated and uninstalled.

The OSGi specification has several implementations, for example, **Equinox**, **Knopflerfish**,and **Apache Felix**. AEM uses [Apache Felix](http://felix.apache.org/) implementation. A bundle is the smallest unit of the modularization which means in OSGi, a software component is a bundle.

In AEM, along with out of the box bundles, we can also install our custom bundles very easily. So without further ado, let us start creating an OSGi bundle.

**Creating a Bundle**

We will now create an OSGi bundle which has OSGi service which reads JSON data using HTTP GET request from the URL - <https://jsonplaceholder.typicode.com/todos/>. Following are the steps to create an OSGi bundle -

* Create an AEM Multimodule Project in Eclipse. If you don't know how to do that, you can see [here](https://aem.redquark.org/2018/10/day-03-setting-up-aem-development.html).
* Create an interface **ReadJsonService** in the package **org.redquark.demo.core.services**. This will be exposed as our service.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.services; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* Service which will be exposed |
|  | \*/ |
|  | public interface ReadJsonService { |
|  |  |
|  | /\*\* |
|  | \* @return JSON String |
|  | \*/ |
|  | public String getData(); |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/8b1df9d8fedd8990d32d02f7e6213e53/raw/7c68d430c77cca3c7d8fd2b5c578462181205fce/ReadJsonService.java)[**ReadJsonService.java**](https://gist.github.com/ani03sha/8b1df9d8fedd8990d32d02f7e6213e53#file-readjsonservice-java) hosted with ❤ by [**GitHub**](https://github.com/)

* Now we need to create an implementation class for this service which will have the business logic. Hence, create a class **ReadJsonDataImpl** in the package **org.redquark.demo.core.services.impl**.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.services.impl; |
|  |  |
|  | import org.osgi.service.component.annotations.Component; |
|  | import static org.redquark.demo.core.constants.AppConstants.URL; |
|  | import org.redquark.demo.core.services.ReadJsonService; |
|  | import org.redquark.demo.core.utils.Network; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* Implementation of ReadJsonService |
|  | \*/ |
|  | @Component(immediate = true, service = ReadJsonService.class) |
|  | public class ReadJsonDataImpl implements ReadJsonService { |
|  |  |
|  |  |
|  | /\*\* |
|  | \* Overridden method which will read the JSON data via an HTTP GET call |
|  | \*/ |
|  | @Override |
|  | public String getData() { |
|  |  |
|  | String response = Network.readJson(URL); |
|  |  |
|  | return response; |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/cf79b34f541e4583de764732878466bc/raw/dda69328874c6dbc8709d7a7f05409cd243c1d93/ReadJsonServiceImpl.java)[**ReadJsonServiceImpl.java**](https://gist.github.com/ani03sha/cf79b34f541e4583de764732878466bc#file-readjsonserviceimpl-java) hosted with ❤ by [**GitHub**](https://github.com/)

* Now create constants class for storing project constants and Network class for the HTTP requests

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.constants; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* This class has all the project related constants |
|  | \*/ |
|  | public final class AppConstants { |
|  |  |
|  | public static final String URL = "https://jsonplaceholder.typicode.com/todos/"; |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/7e2ebd8dfe27df8cf360bea535e193c6/raw/94665e28952ce2c25b965debcf6b25a146e766f5/AppConstants.java)[**AppConstants.java**](https://gist.github.com/ani03sha/7e2ebd8dfe27df8cf360bea535e193c6#file-appconstants-java) hosted with ❤ by [**GitHub**](https://github.com/)

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.utils; |
|  |  |
|  | import java.io.BufferedReader; |
|  | import java.io.IOException; |
|  | import java.io.InputStreamReader; |
|  | import java.net.MalformedURLException; |
|  | import java.net.URL; |
|  |  |
|  | import javax.net.ssl.HttpsURLConnection; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* This class has all the Network related calls |
|  | \*/ |
|  | public final class Network { |
|  |  |
|  | private static final String USER\_AGENT = "Mozilla/5.0"; |
|  |  |
|  | public static String readJson(String url) { |
|  |  |
|  | try { |
|  | /\*\* |
|  | \* Get the URL object from the passed url string |
|  | \*/ |
|  | URL requestURL = new URL(url); |
|  |  |
|  | /\*\* |
|  | \* Creating an object of HttpURLConnection |
|  | \*/ |
|  | HttpsURLConnection connection = (HttpsURLConnection) requestURL.openConnection(); |
|  |  |
|  | /\*\* |
|  | \* Setting the request method |
|  | \*/ |
|  | connection.setRequestMethod("GET"); |
|  |  |
|  | /\*\* |
|  | \* Setting the request property |
|  | \*/ |
|  | connection.setRequestProperty("User-Agent", USER\_AGENT); |
|  |  |
|  | /\*\* |
|  | \* Get the response code |
|  | \*/ |
|  | int responseCode = connection.getResponseCode(); |
|  |  |
|  | if(responseCode == HttpsURLConnection.HTTP\_OK) { |
|  |  |
|  | /\*\* |
|  | \* Getting an instance of BufferedReader to read the response returned |
|  | \*/ |
|  | BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream())); |
|  |  |
|  | /\*\* |
|  | \* String which will read the response line by line |
|  | \*/ |
|  | String inputLine; |
|  |  |
|  | /\*\* |
|  | \* StringBuffer object to append the string as a whole |
|  | \*/ |
|  | StringBuffer response = new StringBuffer(); |
|  |  |
|  | /\*\* |
|  | \* Read until empty line is encountered |
|  | \*/ |
|  | while ((inputLine = in.readLine()) != null) { |
|  |  |
|  | /\*\* |
|  | \* Append each line to make the response as a whole |
|  | \*/ |
|  | response.append(inputLine); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Closing the BufferedReader to avoid memory leaks |
|  | \*/ |
|  | in.close(); |
|  |  |
|  | /\*\* |
|  | \* Return the response |
|  | \*/ |
|  | return response.toString(); |
|  | } |
|  |  |
|  | } catch (MalformedURLException e) { |
|  | e.printStackTrace(); |
|  | } catch (IOException e) { |
|  | e.printStackTrace(); |
|  | } |
|  |  |
|  | return ""; |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/106f3fe19d2c4359c552b1c73a47cedc/raw/64e4771e04a03ad40a46f18e2aa847c539fbb370/Network.java)[**Network.java**](https://gist.github.com/ani03sha/106f3fe19d2c4359c552b1c73a47cedc#file-network-java) hosted with ❤ by [**GitHub**](https://github.com/)

* Let us understand this code. If you see the **ReadJsonServiceImpl.java**closely, you will see there is **a @Component**annotation. This annotation signifies that the given class is a component in AEM and the property **service = ReadJsonService.class**signifies that this is a service which is exposed via the **ReadJsonService** interface.

**OSGi Components and Services**

A service is an object that is registered in the OSGi Service Registry and can be looked up using its interface name(s). The only prerequisite is that a service should implement some interface. For example, I could register a runnable object under the java.lang.Runnable interface and clients could look it up using that interface name.

A "component" tends to be an object whose lifecycle is managed, usually by a component framework such as Declarative Services (DS), Blueprint or iPOJO.

A component may have any of the following features, in combination or alone:

1. A component may be started and stopped; this would be considered an "active" component, though that is also an informal term. A component that doesn't need to be started or stopped is called passive.
2. A component may publish itself as an OSGi service.
3. A component may bind to or consume OSGi services.
4. In general, using a component framework is the easiest way to work with OSGi services because the framework will manage the binding to the services that you want to consume. For example, you could say that your component "depends on" a particular service, in which case the component will only be created and activated when that service is available -- and also it will be destroyed when the service becomes unavailable.

* Once the Java code is done, we can now deploy the code using eclipse and maven (see [this post](https://aem.redquark.org/2018/10/day-03-setting-up-aem-development.html) for the steps).
* Now go to the AEM instance and in CRXDE, you should be seeing your project under the **/apps**folder.

|  |
| --- |
|  |
| Demo project |

* Right-click on **/apps/<your-project>/content/**, then **Create...**and then **Create Component...**
* In the dialog, fill the values as shown in the below screenshot

|  |
| --- |
|  |
| Create a Component |

* Click **Next**, then **OK**
* Go to **/apps/<your-project>/components/content/sampleComponent**and open the **sampleComponent.jsp** file and add the below code and save. Note: We are using JSP only for simplicity. In the latest versions of AEM, we should use HTL (formerly called Sightly) and the use of JSP is discouraged.

|  |  |
| --- | --- |
|  | <%@include file="/libs/foundation/global.jsp"%> |
|  | <% |
|  | org.redquark.demo.core.services.ReadJsonService service = sling.getService(org.redquark.demo.core.services.ReadJsonService.class); |
|  |  |
|  | String result = service.getData(); |
|  |  |
|  | %> |
|  |  |
|  | <h2>This page invokes the AEM ReadJsonService</h2> |
|  | <h3>RESPONSE: <%=result%></h3> |

[**view raw**](https://gist.github.com/ani03sha/69e881bd24837249bdc0d592efc12aa9/raw/bfe2446252563e20fd6715cb36d88029a106ca8c/sampleComponent.jsp)[**sampleComponent.jsp**](https://gist.github.com/ani03sha/69e881bd24837249bdc0d592efc12aa9#file-samplecomponent-jsp) hosted with ❤ by [**GitHub**](https://github.com/)

* Add the sample component on a page and you should be able to see the JSON string returned on that page as follows

[

{ "userId": 1, "id": 1, "title": "delectus aut autem", "completed": false },

{ "userId": 1, "id": 2, "title": "quis ut nam facilis et officia qui", "completed": false },

{ "userId": 1, "id": 3, "title": "fugiat veniam minus", "completed": false },

{ "userId": 1, "id": 4, "title": "et porro tempora", "completed": true },...

          ]

* You can check if your component and service are installed in the AEM server at <http://localhost:4502/system/console/components/> and <http://localhost:4502/system/console/services/> respectively.

# [Day 05: Working with Sling Servlets in AEM](https://aem.redquark.org/2018/10/day-05-working-with-sling-servlets-in_10.html)

A Servlet is a class used to extend the capabilities of servers that host applications accessed by means of a **request-response**programming model. For such applications, Servlet technology defines HTTP-specific servlet classes.  
All servlets must implement the Servlet interface, which defines life-cycle methods. When implementing a generic service, we can use or extend the **GenericServlet**class provided with the Java Servlet API. The **HttpServlet**class provides methods, such as **doGet()** and **doPost()**, for handling HTTP-specific services.

## Sling Servlets

In Sling, servlets can be registered as services. Whenever you create an AEM Multimodule Project (How? see [here](https://aem.redquark.org/2018/10/day-03-setting-up-aem-development.html)) using maven archetype 14, the default template for a Sling Servlet is generated as follows -

|  |  |
| --- | --- |
|  | /\* |
|  | \* Copyright 2015 Adobe Systems Incorporated |
|  | \* |
|  | \* Licensed under the Apache License, Version 2.0 (the "License"); |
|  | \* you may not use this file except in compliance with the License. |
|  | \* You may obtain a copy of the License at |
|  | \* |
|  | \* http://www.apache.org/licenses/LICENSE-2.0 |
|  | \* |
|  | \* Unless required by applicable law or agreed to in writing, software |
|  | \* distributed under the License is distributed on an "AS IS" BASIS, |
|  | \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. |
|  | \* See the License for the specific language governing permissions and |
|  | \* limitations under the License. |
|  | \*/ |
|  | package org.redquark.demo.core.servlets; |
|  |  |
|  | import java.io.IOException; |
|  |  |
|  | import javax.servlet.Servlet; |
|  | import javax.servlet.ServletException; |
|  |  |
|  | import org.apache.sling.api.SlingHttpServletRequest; |
|  | import org.apache.sling.api.SlingHttpServletResponse; |
|  | import org.apache.sling.api.resource.Resource; |
|  | import org.apache.sling.api.servlets.HttpConstants; |
|  | import org.apache.sling.api.servlets.SlingAllMethodsServlet; |
|  | import org.apache.sling.api.servlets.SlingSafeMethodsServlet; |
|  | import org.osgi.framework.Constants; |
|  | import org.osgi.service.component.annotations.Component; |
|  |  |
|  | import com.day.cq.commons.jcr.JcrConstants; |
|  |  |
|  | /\*\* |
|  | \* Servlet that writes some sample content into the response. It is mounted for |
|  | \* all resources of a specific Sling resource type. The |
|  | \* {@link SlingSafeMethodsServlet} shall be used for HTTP methods that are |
|  | \* idempotent. For write operations use the {@link SlingAllMethodsServlet}. |
|  | \*/ |
|  | @Component(service=Servlet.class, |
|  | property={ |
|  | Constants.SERVICE\_DESCRIPTION + "=Simple Demo Servlet", |
|  | "sling.servlet.methods=" + HttpConstants.METHOD\_GET, |
|  | "sling.servlet.resourceTypes="+ "demoproject/components/structure/page", |
|  | "sling.servlet.extensions=" + "txt" |
|  | }) |
|  | public class SimpleServlet extends SlingSafeMethodsServlet { |
|  |  |
|  | private static final long serialVersionUID = 1L; |
|  |  |
|  | @Override |
|  | protected void doGet(final SlingHttpServletRequest req, |
|  | final SlingHttpServletResponse resp) throws ServletException, IOException { |
|  | final Resource resource = req.getResource(); |
|  | resp.setContentType("text/plain"); |
|  | resp.getWriter().write("Title = " + resource.getValueMap().get(JcrConstants.JCR\_TITLE)); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/a69b10a726180d48c5657e0582f1afcb/raw/7bcea359f30d46b8ec8ba31058ec058d8ae494dd/SimpleServlet.java)[**SimpleServlet.java**](https://gist.github.com/ani03sha/a69b10a726180d48c5657e0582f1afcb#file-simpleservlet-java) hosted with ❤ by [**GitHub**](https://github.com/)

Let us discuss its parts one by one

* **@Component annotation -**This OSGi annotation signifies that the class is a Service component and is not processed at runtime by a Service Component Runtime (SCR) implementation. It must be processed by tools and used to add a Component Description to the bundle.
* **service property -**This signifies the types under which to register this Component as a service. In our case, the component is registered under the Servlet type.
* **Constants.SERVICE\_DESCRIPTION -**Defines standard names for the OSGi environment system properties, service properties, and Manifest header attribute keys. In our case, we are defining the description of our servlet.
* **sling.servlet.methods -**Defines which methods this servlet will use. In our case, we are using the HTTP GET method.
* **sling.servlet.resourceTypes -**This property defines how are we accessing the servlet. There are two ways - via resource types and via paths. We will discuss this in the next section.
* **sling.servlet.extensions -**The request URL extensions supported by the servlet for requests. The property value must either be a single String, an array of Strings or a Vector of Strings. This property is only considered for the registration with sling.servlet.resourceTypes.

## Servlet Registration

A **SlingServletResolver** listens for Servlet services and - given the correct service registration properties - provides the servlets as resources in the (virtual) resource tree. Such servlets are provided as **ServletResource** instances which adapt to the ***javax.servlet.Servlet*** class.

For a Servlet registered as an OSGi service to be used by the Sling Servlet Resolver, either one or both of the ***sling.servlet.paths*** or the ***sling.servlet.resourceTypes*** service reference properties must be set. If neither is set, the Servlet service is ignored.

A Sling servlet can be registered in two ways -

#### Using Resource Types - **Using this way, we use the**sling:resourceType**property of the node. For this, we need to hit the path in the browser for which the**sling:resourceType**is the given one.**

#### Using Paths - **Using this way, we can directly use the path specified in the request and our servlet will be executed.**

#### **Registering a servlet via a path is relatively easier than registering it via a resource type but it is still recommended to use resource types instead of paths. Below are the reasons why -**

#### **If we use paths, then we need to be extra careful as we do not want to give any path randomly. The servlets cannot be access-controlled using the default JCR ACLs.**

#### **No suffix handling in the path bound servlets.**

#### **We also need to specify the paths to the servlet consumers and change in the path can have a serious impact.**

## Types of Servlets

#### **There are two types of servlets in Sling which are nothing but the classes we need to extend while creating our servlet.**

#### SlingSafeMethodsServlet - **If we want to use only the read-only methods then we use this. This base class is actually just a better implementation of the Servlet API**HttpServlet**class which accounts for extensibility. So extensions of this class have great control over what methods to overwrite. It supports GET, HEAD, OPTIONS etc methods.**

#### SlingAllMethodsServlet - **If we want to use methods that write as well, then we use this. This class extends the**SlingSafeMethodsServlet**by support for the POST, PUT and DELETE methods.**

## Sling Servlet Example

#### **In this section, we will be creating our custom Sling Servlet to fetch the JSON data from a RESTful webservice.**

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.servlets; |
|  |  |
|  | import javax.servlet.Servlet; |
|  |  |
|  | import org.apache.sling.api.SlingHttpServletRequest; |
|  | import org.apache.sling.api.SlingHttpServletResponse; |
|  | import org.apache.sling.api.servlets.HttpConstants; |
|  | import org.apache.sling.api.servlets.SlingSafeMethodsServlet; |
|  | import org.osgi.framework.Constants; |
|  | import org.osgi.service.component.annotations.Component; |
|  | import static org.redquark.demo.core.constants.AppConstants.URL; |
|  | import org.redquark.demo.core.utils.Network; |
|  | import org.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* This servlet uses the HTTP GET method to read a data from the RESTful webservice |
|  | \*/ |
|  | @Component(service = Servlet.class, property = { |
|  | Constants.SERVICE\_DESCRIPTION + "=JSON Servlet to read the data from the external webservice", |
|  | "sling.servlet.methods=" + HttpConstants.METHOD\_GET, "sling.servlet.paths=" + "/bin/readjson" }) |
|  | public class JSONServlet extends SlingSafeMethodsServlet { |
|  |  |
|  | /\*\* |
|  | \* Generated serialVersionUID |
|  | \*/ |
|  | private static final long serialVersionUID = 4438376868274173005L; |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(JSONServlet.class); |
|  |  |
|  | @Override |
|  | protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) { |
|  |  |
|  | try { |
|  |  |
|  | log.info("Reading the data from the webservice"); |
|  |  |
|  | /\*\* |
|  | \* Getting the JSON string from the webservice |
|  | \*/ |
|  | String responseString = Network.readJson(URL); |
|  |  |
|  | /\*\* |
|  | \* Writing the entire JSON string on the browser |
|  | \*/ |
|  | response.getWriter().println(responseString); |
|  |  |
|  | } catch (Exception e) { |
|  |  |
|  | log.error(e.getMessage(), e); |
|  | } |
|  | } |
|  |  |
|  | } |

#### [view raw](https://gist.github.com/ani03sha/e68fe4d107062fb620cba72c3bb00e4a/raw/4c21b61bf92623a1b359a73fad5ed9d1d228027d/JSONServlet.java)[JSONServlet.java](https://gist.github.com/ani03sha/e68fe4d107062fb620cba72c3bb00e4a#file-jsonservlet-java) hosted with ❤ by [GitHub](https://github.com/)

#### **As you can see that we are using**Network.java**and**AppConstants.java**in the code. This you can find in my**[**previous post**](https://aem.redquark.org/2018/10/day-04-developing-first-osgi-bundle.html)**.**

After deploying the code on your AEM server (how? see [here](https://aem.redquark.org/2018/10/day-03-setting-up-aem-development.html)), hit the URL [http://<host>:<port>/bin/readjson](http://localhost:4502/bin/readjson). You will see JSON string returned on your screen.

# Day 06: Playing with Sling Post Servlet

In our [previous post](https://aem.redquark.org/2018/10/day-05-working-with-sling-servlets-in_10.html), we discussed the Sling Servlets and in the latter part of the post, we created a custom servlet which uses the HTTP GET method.  
In this post, we are going to create a Sling Post servlet which is invoked by an HTTP POST call. We will make an AJAX call to achieve this.  
  
Since this a POST request, we will be using ***SlingAllMethodsServlet***as a parent to our custom servlet.

**Sling Post Servlet Example**

in this example, we will make an HTTP post request via AJAX to add a node at a path in the JCR. After adding the node, we will also modify the node by adding a***name***property.

* Create an AEM Multimodule project in eclipse (how? see [here](https://aem.redquark.org/2018/10/day-03-setting-up-aem-development.html))
* Create a class ***SampleSlingPostServlet***and paste the following code in it.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.servlets; |
|  |  |
|  | import javax.jcr.Node; |
|  | import javax.jcr.RepositoryException; |
|  | import javax.servlet.Servlet; |
|  |  |
|  | import org.apache.sling.api.SlingHttpServletRequest; |
|  | import org.apache.sling.api.SlingHttpServletResponse; |
|  | import org.apache.sling.api.resource.PersistenceException; |
|  | import org.apache.sling.api.resource.Resource; |
|  | import org.apache.sling.api.resource.ResourceResolver; |
|  | import org.apache.sling.api.servlets.HttpConstants; |
|  | import org.apache.sling.api.servlets.SlingAllMethodsServlet; |
|  | import org.osgi.framework.Constants; |
|  | import org.osgi.service.component.annotations.Component; |
|  | import org.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* This class shows the usage of SlingPostServlet |
|  | \*/ |
|  | @Component(service = Servlet.class, property = { Constants.SERVICE\_DESCRIPTION + "=Simple Demo Servlet", |
|  | "sling.servlet.methods=" + HttpConstants.METHOD\_POST, "sling.servlet.paths=" + "/bin/submitdata" }) |
|  | public class SampleSlingPostServlet extends SlingAllMethodsServlet { |
|  |  |
|  | /\*\* |
|  | \* Generated serialVersionUID |
|  | \*/ |
|  | private static final long serialVersionUID = -159625176093879129L; |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(SampleSlingPostServlet.class); |
|  |  |
|  | /\*\* |
|  | \* Overridden doPost() method which is invoked when an HTTP post request is made |
|  | \*/ |
|  | @Override |
|  | protected void doPost(SlingHttpServletRequest request, SlingHttpServletResponse response) { |
|  |  |
|  | try { |
|  |  |
|  | /\*\* |
|  | \* Getting the instance of resource resolver from the request |
|  | \*/ |
|  | ResourceResolver resourceResolver = request.getResourceResolver(); |
|  |  |
|  | /\*\* |
|  | \* Getting the resource object via path |
|  | \*/ |
|  | Resource resource = resourceResolver.getResource("/content/demoproject/en"); |
|  |  |
|  | log.info("Resource is at path {}", resource.getPath()); |
|  |  |
|  | /\*\* |
|  | \* Adapt the resource to javax.jcr.Node type |
|  | \*/ |
|  | Node node = resource.adaptTo(Node.class); |
|  |  |
|  | /\*\* |
|  | \* Create a new node with name and primary type and add it below the path specified by the resource |
|  | \*/ |
|  | Node newNode = node.addNode("demoNode", "nt:unstructured"); |
|  |  |
|  | /\*\* |
|  | \* Setting a name property for this node |
|  | \*/ |
|  | newNode.setProperty("name", "Demo Node"); |
|  |  |
|  | /\*\* |
|  | \* Commit the changes to JCR |
|  | \*/ |
|  | resourceResolver.commit(); |
|  |  |
|  | } catch (RepositoryException e) { |
|  |  |
|  | log.error(e.getMessage(), e); |
|  |  |
|  | e.printStackTrace(); |
|  |  |
|  | } catch (PersistenceException e) { |
|  |  |
|  | log.error(e.getMessage(), e); |
|  |  |
|  | e.printStackTrace(); |
|  | } |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/76a6ced1cabe800474c9eb45c3c3a491/raw/abc8ddd5ffb393c8bd2f59f29bd27e154a71ede3/SampleSlingPostServlet.java)[**SampleSlingPostServlet.java**](https://gist.github.com/ani03sha/76a6ced1cabe800474c9eb45c3c3a491#file-sampleslingpostservlet-java) hosted with ❤ by [**GitHub**](https://github.com/)

* Here we are registering the servlet via path ***/bin/submitdata.***Property ***sling.servlet.methods***define the HTTP method to be used i.e. POST in this case. Since we are writing the data, we have extended the class from ***SlingAllMethodsServlet***.
* The logic in the ***doPost()***creates a node under a given path and added a ***name*** property in it.
* Now to call this servlet via AJAX, create a new ***cq:component*** (how? see [here](https://aem.redquark.org/2018/10/day-04-developing-first-osgi-bundle.html)) and in the HTML file of the component, add below code

|  |  |
| --- | --- |
|  | AJAX call for SampleSlingPostServlet |
|  |  |
|  | <script> |
|  | $.ajax({ |
|  | type : "POST", |
|  | url : '/bin/submitdata', |
|  | /\*data : { |
|  | pass your request parameter here, currently we are not passing any data |
|  | },\*/ |
|  | success : function(data, textStatus, jqXHR) { |
|  | //write your logic that you need to perform on sucess |
|  | }, |
|  | error : function(XMLHttpRequest, textStatus, errorThrown) { |
|  | //write your logic that you need to perform on error |
|  | } |
|  | }); |
|  | </script> |

[**view raw**](https://gist.github.com/ani03sha/1745e284e4272a6f18d11c48ee9fdd7f/raw/ced414aa2e48a8dd3c481f90bda887cad0ebdb0b/ajaxComponent.html)[**ajaxComponent.html**](https://gist.github.com/ani03sha/1745e284e4272a6f18d11c48ee9fdd7f#file-ajaxcomponent-html) hosted with ❤ by [**GitHub**](https://github.com/)

* Note that the url property has the same path by which the servlet is registered.
* When we drag and drop the component on an AEM page, the post servlet will invoke and a node with name property will be created. You can check the same in CRXDE.

|  |
| --- |
|  |
| CRXDE |

# Day 07: Creating your first component in AEM

Suppose you need to create a website for cute and adorable ***teddy bears***🐻. In this site, on some web pages, there has to be an image of a teddy bear along with its title, some descriptive text and price.  
As you may have guessed, a good way to do it to create a small entity having these fields and which is ***configurable*** and can be ***reused*** on these pages.  
  
Here to our rescue come the AEM Components. A component as its name suggests a reusable entity that can be used anywhere on our website. As per our requirement, we can configure the same component differently on different pages (different look with the same feel 😊). A typical AEM page is composed of many such components.  
  
Components are the building blocks of pages. A component usually contains a JSP or HTML file that contains the HTML markup and maybe some business logic for that component.  
  
Note: Adobe recommends HTL (HTML Template Language) to be used for markup instead of JSP.  
  
In this post, we will be creating one video component that takes video id from YouTube or Vimeo and plays the video on our page. Let's get started.

**Creating the component dialog**

For using the component, we first need to create a proper dialog structure in AEM

* Open CRXDE and move to your project under **/apps**.
* Right-Click on the desired folder, then **Create...**, then **Create Component...**and configure as per the below screenshot and save and **Save All...**

|  |
| --- |
|  |
| Create a Component |

* Create nodes and properties under the component node as per the following XML file

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?> |
|  | <jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0" xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0" xmlns:nt="http://www.jcp.org/jcr/nt/1.0" |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="Video Component" |
|  | sling:resourceType="cq/gui/components/authoring/dialog"> |
|  | <content |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <layout |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/layouts/tabs" |
|  | type="nav"/> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <tab |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="Properties" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <layout |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/layouts/fixedcolumns"/> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <columns |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <title |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/textfield" |
|  | class="field-whitespace" |
|  | fieldDescription="Please enter title of the video" |
|  | fieldLabel="Title" |
|  | name="./title"/> |
|  | <videoLink |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/textfield" |
|  | class="field-whitespace" |
|  | fieldDescription="Enter the valid external webm video link" |
|  | fieldLabel="External Link" |
|  | name="./link" |
|  | required="{Boolean}true"/> |
|  | <play |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/textfield" |
|  | fieldDescription="Enter the text for Play button" |
|  | fieldLabel="Play Button Text" |
|  | name="./play" |
|  | required="{Boolean}true"/> |
|  | <pause |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/textfield" |
|  | fieldDescription="Enter the text for Pause button" |
|  | fieldLabel="Pause Button Text" |
|  | name="./pause" |
|  | required="{Boolean}true"/> |
|  | </items> |
|  | </columns> |
|  | </items> |
|  | </tab> |
|  | </items> |
|  | </content> |
|  | </jcr:root> |

[**view raw**](https://gist.github.com/ani03sha/577661613ffce2a9e4e10a1911e47db0/raw/62df240fa21c6b9155dd675defa4164d703b0c02/video-_content.xml)[**video-\_content.xml**](https://gist.github.com/ani03sha/577661613ffce2a9e4e10a1911e47db0#file-video-_content-xml) hosted with ❤ by [**GitHub**](https://github.com/)

* Now rename the ***video.jsp*** file to ***video.html*** and paste the below code in it

|  |  |
| --- | --- |
|  | <style> |
|  | .first { |
|  | transition: 0.30s; |
|  | -webkit-transition: 0.30s; |
|  | -moz-transition: 0.30s; |
|  | -o-transition: 0.30s; |
|  | -ms-transition: 0.30s; |
|  | font-family: verdana; |
|  | text-decoration: none; |
|  | color: white; |
|  | background-color: #81e55c; |
|  | padding: 9px; |
|  | border-radius: 10px; |
|  | box-shadow: 2px 2px 2px black; |
|  | } |
|  |  |
|  | .first:hover { |
|  | background-color: #a5f488; |
|  | } |
|  |  |
|  | .first:active { |
|  | box-shadow: 0px 0px 0px black; |
|  | } |
|  |  |
|  | .second { |
|  | transition: 0.30s; |
|  | -webkit-transition: 0.30s; |
|  | -moz-transition: 0.30s; |
|  | -o-transition: 0.30s; |
|  | -ms-transition: 0.30s; |
|  | font-family: verdana; |
|  | text-decoration: none; |
|  | color: white; |
|  | background-color: #c0362c; |
|  | padding: 9px; |
|  | border-radius: 10px; |
|  | box-shadow: 2px 2px 2px black; |
|  | } |
|  |  |
|  | .second:hover { |
|  | background-color: #db7f7f; |
|  | } |
|  |  |
|  | .second:active { |
|  | box-shadow: 0px 0px 0px black; |
|  | } |
|  |  |
|  | video { |
|  | -webkit-transition: 0.60s; |
|  | -moz-transition: 0.60s; |
|  | -o-transition: 0.60s; |
|  | -ms-transition: 0.60s; |
|  | border: 1px solid black; |
|  | box-shadow: 0px 0px 10px black; |
|  | } |
|  |  |
|  | video:hover { |
|  | box-shadow: 0px 0px 20px black; |
|  | } |
|  | </style> |
|  |  |
|  | <script> |
|  | window.onload = function() { |
|  | var video = document.getElementById('my-video'); |
|  | var play = document.getElementById('play'); |
|  | var pause = document.getElementById('pause'); |
|  |  |
|  | // associate functions with the 'onclick' events |
|  | play.onclick = playVideo; |
|  | pause.onclick = pauseVideo; |
|  |  |
|  | function playVideo(e) { |
|  | e.preventDefault(); |
|  | video.play(); |
|  | } |
|  |  |
|  | function pauseVideo(e) { |
|  | e.preventDefault(); |
|  | video.pause(); |
|  | } |
|  | } |
|  | </script> |
|  |  |
|  | <body bgcolor="${properties.bgColor}"> |
|  | <center> |
|  | <video style="background-color: ${properties.bgColor" preload="auto" |
|  | width="480" height="270" id="my-video"> |
|  | <source src="${properties.link}"> |
|  | </video> |
|  | <br> <br> |
|  | <div> |
|  | <a class="first" href="#" id="play">${properties.play}</a> <a |
|  | class="second" href="#" id="pause">${properties.pause}</a> |
|  | </div> |
|  | </center> |

[**view raw**](https://gist.github.com/ani03sha/28f3279d1b07e9e78fe7740cd56d07ee/raw/704c98e92313b578731c96ab0214d52ca94c5b89/video.html)[**video.html**](https://gist.github.com/ani03sha/28f3279d1b07e9e78fe7740cd56d07ee#file-video-html) hosted with ❤ by [**GitHub**](https://github.com/)

* This is a simple file where we have applied styling and added some JavaScript logic to run the component. If you look closely we are using ***properties.title***, ***properties.link***, ***properties.play*** and ***properties.pause*** in our HTML code.
* Actually, we are configuring the values from the edit dialog (shown below) and the values we save are getting displayed dynamically in the markup using the ***property***object.

|  |
| --- |
|  |
| Edit Dialog |

* The values of fields in JCR are saved using the ***name***class in each field node. The value of the name property is used to access the value in the JCR. For e.g. to access the External Link field value, we have to use ***property.link.***

|  |
| --- |
|  |
| Node Properties |

* Once you save the dialog, you will see the output as -

|  |
| --- |
|  |
| Video Component Output |

# Day 08: Dueling with JavaScript Use API

Having learnt the nitty-gritty details of component development in AEM in the [previous post](https://aem.redquark.org/2018/10/day-07-creating-your-first-component-in.html), we will now explore further. In this post, we are going to look into the concepts of ***Use*** API.  
  
This API lets us write server-side code which can be consumed by HTL. AEM provides support for two types of ***Use***API - ***Java Use API*** and ***JavaScript Use API***. In this post, we are going to learn about JavaScript Use API and in the next post, we will look into Java Use API.

**JavaScript Use API**

A ***JavaScript Engine*** (for e.g. Google's V8 Engine) is responsible for executing JavaScript in the browser. But when we are using Use API, what makes a JavaScript engine to execute it is [Rhino](https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino). It is an open-source implementation of JavaScript written entirely in Java.

Since JavaScript is translated into Java, we have Java libraries at our disposal. To understand how we can use JavaScript Use API, let's take an example.

**Example**

* Create a simple text component with a basic configuration as follows (how?[see here](https://aem.redquark.org/2018/10/day-07-creating-your-first-component-in.html))

|  |
| --- |
|  |
| Create a component |

* Create nodes and properties under the component node as per the following XML file

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?> |
|  | <jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0" xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0" xmlns:nt="http://www.jcp.org/jcr/nt/1.0" |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="Text component" |
|  | sling:resourceType="cq/gui/components/authoring/dialog"> |
|  | <content |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <layout |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/layouts/tabs" |
|  | type="nav"/> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <tab |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="Properties" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <layout |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/layouts/fixedcolumns"/> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <columns |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <title |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/textfield" |
|  | class="field-whitespace" |
|  | fieldDescription="Enter the title" |
|  | fieldLabel="Title" |
|  | name="./title"/> |
|  | <description |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/textarea" |
|  | class="field-whitespace" |
|  | fieldDescription="Enter the description" |
|  | fieldLabel="Description" |
|  | name="./description"/> |
|  | </items> |
|  | </columns> |
|  | </items> |
|  | </tab> |
|  | </items> |
|  | </content> |
|  | </jcr:root> |

[**view raw**](https://gist.github.com/ani03sha/4d73e42d86be6723830b73b2c0a85157/raw/493eefd74d2ccbc6c5566e29eeddbdc28453a501/text-_content.xml)[**text-\_content.xml**](https://gist.github.com/ani03sha/4d73e42d86be6723830b73b2c0a85157#file-text-_content-xml) hosted with ❤ by [**GitHub**](https://github.com/)

* Now, create a new file ***text.js*** under the path  ***/apps/demoproject/components/content/text*** and paste the following code in it

|  |  |
| --- | --- |
|  | "use strict"; |
|  | use(function() { |
|  |  |
|  | var text = {}; |
|  |  |
|  | text.title = granite.resource.properties["title"]; |
|  | text.description = granite.resource.properties["description"]; |
|  |  |
|  | return text; |
|  | }); |

[**view raw**](https://gist.github.com/ani03sha/a450edf26f8574ec6eba63ceb05178f7/raw/9d44a02780dd1738a8193dafc65e0b03d6a1d42f/text.js)[**text.js**](https://gist.github.com/ani03sha/a450edf26f8574ec6eba63ceb05178f7#file-text-js) hosted with ❤ by [**GitHub**](https://github.com/)

Here, note that we are using***use(function())***which signifies the usage of the Use API and in the function, at line #4, we are initializing a JS object call ***text***.

* Then, we created two members of the ***text*** object, **title** and **description**. we are setting these with the JCR properties title and description respectively. Note that to access these values from the JCR, we are using ***granite.resource.properties[]***object.
* In line #9, we are returning the ***text*** object.
* Now rename ***text.jsp*** to ***text.html***and paste the following code in it.

|  |  |
| --- | --- |
|  | Sample Text |
|  |  |
|  | <div data-sly-use.text="text.js"> |
|  | <h1>Title: ${text.title}</h1> |
|  | <p>Description: ${text.description}</p> |
|  | </div> |

[**view raw**](https://gist.github.com/ani03sha/7031a4d7922c482ad947bce929cc0bc8/raw/72d2008789a15b3d9f916aefccfceff2b826bccd/text.html)[**text.html**](https://gist.github.com/ani03sha/7031a4d7922c482ad947bce929cc0bc8#file-text-html) hosted with ❤ by [**GitHub**](https://github.com/)

Here we are using ***data-sly-use.text="text.js"***which includes the JS file in the HTML code. You have to use the right path of the file if the HTML file and the referred file are not in the same location. This code gets the ***text***object and accesses the properties via dot (.) operator.

* Now drag and drop the component on the page and configure it using the edit dialog. You will see the passed values on your page.

# Day 09: Dueling with Java Use API

We discussed JavaScript Use API and coded server-side logic using JavaScript in the [previous post](https://aem.redquark.org/2018/10/day-08-dueling-with-javascript-use-api.html). Today, we are going to look into a more popular way of dealing with server-side logic using ***Java Use API***.  
We will be creating a simple text component (similar to the one in the previous post) and will use Java backend logic to do our task.

**What on earth is WCMUsePojo?**

You must have heard about [WCMUsePojo](https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/reference-materials/javadoc/com/adobe/cq/sightly/WCMUsePojo.html) term while developing with AEM. What is it? Actually, it is a class that enables us to talk to the front end of our component. It is similar to the JavaScript Use API in usage.

Since we likely need a way to provide back-end logic to components in AEM’s component development mechanism, we write Use API logic

We also know that HTL is a limited template language that allows doing only small basic operations, and the heavy lifting logic should be done inside a Java class or a server-side JS.

This provides better decoupling of the business logic and so your code will be more easily maintainable and also easier to debug. All of this is achieved by extending ***WCMUsePojo*** in our custom class which provides us with the capability to deal with JCR.

**Example**

After understanding the purpose and usage of the API, it's time to see the concepts in action 😮.

* Create a new AEM project using AEM Multimodule Project (how? see [here](https://aem.redquark.org/2018/10/day-03-setting-up-aem-development.html)) and deploy in your AEM instance.
* Go to CRXDE and under your project in **/apps**create a new component with the following configuration

|  |
| --- |
|  |
| Create a component |

* After creating the component node, create a ***cq:dialog***underneath and configure it as follows -

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?> |
|  | <jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0" xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0" xmlns:nt="http://www.jcp.org/jcr/nt/1.0" |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="Text Component 2.0" |
|  | sling:resourceType="cq/gui/components/authoring/dialog"> |
|  | <content |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <layout |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/layouts/tabs" |
|  | type="nav"/> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <tab |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="Properties" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <layout |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/layouts/fixedcolumns"/> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <columns |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <title |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/textfield" |
|  | class="field-whitespace" |
|  | fieldDescription="Enter the title" |
|  | fieldLabel="Title" |
|  | name="./title"/> |
|  | <description |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/textarea" |
|  | class="field-whitespace" |
|  | fieldDescription="Enter the description" |
|  | fieldLabel="Description" |
|  | name="./description"/> |
|  | </items> |
|  | </columns> |
|  | </items> |
|  | </tab> |
|  | </items> |
|  | </content> |
|  | </jcr:root> |

[**view raw**](https://gist.github.com/ani03sha/e9218c77586ef9a8c7b243822ff139f0/raw/0fc16477bd25886206ded7b52502349017d2e160/text2-_content.xml)[**text2-\_content.xml**](https://gist.github.com/ani03sha/e9218c77586ef9a8c7b243822ff139f0#file-text2-_content-xml) hosted with ❤ by [**GitHub**](https://github.com/)

* Now go to the eclipse in the core child project and create a class named ***TextComponent2.java*** and paste the following code in it

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.cqcomponents; |
|  |  |
|  | import javax.jcr.Node; |
|  |  |
|  | import org.redquark.demo.core.models.TextComponent2Model; |
|  | import org.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | import com.adobe.cq.sightly.WCMUsePojo; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* This class has the back-end logic for the Text Component 2.0 |
|  | \*/ |
|  | public class TextComponent2 extends WCMUsePojo { |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(TextComponent2.class); |
|  |  |
|  | /\*\* |
|  | \* Model object |
|  | \*/ |
|  | private TextComponent2Model model; |
|  |  |
|  | /\*\* |
|  | \* Overridden activate method |
|  | \*/ |
|  | @Override |
|  | public void activate() throws Exception { |
|  |  |
|  | try { |
|  |  |
|  | log.info("Text Component 2.0 backend logic starts"); |
|  |  |
|  | /\*\* |
|  | \* Initializing the model object to set the values |
|  | \*/ |
|  | model = new TextComponent2Model(); |
|  |  |
|  | /\*\* |
|  | \* Getting the current node from the resource object which is available in Use |
|  | \* API |
|  | \*/ |
|  | Node node = getResource().adaptTo(Node.class); |
|  |  |
|  | /\*\* |
|  | \* Check if the node has title property |
|  | \*/ |
|  | if(node.hasProperty("title")) { |
|  |  |
|  | /\*\* |
|  | \* Reading the title property from the string |
|  | \*/ |
|  | String title = node.getProperty("title").getString(); |
|  |  |
|  | /\*\* |
|  | \* Setting the value entered by the user in the model object |
|  | \*/ |
|  | model.setTitle(title); |
|  |  |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Check if the node has description property |
|  | \*/ |
|  | if(node.hasProperty("description")) { |
|  |  |
|  | /\*\* |
|  | \* Reading the title property from the string |
|  | \*/ |
|  | String description = node.getProperty("description").getString(); |
|  |  |
|  | /\*\* |
|  | \* Setting the value entered by the user in the model object |
|  | \*/ |
|  | model.setDescription(description); |
|  |  |
|  | } |
|  |  |
|  | } catch (Exception e) { |
|  |  |
|  | log.error(e.getMessage(), e); |
|  | } |
|  | } |
|  |  |
|  | /\*\* |
|  | \* This method is wrapper to return the model object |
|  | \* |
|  | \* @return {@link TextComponent2Model} |
|  | \*/ |
|  | public TextComponent2Model getModel() { |
|  | return model; |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/15bc4f16404bf75a262e566f7ee938e3/raw/0200b7ab0435c51b40409019de7eed32ae143a7e/TextComponent2.java)[**TextComponent2.java**](https://gist.github.com/ani03sha/15bc4f16404bf75a262e566f7ee938e3#file-textcomponent2-java) hosted with ❤ by [**GitHub**](https://github.com/)

This class requires a model class to save the values in memory. For this, I created one simple model class with two properties - title and description (we mimicked the edit dialog properties). Now, create a model class ***TextComponent2Model.java*** and paste the following code in it.

* Let's look at the class TextComponent2.java and understand its code. First, we have extended the class ***WCMsePojo*** to make use of ***Java Use API***.
* Now, in the overridden ***activate()***method, we will write the logic. We are getting the adapted ***Node***object from the ***getResource()***method which is available due to the use of ***WCMUsePojo*** class. We then are checking the existence and getting the properties from the JCR using ***javax.jcr.Node***API. After getting the properties, we are setting them in the model object.
* Now, rename text2.jsp to text2.html and paste the following code in it

|  |  |
| --- | --- |
|  | Text Component 2.0 - Showcases Java Use API |
|  |  |
|  | <div data-sly-use.text2="org.redquark.demo.core.cqcomponents.TextComponent2"> |
|  | <h1>Title: ${text2.model.title}</h1> |
|  | <p>Description: ${text2.model.description}</p> |
|  | </div> |

[**view raw**](https://gist.github.com/ani03sha/2288bc491cb7b11b363aaac96b0f1ad6/raw/8f6cc786b534736fe92815c1ab985b2c65ddebe1/textComponent2.html)[**textComponent2.html**](https://gist.github.com/ani03sha/2288bc491cb7b11b363aaac96b0f1ad6#file-textcomponent2-html) hosted with ❤ by [**GitHub**](https://github.com/)

* Here, we are using ***data-sly-use***to include the Java file (***TextComponent2.java***), then we are accessing the values via the model object.
* Now drag and drop the component on the page and configure it using the edit dialog. You will see the passed values on your page.

# Day 10: Getting to know Sling Models

Wouldn't it be great if you have a direct way to map your JCR node properties to your model class? I guess it would be. In AEM, we can achieve this via [***Sling Models***](https://sling.apache.org/documentation/bundles/models.html).  
Sling Models are "pure" POJOs which maps Sling objects (resources, request objects etc.).  
Since Sling Models are annotation-driven Plain Old Java Objects (POJOs), annotations are used a lot. They allow you to map resource properties, assign default values, inject OSGi services and much more.  
  
Without making further delay, let's see the concepts in action.

**Working with Sling Models**

**For AEM 6.2**

* Download the latest Sling Models API and Implementation bundles from [here](https://sling.apache.org/downloads.cgi) and upload them to .***/system/console/bundles***
* In the project’s ***pom*** files, make sure the version numbers for Sling Models API and Implementation are updated based on the ones installed in your AEM server.
* In your core module ***pom.xml*** file, check for maven-bundle-plugin, and make sure you have all packages that contain the model classes or interfaces in header Sling-Model-Packages, so that your models can be picked up.

**For AEM 6.3+**

* Since AEM 6.3 is built on Sling 1.3, we don't have to install any bundles in our AEM server.
* But we need to make sure that the Sling Models API version in AEM matches with the one in our project's ***pom*** files.

**Example**

Now let us look at an example for sling models 😮

* Create a new AEM project using AEM Multimodule Project (how? see [here](https://aem.redquark.org/2018/10/day-03-setting-up-aem-development.html)) and deploy in your AEM instance.
* Go to CRXDE and under your project in /apps create a new component with the following configuration

|  |
| --- |
|  |
| Create a Component |

* Now create the ***cq:dialog***node under the component node with the following configuration.

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?> |
|  | <jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0" xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0" xmlns:nt="http://www.jcp.org/jcr/nt/1.0" |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="User Component" |
|  | sling:resourceType="cq/gui/components/authoring/dialog"> |
|  | <content |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <layout |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/layouts/tabs" |
|  | type="nav"/> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <properties |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="User Properties" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <layout |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/layouts/fixedcolumns"/> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <columns |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <firstName |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/textfield" |
|  | class="field-whitespace" |
|  | fieldDescription="Please enter the first name" |
|  | fieldLabel="First Name" |
|  | name="./firstName"/> |
|  | <lastName |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/textfield" |
|  | class="field-whitespace" |
|  | fieldDescription="Please enter the last name" |
|  | fieldLabel="Last Name" |
|  | name="./lastName"/> |
|  | <gender |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/select" |
|  | fieldDescription="Select your grnder" |
|  | fieldLabel="gender" |
|  | name="./gender"> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <male |
|  | jcr:primaryType="nt:unstructured" |
|  | text="Male" |
|  | value="Male"/> |
|  | <female |
|  | jcr:primaryType="nt:unstructured" |
|  | text="Female" |
|  | value="Female"/> |
|  | <other |
|  | jcr:primaryType="nt:unstructured" |
|  | text="Other" |
|  | value="Other"/> |
|  | </items> |
|  | </gender> |
|  | <country |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/select" |
|  | fieldDescription="Select your country" |
|  | fieldLabel="Country" |
|  | name="./country"> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <option1 |
|  | jcr:primaryType="nt:unstructured" |
|  | text="India" |
|  | value="India"/> |
|  | <option2 |
|  | jcr:primaryType="nt:unstructured" |
|  | text="Canada" |
|  | value="Canada"/> |
|  | <option3 |
|  | jcr:primaryType="nt:unstructured" |
|  | text="Israel" |
|  | value="Israel"/> |
|  | <option4 |
|  | jcr:primaryType="nt:unstructured" |
|  | text="Other" |
|  | value="Other"/> |
|  | </items> |
|  | </country> |
|  | </items> |
|  | </columns> |
|  | </items> |
|  | </properties> |
|  | </items> |
|  | </content> |
|  | </jcr:root> |

[**view raw**](https://gist.github.com/ani03sha/c56569b8eb3736b423e848f6ed13fc48/raw/d6761365d8b7d193a8c31b7c5f3c1d8a02e28f59/userCompopnent-_content.xml)[**userCompopnent-\_content.xml**](https://gist.github.com/ani03sha/c56569b8eb3736b423e848f6ed13fc48#file-usercompopnent-_content-xml) hosted with ❤ by [**GitHub**](https://github.com/)

* Create a new class UserModel.java in your project and paste the following code into it

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.models; |
|  |  |
|  | import javax.inject.Inject; |
|  |  |
|  | import org.apache.sling.api.resource.Resource; |
|  | import org.apache.sling.models.annotations.Model; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* User Component model class |
|  | \*/ |
|  | @Model(adaptables = Resource.class) |
|  | public class UserModel { |
|  |  |
|  | @Inject |
|  | private String firstName; |
|  |  |
|  | @Inject |
|  | private String lastName; |
|  |  |
|  | @Inject |
|  | private String gender; |
|  |  |
|  | @Inject |
|  | private String country; |
|  |  |
|  | /\*\* |
|  | \* @return the firstName |
|  | \*/ |
|  | public String getFirstName() { |
|  | return firstName; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* @return the lastName |
|  | \*/ |
|  | public String getLastName() { |
|  | return lastName; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* @return the gender |
|  | \*/ |
|  | public String getGender() { |
|  | return gender; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* @return the country |
|  | \*/ |
|  | public String getCountry() { |
|  | return country; |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/774d4b41a093a4a7c49454135c7aecac/raw/b2ee6984184df78a629496940fd3acc0baf5c65a/UserModel.java)[**UserModel.java**](https://gist.github.com/ani03sha/774d4b41a093a4a7c49454135c7aecac#file-usermodel-java) hosted with ❤ by [**GitHub**](https://github.com/)

* Let us understand the code of the class. We are using ***@Model***annotation to specify that this model class is a Sling Model. Each data member is annotated with ***@Inject.***This class will be mapped to a resource in JCR.
* Drag and drop the component on the page and configure it

|  |
| --- |
|  |
| Configure the component |

* After saving the component, the data will save under the page node as shown in the figure

|  |
| --- |
|  |
| Data stored in JCR |

* To access this resource in the JCR, let us create a Sling Servlet (what is it? see [here](https://aem.redquark.org/2018/10/day-05-working-with-sling-servlets-in_10.html)) and paste the following code in it.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.servlets; |
|  |  |
|  | import javax.servlet.Servlet; |
|  |  |
|  | import org.apache.sling.api.SlingHttpServletRequest; |
|  | import org.apache.sling.api.SlingHttpServletResponse; |
|  | import org.apache.sling.api.resource.Resource; |
|  | import org.apache.sling.api.resource.ResourceResolver; |
|  | import org.apache.sling.api.servlets.HttpConstants; |
|  | import org.apache.sling.api.servlets.SlingSafeMethodsServlet; |
|  | import org.osgi.framework.Constants; |
|  | import org.osgi.service.component.annotations.Component; |
|  | import org.redquark.demo.core.models.UserModel; |
|  | import org.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* Servlet to consume the Sling Model |
|  | \*/ |
|  | @Component(service = Servlet.class, property = { Constants.SERVICE\_DESCRIPTION + "=Sling Demo Servlet", |
|  | "sling.servlet.methods=" + HttpConstants.METHOD\_GET, "sling.servlet.paths=" + "/bin/slingmodel/user" }) |
|  | public class SlingModelServlet extends SlingSafeMethodsServlet { |
|  |  |
|  | /\*\* |
|  | \* Generated serialVersionUID |
|  | \*/ |
|  | private static final long serialVersionUID = 7558680464517017317L; |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(SlingModelServlet.class); |
|  |  |
|  | /\*\* |
|  | \* Overridden method |
|  | \*/ |
|  | @Override |
|  | protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) { |
|  |  |
|  | try { |
|  |  |
|  | log.info("----------< Processing starts >----------"); |
|  |  |
|  | /\*\* |
|  | \* Getting the instance of resource resolver |
|  | \*/ |
|  | ResourceResolver resourceResolver = request.getResourceResolver(); |
|  |  |
|  | /\*\* |
|  | \* Getting the resource which has the data stored |
|  | \*/ |
|  | Resource resource = resourceResolver |
|  | .getResource("/content/we-retail/language-masters/en/user/jcr:content/root/responsivegrid/user"); |
|  |  |
|  | /\*\* |
|  | \* Adapting the resource to the UserModel class |
|  | \*/ |
|  | UserModel model = resource.adaptTo(UserModel.class); |
|  |  |
|  | /\*\* |
|  | \* Printing the value stored on the browser window |
|  | \*/ |
|  | response.getWriter() |
|  | .print("Data stored in the resource is:\nFirst Name: " + model.getFirstName() + "\nLast Name: " |
|  | + model.getLastName() + "\nGender: " + model.getGender() + "\nCountry: " |
|  | + model.getCountry()); |
|  |  |
|  | /\*\* |
|  | \* Closing the resource resolver |
|  | \*/ |
|  | resourceResolver.close(); |
|  |  |
|  | } catch (Exception e) { |
|  |  |
|  | log.error(e.getMessage(), e); |
|  | } |
|  |  |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/645cfb92efbdecc483ea9f6bbf9e8caa/raw/1daf3efe96338664bc2d6dd97248f449f434faf8/SlingModelServlet.java)[**SlingModelServlet.java**](https://gist.github.com/ani03sha/645cfb92efbdecc483ea9f6bbf9e8caa#file-slingmodelservlet-java) hosted with ❤ by [**GitHub**](https://github.com/)

* In line #54-55, we are getting the resource and in line #60, we are adapting the resource to our ***UserModel*** class
* Note that we are not setting any data in the model object, we are just calling the getters to access the data stored in the JCR.
* Now, deploy the code in your AEM server (how? see [here](https://aem.redquark.org/2018/10/day-03-setting-up-aem-development.html)) and hit the request - [***http://<host>:<port>/bin/slingmodel/user***](http://localhost:4502/bin/slingmodel/user)you will see the following result.

Data stored in the resource is:

First Name: Anirudh

Last Name: Sharma

Gender: Male

Country: India

* As you see, we can directly access a Sling Resource using Sling Model classes. This saves us a lot of boilerplate code and lets us do separation of concerns.

# Day 11: Client Libraries in Action

Let us say you want to create a new website for your Teddy Bear product. You want this side to be attractive and beautiful along with cool features. Most likely, to achieve this you will have to write lots of CSS and JavaScript in your code.  
  
With the complexity of your website, your code complexity will also increase. With this, it becomes more and more difficult to maintain your JavaScript and CSS code. A website made on top of AEM also faces this issue. To combat this, AEM provides a cool feature called ***Client Libraries.***

## Client Libraries in AEM

Client libraries allow us to manage our client-side code including JS and CSS and provide options to debug, minify, merge and gzip the client-side code.

* Go to your project folder in CRXDE and create a node called ***clientlibs***of type ***cq:ClientLibraryFolder***

|  |
| --- |
|  |
| Create a clientlib |

* Now we need to add the categories for the clientlib. ***Categories*** is the identifier used to directly include this clientlib from a content page or to embed it in other clientlibs. We can categorize our clientlibs using this property to manage files easily and only load whatever we need. The categories property, being multi-valued, allows a library folder to be the part of more than one category. Add the categories property to the ***clientlibs*** node.

***Name: Categories***

***Type: String[]***

***Value: cq.authoring.dialog***

* Create the following structure in clientlibs folder

|  |
| --- |
|  |
| JS and CSS libs |

As you can see, we have created ***js***and ***css*** folders (all JS files will go under ***js*** folder and all CSS files will go under cssfolder) along with two text files ***js.txt*** and ***css.txt***.

* Create a new component in your project folder as below

|  |
| --- |
|  |
| Create a component |

* Create a new node under your component ***cq:dialog*** of type nt***:unstructured***. Now create its children nodes as per the below configuration.

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?> |
|  | <jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0" xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0" xmlns:nt="http://www.jcp.org/jcr/nt/1.0" |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="Client Lib Demo" |
|  | sling:resourceType="cq/gui/components/authoring/dialog"> |
|  | <content |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <layout |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/layouts/tabs" |
|  | type="nav"/> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <tab1 |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="Properties" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <layout |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/layouts/fixedcolumns"/> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <columns |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <imageSrc |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="cq/gui/components/authoring/dialog/fileupload" |
|  | autoStart="{Boolean}false" |
|  | class="cq-droptarget field-bgvalue" |
|  | fieldDescription="Drag &amp; Drop Image" |
|  | fieldLabel="Image" |
|  | fileNameParameter="./imageName" |
|  | fileReferenceParameter="./imageRef" |
|  | id="file-upload-special" |
|  | mimeTypes="[image]" |
|  | multiple="{Boolean}false" |
|  | name="./image" |
|  | title="Image" |
|  | uploadUrl="${suffix.path}" |
|  | useHTML5="{Boolean}true"/> |
|  | <altText |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/textfield" |
|  | class="field-prereq" |
|  | fieldDescription="Please describe the image for screen readers." |
|  | fieldLabel="Alt Text" |
|  | id="alt-special" |
|  | name="./altText" |
|  | validation-error-msg="This field is required when the Image is set." |
|  | validation-prereq=".field-bgvalue img" |
|  | validation-prereq-attr="src"/> |
|  | </items> |
|  | </columns> |
|  | </items> |
|  | </tab1> |
|  | </items> |
|  | </content> |
|  | </jcr:root> |

[**view raw**](https://gist.github.com/ani03sha/dbba49761c310e87609d74527516c7d2/raw/11ad0fae8eaec7919108ec5e987359fbab73644d/clientlib-demo-_content.xml)[**clientlib-demo-\_content.xml**](https://gist.github.com/ani03sha/dbba49761c310e87609d74527516c7d2#file-clientlib-demo-_content-xml) hosted with ❤ by [**GitHub**](https://github.com/)

This component has one fileupload widget and one textfield. Our goal is to put validation on the Alt text field. We want the Alt Text field should be required only if we author the Image in the fileupload widget.

* For the validation, we have to write some JS code. Therefore, create a new file named ***validation.js***in the ***js***folder and paste the following code in it.

|  |  |
| --- | --- |
|  | $(window).adaptTo("foundation-registry").register( |
|  | "foundation.validation.validator", { |
|  | selector : "#alt-special", // validates the specific alt field |
|  | validate : function(el) { |
|  | var $el = $(el); |
|  | var $form = $el.closest('form'); // get the form |
|  | var $upload = $form.find("coral-fileupload[name$=image]"); // find the file upload widget |
|  | if ($upload.hasClass('is-filled') && !$el.val()) { // if class exists, return the validation message |
|  | return "Enter Alt Text"; |
|  | } else { |
|  | return; |
|  | } |
|  | } |
|  | }); |

[**view raw**](https://gist.github.com/ani03sha/abce6188c90acfe26052f908bc0fbd4c/raw/7f284eeb242c04d501d377b82d68e20a6ca788e1/validation.js)[**validation.js**](https://gist.github.com/ani03sha/abce6188c90acfe26052f908bc0fbd4c#file-validation-js) hosted with ❤ by [**GitHub**](https://github.com/)

This code searches the required class and if the class is found, then the validation passes and the edit dialog will be saved, otherwise, it won't.

* Now open the js.txt file and paste the following code in it.

***#base=js***

***validation.js***

* The first line determines the path of the JS files relative to the ***js.txt.***
* The second line determines the JS file be included in the client library. Each JS file we want to include has to be written in a new line.
* Now open the dialog and try to save it after authoring image but the Alt Text field is empty. The dialog will not save and show validation error.

|  |
| --- |
|  |
| Validation error |

* Thus, our client libraries are including. Similarly, we can write CSS and include it in the ***css.txt***.

## Other properties in clientlibs

Apart from the few properties discussed in the above section, we have a few other important properties.

#### dependencies

Let's say clientlibA depends on clientlibB which depends on clientlibC, then on the page which is referring clientlibC, all clientlibA, clientlibB and clientlibC will be included. Thus, ***dependency***property signifies if a one clientlib file depends on others.

#### embed

For the minifying purpose, AEM will merge all the clienlibs in the current one. If clientlibA embeds clientlibB which further embeds clientlibC, then clientlibA will be loaded by embedding clientlibB's code and clientlibC will not be embedded.

## Clientlib configuration

If you navigate to http://<host>:<port>/system/console/configMgr and search for ***Adobe Granite HTML Library Manager*** and open it then you will see it has configurations for ***Minify, Debug***and***GZip***.

|  |
| --- |
|  |
| Adobe Granite HTML Library Manager |

Apart from these basic configurations, we are also provided many other configurations to make clientlibs better.

# Day 12: Creating your custom OSGi Configuration

We all know that AEM works on ***Apache Felix*** which is an implementation of ***OSGi***. OSGi provides a way to manage bundles and configurations. We saw one example of an OSGi configuration in the [previous post](https://aem.redquark.org/2018/10/day-11-client-libraries-in-action.html) where we saw ***Adobe Granite HTML Library Manager***.  
You can find all the Out of the box OSGi configurations at - http://<host>:<port>/system/console/configMgr. Apart from out of the box configurations, we can also create our custom configurations. In this post, we will be creating a custom out of the box configuration which reads the user input and gets the JSON reponse from a web service.

**Code in Action**

* To make a custom OSGi configuration, we need to first create an interface whose public methods will represent the fields in the configuration.
* Create an interface named ***HttpConfiguration***and paste the following code in it.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.services; |
|  |  |
|  | import org.osgi.service.metatype.annotations.AttributeDefinition; |
|  | import org.osgi.service.metatype.annotations.AttributeType; |
|  | import org.osgi.service.metatype.annotations.ObjectClassDefinition; |
|  | import org.osgi.service.metatype.annotations.Option; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* This interface represents an OSGi configuration which can be found at - |
|  | \* ./system/console/configMgr |
|  | \*/ |
|  | @ObjectClassDefinition( |
|  | name = "Http Configuration", |
|  | description = "This configuration reads the values to make an HTTP call to a JSON webservice") |
|  | public @interface HttpConfiguration { |
|  |  |
|  | /\*\* |
|  | \* This is a checkbox property which will indicate of the configuration is |
|  | \* executed or not |
|  | \* |
|  | \* @return {@link Boolean} |
|  | \*/ |
|  | @AttributeDefinition( |
|  | name = "Enable config", |
|  | description = "This property indicates whether the configuration values will taken into account or not", |
|  | type = AttributeType.BOOLEAN) |
|  | public boolean enableConfig(); |
|  |  |
|  | /\*\* |
|  | \* This method returns the protocol that is being used |
|  | \* |
|  | \* @return Protocol |
|  | \*/ |
|  | @AttributeDefinition( |
|  | name = "Protocol", |
|  | description = "Choose Protocol", |
|  | options = { |
|  | @Option(label = "HTTP", value = "http"), @Option(label = "HTTPS", value = "https") }) |
|  | public String getProtocol(); |
|  |  |
|  | /\*\* |
|  | \* Returns the server |
|  | \* |
|  | \* @return {@link String} |
|  | \*/ |
|  | @AttributeDefinition( |
|  | name = "Server", |
|  | description = "Enter the server name") |
|  | public String getServer(); |
|  |  |
|  | /\*\* |
|  | \* Returns the endpoint |
|  | \* |
|  | \* @return {@link String} |
|  | \*/ |
|  | @AttributeDefinition( |
|  | name = "Endpoint", |
|  | description = "Enter the endpoint") |
|  | public String getEndpoint(); |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/7e7b589bc96a307c15824c6789c36ba0/raw/7bca9f63f4c9d97b12c8427a3f845f3c40a1358c/HttpConfiguration.java)[**HttpConfiguration.java**](https://gist.github.com/ani03sha/7e7b589bc96a307c15824c6789c36ba0#file-httpconfiguration-java) hosted with ❤ by [**GitHub**](https://github.com/)

* This configuration has a checkbox ***Enable Config***, a drop-down ***Protocol***, two text fields ***Server*** and ***Endpoint***.
* Now create an interface ***HttpService*** to make the http call as below

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.services; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* This interface exposes the functionality of calling a JSON Web Service |
|  | \*/ |
|  | public interface HttpService { |
|  |  |
|  | /\*\* |
|  | \* This method makes the HTTP call on the given URL |
|  | \* |
|  | \* @param url |
|  | \* @return {@link String} |
|  | \*/ |
|  | public String makeHttpCall(); |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/88f7083d5b028f5afdcfa52b43b92922/raw/023f8fef49958eba42971354997dee9aee758ec9/HttpService.java)[**HttpService.java**](https://gist.github.com/ani03sha/88f7083d5b028f5afdcfa52b43b92922#file-httpservice-java) hosted with ❤ by [**GitHub**](https://github.com/)

* Create an implementation class HttpServiceImpl as below.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.services.impl; |
|  |  |
|  | import org.osgi.service.component.annotations.Activate; |
|  | import org.osgi.service.component.annotations.Component; |
|  | import org.osgi.service.metatype.annotations.Designate; |
|  | import org.redquark.demo.core.services.HttpConfiguration; |
|  | import org.redquark.demo.core.services.HttpService; |
|  | import org.redquark.demo.core.utils.Network; |
|  | import org.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* Implementation class of HttpService interface and this class reads values from the OSGi configuration as well |
|  | \*/ |
|  | @Component(service = HttpService.class, immediate = true) |
|  | @Designate(ocd = HttpConfiguration.class) |
|  | public class HttpServiceImpl implements HttpService { |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(HttpServiceImpl.class); |
|  |  |
|  | /\*\* |
|  | \* Instance of the OSGi configuration class |
|  | \*/ |
|  | private HttpConfiguration configuration; |
|  |  |
|  | @Activate |
|  | protected void activate(HttpConfiguration configuration) { |
|  | this.configuration = configuration; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Overridden method of the HttpService |
|  | \*/ |
|  | @Override |
|  | public String makeHttpCall() { |
|  |  |
|  | log.info("----------< Reading the config values >----------"); |
|  |  |
|  | try { |
|  |  |
|  | /\*\* |
|  | \* Reading values from the configuration |
|  | \*/ |
|  | boolean enable = configuration.enableConfig(); |
|  | String protocol = configuration.getProtocol(); |
|  | String server = configuration.getServer(); |
|  | String endpoint = configuration.getEndpoint(); |
|  |  |
|  | /\*\* |
|  | \* Constructing the URL |
|  | \*/ |
|  | String url = protocol + "://" + server + "/" + endpoint; |
|  |  |
|  | /\*\* |
|  | \* Make HTTP call only if "enable" is true |
|  | \*/ |
|  | if (enable) { |
|  | /\*\* |
|  | \* Making the actual HTTP call |
|  | \*/ |
|  | String response = Network.readJson(url); |
|  |  |
|  | /\*\* |
|  | \* Printing the response in the logs |
|  | \*/ |
|  | log.info("----------< JSON response from the webservice is >----------"); |
|  | log.info(response); |
|  |  |
|  | return response; |
|  |  |
|  | } else { |
|  |  |
|  | log.info("----------< Configuration is not enabled >----------"); |
|  |  |
|  | return "Configuration not enabled"; |
|  | } |
|  |  |
|  | } catch (Exception e) { |
|  |  |
|  | log.error(e.getMessage(), e); |
|  |  |
|  | return "Error occurred" + e.getMessage(); |
|  | } |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/6e92a9c1968c4350a7af2cdecc09ea20/raw/57600bea1b2f89ed54efb4e378a925665a2154f4/HttpServiceImpl.java)[**HttpServiceImpl.java**](https://gist.github.com/ani03sha/6e92a9c1968c4350a7af2cdecc09ea20#file-httpserviceimpl-java) hosted with ❤ by [**GitHub**](https://github.com/)

* This is an OSGi component in which we are reading values from the OSGi configuration. Notice that we are using @Designate annotation to link this class to the configuration.
* Now create a simple Sling Servlet to use this component as follows.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.servlets; |
|  |  |
|  | import javax.servlet.Servlet; |
|  |  |
|  | import org.apache.sling.api.SlingHttpServletRequest; |
|  | import org.apache.sling.api.SlingHttpServletResponse; |
|  | import org.apache.sling.api.servlets.HttpConstants; |
|  | import org.apache.sling.api.servlets.SlingSafeMethodsServlet; |
|  | import org.osgi.framework.Constants; |
|  | import org.osgi.service.component.annotations.Component; |
|  | import org.osgi.service.component.annotations.Reference; |
|  | import org.redquark.demo.core.services.HttpService; |
|  | import org.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* This method makes an HTTP call and read the value from the JSON webservice via an OSGi configuration |
|  | \* |
|  | \*/ |
|  | @Component(service = Servlet.class, property = { Constants.SERVICE\_DESCRIPTION + "=HTTP servlet", |
|  | "sling.servlet.methods=" + HttpConstants.METHOD\_GET, "sling.servlet.paths=" + "/bin/demo/httpcall" }) |
|  | public class HttpServlet extends SlingSafeMethodsServlet { |
|  |  |
|  | /\*\* |
|  | \* Generated serialVersionUid |
|  | \*/ |
|  | private static final long serialVersionUID = -2014397651676211439L; |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(HttpServlet.class); |
|  |  |
|  | @Reference |
|  | private HttpService httpService; |
|  |  |
|  | /\*\* |
|  | \* Overridden doGet() method |
|  | \*/ |
|  | @Override |
|  | protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) { |
|  |  |
|  | try { |
|  |  |
|  | String jsonResponse = httpService.makeHttpCall(); |
|  |  |
|  | /\*\* |
|  | \* Printing the json response on the browser |
|  | \*/ |
|  | response.getWriter().println(jsonResponse); |
|  |  |
|  | } catch (Exception e) { |
|  |  |
|  | log.error(e.getMessage(), e); |
|  | } |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/d48ced6073d0b75799dea2f5ee57f95e/raw/cbc1cc7ec7590eff1f3f10e50347d00c6efbc906/HttpServlet.java)[**HttpServlet.java**](https://gist.github.com/ani03sha/d48ced6073d0b75799dea2f5ee57f95e#file-httpservlet-java) hosted with ❤ by [**GitHub**](https://github.com/)

* Go to the ***./system/console/configMgr*** and search for ***Http Configuration*** and open it and configure it accordingly. Then save.

|  |
| --- |
|  |
| OSGi Configuration |

* Now hit the request - [http://<host>:<port>/bin/demo/httpcall](http://localhost:4502/bin/demo/httpcall) and you will get the JSON response as follows

[{"userId": 1, "id": 1, "title": "delectus aut autem", "completed": false}, {"userId": 1, "id": 2, "title": "quis ut nam facilis et officia qui", "completed": false}, {"userId": 1, "id": 3, "title": "fugiat veniam minus", "completed": false}, {"userId": 1, "id": 4, "title": "et porro tempora", "completed": true}, ...]

# Day 13: Schedulers in AEM

In simple terms, a scheduler is a service to schedule jobs. Thus if we want some work to happen at a particular time or at a regular interval, then [Schedulers](https://sling.apache.org/documentation/bundles/scheduler-service-commons-scheduler.html) are the way to go.  
  
In AEM, we can create scheduler in two ways -

#### Whiteboard Pattern - **In this, we create a Runnable thread to perform our task. This is similar to the Java Thread concept.**

#### Scheduler API - **In this, we use Apache Commons' Scheduler API to perform our task. It uses open-source**[**Quartz**](http://www.quartz-scheduler.org/)**library.**

#### **Both of these methods can be seen**[**here**](https://sling.apache.org/documentation/bundles/scheduler-service-commons-scheduler.html)**. However, from AEM 6.2 onwards Felix SCR annotations are deprecated and are replaced with**[**OSGi R6**](https://osgi.org/javadoc/r6/cmpn/org/osgi/service/component/annotations/package-summary.html)**annotations. Therefore, in this post, we will be discussing the clean way of creating schedulers that**OSGi R6**annotations provide.**

## Scheduler in AEM

To create a scheduler in AEM, we will follow the below steps -

* Create an OSGi configuration to read the scheduler specific values from the user i.e. cron expression, the name of the scheduler, custom parameter etc.
* Create a sling scheduler which displays the custom parameter at an interval specified by the cron expression.

Without making further delay, let's dive into our code

Create an OSGi configuration

Follow this [post](https://aem.redquark.org/2018/10/day-12-creating-your-custom-osgi.html) to create a new OSGi configuration class and paste the following code in it.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.services; |
|  |  |
|  | import org.osgi.service.metatype.annotations.AttributeDefinition; |
|  | import org.osgi.service.metatype.annotations.AttributeType; |
|  | import org.osgi.service.metatype.annotations.ObjectClassDefinition; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* This is the configuration class that takes properties for a scheduler to run |
|  | \*/ |
|  | @ObjectClassDefinition(name = "SlingSchedulerConfiguration", description = "Sling scheduler configuration") |
|  | public @interface SlingSchedulerConfiguration { |
|  |  |
|  | /\*\* |
|  | \* This method will return the name of the Scheduler |
|  | \* |
|  | \* @return {@link String} |
|  | \*/ |
|  | @AttributeDefinition( |
|  | name = "Scheduler name", |
|  | description = "Name of the scheduler", |
|  | type = AttributeType.STRING) |
|  | public String schdulerName() default "Custom Sling Scheduler Configuration"; |
|  |  |
|  | /\*\* |
|  | \* This method will check if the scheduler is concurrent or not |
|  | \* |
|  | \* @return {@link Boolean} |
|  | \*/ |
|  | @AttributeDefinition( |
|  | name = "Enabled", |
|  | description = "True, if scheduler service is enabled", |
|  | type = AttributeType.BOOLEAN) |
|  | public boolean enabled() default false; |
|  |  |
|  | /\*\* |
|  | \* This method returns the Cron expression which will decide how the scheduler will run |
|  | \* |
|  | \* @return {@link String} |
|  | \*/ |
|  | @AttributeDefinition( |
|  | name = "Cron Expression", |
|  | description = "Cron expression used by the scheduler", |
|  | type = AttributeType.STRING) |
|  | public String cronExpression() default "0 \* \* \* \* ?"; |
|  |  |
|  | /\*\* |
|  | \* This method returns a custom parameter just to show case the functionality |
|  | \* |
|  | \* @return {@link String} |
|  | \*/ |
|  | @AttributeDefinition( |
|  | name = "Custom Parameter", |
|  | description = "Custom parameter to be displayed by the scheduler", |
|  | type = AttributeType.STRING) |
|  | public String customParameter() default "AEM Scheduler Demo"; |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/ddd91ade74622f4a83aa35d74b0ac597/raw/7ec860be502d87be368fdb2a5fff3078fac60e3f/SlingSchedulerConfiguration.java)[**SlingSchedulerConfiguration.java**](https://gist.github.com/ani03sha/ddd91ade74622f4a83aa35d74b0ac597#file-slingschedulerconfiguration-java) hosted with ❤ by [**GitHub**](https://github.com/)

Here we are getting configuration properties for the Sling Scheduler.

Create Scheduler

Now we are going to create a sling scheduler. Create a class named ***CustomScheduler*** and paste the following code in it.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.schedulers; |
|  |  |
|  | 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.Modified; |
|  | import org.osgi.service.component.annotations.Reference; |
|  | import org.osgi.service.metatype.annotations.Designate; |
|  | import org.redquark.demo.core.services.SlingSchedulerConfiguration; |
|  | import org.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* A Sling Scheduler demo using OSGi R6 annotations |
|  | \* |
|  | \*/ |
|  | @Component(immediate = true, service = CustomScheduler.class) |
|  | @Designate(ocd = SlingSchedulerConfiguration.class) |
|  | public class CustomScheduler implements Runnable { |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(CustomScheduler.class); |
|  |  |
|  | /\*\* |
|  | \* Custom parameter that is to be read from the configuration |
|  | \*/ |
|  | private String customParameter; |
|  |  |
|  | /\*\* |
|  | \* Id of the scheduler based on its name |
|  | \*/ |
|  | private int schedulerId; |
|  |  |
|  | /\*\* |
|  | \* Scheduler instance injected |
|  | \*/ |
|  | @Reference |
|  | private Scheduler scheduler; |
|  |  |
|  | /\*\* |
|  | \* Activate method to initialize stuff |
|  | \* |
|  | \* @param config |
|  | \*/ |
|  | @Activate |
|  | protected void activate(SlingSchedulerConfiguration config) { |
|  |  |
|  | /\*\* |
|  | \* Getting the scheduler id |
|  | \*/ |
|  | schedulerId = config.schdulerName().hashCode(); |
|  |  |
|  | /\*\* |
|  | \* Getting the custom parameter |
|  | \*/ |
|  | customParameter = config.customParameter(); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Modifies the scheduler id on modification |
|  | \* |
|  | \* @param config |
|  | \*/ |
|  | @Modified |
|  | protected void modified(SlingSchedulerConfiguration config) { |
|  |  |
|  | /\*\* |
|  | \* Removing the scheduler |
|  | \*/ |
|  | removeScheduler(); |
|  |  |
|  | /\*\* |
|  | \* Updating the scheduler id |
|  | \*/ |
|  | schedulerId = config.schdulerName().hashCode(); |
|  |  |
|  | /\*\* |
|  | \* Again adding the scheduler |
|  | \*/ |
|  | addScheduler(config); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* This method deactivates the scheduler and removes it |
|  | \* @param config |
|  | \*/ |
|  | @Deactivate |
|  | protected void deactivate(SlingSchedulerConfiguration config) { |
|  |  |
|  | /\*\* |
|  | \* Removing the scheduler |
|  | \*/ |
|  | removeScheduler(); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* This method removes the scheduler |
|  | \*/ |
|  | private void removeScheduler() { |
|  |  |
|  | log.info("Removing scheduler: {}", schedulerId); |
|  |  |
|  | /\*\* |
|  | \* Unscheduling/removing the scheduler |
|  | \*/ |
|  | scheduler.unschedule(String.valueOf(schedulerId)); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* This method adds the scheduler |
|  | \* |
|  | \* @param config |
|  | \*/ |
|  | private void addScheduler(SlingSchedulerConfiguration config) { |
|  |  |
|  | /\*\* |
|  | \* Check if the scheduler is enabled |
|  | \*/ |
|  | if(config.enabled()) { |
|  |  |
|  | /\*\* |
|  | \* Scheduler option takes the cron expression as a parameter and run accordingly |
|  | \*/ |
|  | ScheduleOptions scheduleOptions = scheduler.EXPR(config.cronExpression()); |
|  |  |
|  | /\*\* |
|  | \* Adding some parameters |
|  | \*/ |
|  | scheduleOptions.name(config.schdulerName()); |
|  | scheduleOptions.canRunConcurrently(false); |
|  |  |
|  | /\*\* |
|  | \* Scheduling the job |
|  | \*/ |
|  | scheduler.schedule(this, scheduleOptions); |
|  |  |
|  | log.info("Scheduler added"); |
|  |  |
|  | } else { |
|  |  |
|  | log.info("Scheduler is disabled"); |
|  |  |
|  | } |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Overridden run method to execute Job |
|  | \*/ |
|  | @Override |
|  | public void run() { |
|  |  |
|  | log.info("Custom Scheduler is now running using the passed custom paratmeter, customParameter {}", customParameter); |
|  |  |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/68411fd4b98bcf113d88b4160301d218/raw/2e5c66431cff9931eecebd580eeadd206417c69b/CustomScheduler.java)[**CustomScheduler.java**](https://gist.github.com/ani03sha/68411fd4b98bcf113d88b4160301d218#file-customscheduler-java) hosted with ❤ by [**GitHub**](https://github.com/)

* Let us understand the code step-by-step. First, we are registering the class as a service and implementing the Runnable interface. At the same time, using ***@Desginate*** annotation, we are linking the OSGi configuration created in the previous section with this class.
* Now, we are injecting the ***org.apache.sling.commons.scheduler.Scheduler***dependency.
* In the activate() method, we are reading the required values. Then we are getting the ***schedulerId*** from the scheduler name.
* The ***modified()*** method recalculates the ***schedulerId*** in case the OSGi configuration is modified.
* In the ***addScheduler()*** method, we are registering the scheduler using the ***Scheduler API.***
* The ***run()*** method will be defining our task. Here we are just printing the ***customParameter*** in the logs.

Configuring OSGi

* Go to ***./system/console/configMgr***and search for ***SlingSchedulerConfiguration***and configure it like below screenshot.

|  |
| --- |
|  |
| OSGi Configuration |

* Save the configuration.

Result

Now go to your project-specific log (my log file is - ***project-demoproject.log***) and look for the traces, you will find like below

org.redquark.demo.core.schedulers.CustomScheduler Scheduler added

org.redquark.demo.core.schedulers.CustomScheduler Custom Scheduler is now running using the passed custom parameter, customParameter AEM Scheduler Demo

org.redquark.demo.core.schedulers.CustomScheduler Custom Scheduler is now running using the passed custom parameter, customParameter AEM Scheduler Demo

org.redquark.demo.core.schedulers.CustomScheduler Removing scheduler: -1760311957

org.redquark.demo.core.schedulers.CustomScheduler Scheduler added

org.redquark.demo.core.schedulers.CustomScheduler Custom Scheduler is now running using the passed custom parameter, customParameter AEM Scheduler Demo

org.redquark.demo.core.schedulers.CustomScheduler Custom Scheduler is now running using the passed custom parameter, customParameter AEM Scheduler Demo

# Day 14: Eventing in AEM

There are various ways by which event handling can be done in AEM -

* **Event Listener -**JCR level events with the observation
* **Event Handler -**Sling level events
* Workflow and Launchers
* Schedulers with cron expressions

In this post, we will be discussing ***Event Handlers*** and ***Event Listeners***. Eventing with ***Workflow and Launchers*** will be discussed in the [next](https://aem.redquark.org/2018/10/day-15-custom-workflows-in-aem.html) post while we have already discussed ***Schedulers***in one of our [previous](https://aem.redquark.org/2018/10/day-13-schedulers-in-aem.html) posts. All the events in AEM can be viewed at - [http://<host>:<port>/system/console/events](http://localhost:4502/system/console/events).

**Event Handlers**

We can create an Event Handler by following below steps -

* Write a service class that implements the ***EventHandler*** interface.
* Register the service with property ***EventConstants.EVENT\_TOPIC.***
* Implement the ***handleEvent(Event event)***method to trigger the job.

Let us create a custom event handler that will be triggered when a page is activated and logs the topic for which the handler was registered.

* Create a class ***CustomEventHandler*** and paste the following code in it.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.listeners; |
|  |  |
|  | import org.apache.sling.api.SlingConstants; |
|  | import org.apache.sling.api.resource.observation.ResourceChangeListener; |
|  | import org.osgi.framework.Constants; |
|  | import org.osgi.service.component.annotations.Component; |
|  | import org.osgi.service.event.Event; |
|  | import org.osgi.service.event.EventConstants; |
|  | import org.osgi.service.event.EventHandler; |
|  | import org.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* Event Handler that listens to the Sling events |
|  | \*/ |
|  | @Component(immediate = true, service = EventHandler.class, property = { |
|  | Constants.SERVICE\_DESCRIPTION + "= This event handler listens the events on page activation", |
|  | EventConstants.EVENT\_TOPIC + "=org/apache/sling/api/resource/Resource/ADDED", |
|  | EventConstants.EVENT\_TOPIC + "=org/apache/sling/api/resource/Resource/CHANGED", |
|  | EventConstants.EVENT\_FILTER + "(&" + "(path=/content/we-retail/us/en/\*/jcr:content) (|(" |
|  | + SlingConstants.PROPERTY\_CHANGED\_ATTRIBUTES + "=\*jcr:title) " + "(" + ResourceChangeListener.CHANGES |
|  | + "=\*jcr:title)))" }) |
|  | public class CustomEventHandler implements EventHandler { |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(CustomEventHandler.class); |
|  |  |
|  | @Override |
|  | public void handleEvent(Event event) { |
|  |  |
|  | log.info("Event is: {}", event.getTopic()); |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/46ac816b924f2f4c5ed5a58f20ea0478/raw/7e26ba506d15dd98b53bb755bc286dbeb65555be/CustomEventHandler.java)[**CustomEventHandler.java**](https://gist.github.com/ani03sha/46ac816b924f2f4c5ed5a58f20ea0478#file-customeventhandler-java) hosted with ❤ by [**GitHub**](https://github.com/)

As you can see that we have service property ***service=EventHandler.class***and EventConstants.EVENT\_TOPIC registered to resource added and changed which normally happens in AEM Replication

* We have implemented the handleEvent() method and it is logging the topic for which this event handler is registered.
* Deploy the code and activate (publish) any page. You will see following traces in the logs

*org.redquark.demo.core.listeners.CustomEventHandler Event is: org/apache/sling/api/resource/Resource/ADDED*

*org.redquark.demo.core.listeners.CustomEventHandler Event is: org/apache/sling/api/resource/Resource/CHANGED*

**Event Listeners**

We can achieve event handling at the JCR level by implementing the ***EventListener*** interface in a class. JCR Observer is the lowest-level event handling in AEM. As its name indicates, it is at the JCR level and allows to listen to JCR-level events, gathered in sets (corresponding to persistence transactions). ***javax.jcr.observation.Event*** lists the following types:

1. Event.NODE\_ADDED
2. Event.NODE\_MOVED
3. Event.NODE\_REMOVED
4. Event.PERSIST
5. Event.PROPERTY\_ADDED
6. Event.PROPERTY\_CHANGED
7. Event.PROPERTY\_REMOVED

Let us create an event listener that listens to the event when a new node is added at a specified path.

* Create a new class ***CustomEventListener***and paste the following code in it

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.listeners; |
|  |  |
|  | import java.util.HashMap; |
|  | import java.util.Map; |
|  |  |
|  | import javax.jcr.Session; |
|  | import javax.jcr.observation.Event; |
|  | import javax.jcr.observation.EventIterator; |
|  | import javax.jcr.observation.EventListener; |
|  |  |
|  | import org.apache.sling.api.resource.ResourceResolver; |
|  | import org.apache.sling.api.resource.ResourceResolverFactory; |
|  | import org.apache.sling.jcr.api.SlingRepository; |
|  | import org.osgi.service.component.ComponentContext; |
|  | 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.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* Event Listener that listens to JCR events |
|  | \*/ |
|  | @Component(service = EventListener.class, immediate = true) |
|  | public class CustomEventListener implements EventListener { |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(CustomEventListener.class); |
|  |  |
|  | /\*\* |
|  | \* Resource Resolver Factory |
|  | \*/ |
|  | @Reference |
|  | private ResourceResolverFactory resolverFactory; |
|  |  |
|  | /\*\* |
|  | \* Resource Resolver |
|  | \*/ |
|  | private ResourceResolver resolver; |
|  |  |
|  | @Reference |
|  | private SlingRepository repository; |
|  |  |
|  | /\*\* |
|  | \* Session object |
|  | \*/ |
|  | private Session session; |
|  |  |
|  | /\*\* |
|  | \* Activate method to initialize stuff |
|  | \*/ |
|  | @Activate |
|  | protected void activate(ComponentContext componentContext) { |
|  |  |
|  | log.info("Activating the observation"); |
|  |  |
|  | try { |
|  |  |
|  | /\*\* |
|  | \* This map will be used to get session via getServiceResourceResolver() method |
|  | \*/ |
|  | Map<String, Object> params = new HashMap<>(); |
|  |  |
|  | /\*\* |
|  | \* Adding the subservice name in the param map |
|  | \*/ |
|  | params.put(ResourceResolverFactory.SUBSERVICE, "eventingService"); |
|  |  |
|  | /\*\* |
|  | \* Getting resource resolver from the service factory |
|  | \*/ |
|  | resolver = resolverFactory.getServiceResourceResolver(params); |
|  |  |
|  | /\*\* |
|  | \* Adapting the resource resolver to session object |
|  | \*/ |
|  | session = resolver.adaptTo(Session.class); |
|  |  |
|  | log.info("Session created"); |
|  |  |
|  | /\*\* |
|  | \* Adding the event listener |
|  | \*/ |
|  | session.getWorkspace().getObservationManager().addEventListener(this, |
|  | Event.PROPERTY\_ADDED | Event.NODE\_ADDED, "/apps/demoproject", true, null, null, false); |
|  |  |
|  | } catch (Exception e) { |
|  |  |
|  | log.error(e.getMessage(), e); |
|  | } |
|  | } |
|  |  |
|  | @Deactivate |
|  | protected void deactivate() { |
|  |  |
|  | if(session != null) { |
|  |  |
|  | session.logout(); |
|  | } |
|  | } |
|  |  |
|  | @Override |
|  | public void onEvent(EventIterator events) { |
|  |  |
|  | try { |
|  |  |
|  | while(events.hasNext()) { |
|  |  |
|  | log.info("Something has been added: {} ", events.nextEvent().getPath() ); |
|  | } |
|  | } catch (Exception e) { |
|  |  |
|  | log.error("Exception occurred", e); |
|  | } |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/210d729c61f229424a33e1f2a026ced9/raw/a3004d92b2d7d0832cea50efc375bceed51983b1/CustomEventListener.java)[**CustomEventListener.java**](https://gist.github.com/ani03sha/210d729c61f229424a33e1f2a026ced9#file-customeventlistener-java) hosted with ❤ by [**GitHub**](https://github.com/)

* First, we need to get the ***javax.jcr.Session***object to add the event listener. Here we are using the concept of [service/system user](https://helpx.adobe.com/in/experience-manager/6-4/sites/administering/using/security-service-users.html) in AEM.
* After getting the session, we are adding the event listener and registering them for ***PROPERTY\_ADDED*** and ***NODE\_ADDED*** events on the path ***/apps/demoproject***.
* In the implemented ***onEvent(EventIterator events)***, we are logging the paths where a new node is added
* Now go to the CRX Explorer and create a system user "demouser" and give all permissions on the root.
* Go to ./system/console/configMgr and search for ***Apache Sling Mapper User Mapper Service***and configure as per the below screenshot

|  |
| --- |
|  |
| User Mapper Service |

* Deploy the code and create new folder at location ***/apps/demoproject***, you will see the following traces in the logs

*org.redquark.demo.core.listeners.CustomEventListener Something has been added: /apps/demoproject/eventTest*

*org.redquark.demo.core.listeners.CustomEventListener Something has been added: /apps/demoproject/eventTest/jcr:primaryType*

*org.redquark.demo.core.listeners.CustomEventListener Something has been added: /apps/demoproject/eventTest/jcr:createdBy*

*org.redquark.demo.core.listeners.CustomEventListener Something has been added: /apps/demoproject/eventTest/jcr:created*

# Day 15: Custom Workflows in AEM

A workflow enables us to automate a wide range of activities in AEM. A workflow consists of steps that are executed in a specific order. Each step performs a distinct activity such as activating a page or sending an email message.  
  
Workflows can interact with assets in the repository, user accounts, and services. Therefore, workflows can coordinate complicated activities that involve any aspect of AEM.  
  
Many useful workflow models are provided out of the box with AEM. In addition, any number of custom workflow models, tailored to the specific needs of our project, can be also be created.  
  
In this post, we will be creating a custom workflow that will read the user passed data and logs it. So, without wasting much time, let's dive into the code.

**Custom Workflow**

* To create a custom workflow component, go to your project folder under ***/apps*** in CRXDE and create a component with the following configuration.

|  |
| --- |
|  |
| Create a component |

* Now create ***cq:dialog***node with the following configuration -

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?> |
|  | <jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0" xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0" xmlns:nt="http://www.jcp.org/jcr/nt/1.0" |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="Custom Workflow" |
|  | sling:resourceType="cq/gui/components/authoring/dialog"> |
|  | <content |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <layout |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/layouts/tabs" |
|  | type="nav"/> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <tab |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="Properties" |
|  | sling:resourceType="granite/ui/components/foundation/section"> |
|  | <layout |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/layouts/fixedcolumns"/> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <columns |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <text |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/textfield" |
|  | fieldDescription="Please enter title" |
|  | fieldLabel="Title" |
|  | name="./metaData/textValue"/> |
|  | <date |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/coral/foundation/form/datepicker" |
|  | class="field" |
|  | fieldDescription="Enter the date" |
|  | fieldLabel="Date" |
|  | name="./metaData/dateValue" |
|  | type="datetime"/> |
|  | </items> |
|  | </columns> |
|  | </items> |
|  | </tab> |
|  | </items> |
|  | </content> |
|  | </jcr:root> |

[**view raw**](https://gist.github.com/ani03sha/273e707f6d7b5a7e40a62284cecbf284/raw/08f10289f8ded868b4a9d1fdc716548116160673/workflow-_content.xml)[**workflow-\_content.xml**](https://gist.github.com/ani03sha/273e707f6d7b5a7e40a62284cecbf284#file-workflow-_content-xml) hosted with ❤ by [**GitHub**](https://github.com/)

* Now create a new node ***cq:ediConfig*** of type ***cq:EditConfig*** underneath ***workflowDemo*** node with the following configuration -

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?> |
|  | <jcr:root xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0" xmlns:nt="http://www.jcp.org/jcr/nt/1.0" |
|  | cq:dialogMode="floating" |
|  | cq:inherit="{Boolean}true" |
|  | jcr:primaryType="cq:EditConfig"> |
|  | <cq:formParameters |
|  | jcr:description="This workflow gets the user input and logs them" |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="Custom Workflow" |
|  | PROCESS="org.redquark.demo.core.workflow.CustomWorkflow" |
|  | PROCESS\_AUTO\_ADVANCE="true"/> |
|  | </jcr:root> |

[**view raw**](https://gist.github.com/ani03sha/1d3a2a632438883b564f9a040381def0/raw/9523778616837d1f7b4af4fcba31393bd8776916/workflow-_cq_editConfig.xml)[**workflow-\_cq\_editConfig.xml**](https://gist.github.com/ani03sha/1d3a2a632438883b564f9a040381def0#file-workflow-_cq_editconfig-xml) hosted with ❤ by [**GitHub**](https://github.com/)

* The PROCESS value is the name of the class that will have Java backend logic.
* Create a java class named ***CustomWorkflow***and paste the following code in it.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.workflows; |
|  |  |
|  | import org.osgi.service.component.annotations.Component; |
|  | import org.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | 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; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* Custom Workflow Step |
|  | \*/ |
|  | @Component(service = WorkflowProcess.class, property = {"process.label = Custom Workflow"}) |
|  | public class CustomWorkflow implements WorkflowProcess { |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(CustomWorkflow.class); |
|  |  |
|  | /\*\* |
|  | \* Overridden method which executes when the workflow is invoked |
|  | \*/ |
|  | @Override |
|  | public void execute(WorkItem workItem, WorkflowSession workflowSession, MetaDataMap metaDataMap) throws WorkflowException { |
|  |  |
|  | log.info("Executing the workflow"); |
|  |  |
|  | try { |
|  |  |
|  | String textValue = metaDataMap.get("textValue", "empty"); |
|  |  |
|  | String dateValue = metaDataMap.get("dateValue", "empty"); |
|  |  |
|  | log.info("Text: {}", textValue); |
|  | log.info("Date: {}", dateValue); |
|  |  |
|  | } catch (Exception e) { |
|  |  |
|  | log.error(e.getMessage(), e); |
|  | } |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/b58e021be77c1093f414fb7392fb187b/raw/eeb3bf4acf3f222bfb6e5263f07a6d7ac8fb36cd/CustomWorkflow.java)[**CustomWorkflow.java**](https://gist.github.com/ani03sha/b58e021be77c1093f414fb7392fb187b#file-customworkflow-java) hosted with ❤ by [**GitHub**](https://github.com/)

* Here we are implementing the ***WorkflowProcess*** interface and defining the ***process.label*** property which defines the name of the workflow step.
* Deploy the code on your AEM server.
* Navigate to [http://<host>:<post>/libs/cq/workflow/admin/console/content/models.html](http://localhost:4502/libs/cq/workflow/admin/console/content/models.html) and click on ***Create*** >> ***Create Model***and enter the Title of the workflow as below and configure it as per the below screenshots

|  |
| --- |
|  |
| Create a workflow model |

|  |
| --- |
|  |
| Edit the workflow model |

|  |
| --- |
|  |
| Add the steps in the workflow |

* Drag and drop the steps from the side rail and double click on the ***Custom Workflow*** step.

|  |
| --- |
|  |
| Add values in the workflow |

* Go to the ***Properties*** tab and enter sample values and save the dialog. Now click on ***Sync.***
* Now go to sites console and run the workflow as per below screenshots.

|  |
| --- |
|  |
| Create a workflow on page |

|  |
| --- |
|  |
| Select Workflow Model |

|  |
| --- |
|  |
| Create |

|  |
| --- |
|  |
| Inbox notifications |

|  |
| --- |
|  |
| Complete the workflow step |

* After completing the workflow, you will see the following traces in logs

*17.10.2018 20:30:51.869 \*INFO\* [JobHandler: /var/workflow/instances/server0/2018-10-17\_1/customWorkflow\_11:/content/we-retail/language-masters/en/women] org.redquark.demo.core.workflows.CustomWorkflow Executing the workflow*

*17.10.2018 20:30:51.874 \*INFO\* [JobHandler: /var/workflow/instances/server0/2018-10-17\_1/customWorkflow\_11:/content/we-retail/language-masters/en/women] org.redquark.demo.core.workflows.CustomWorkflow Text: Custom Workflow Testing Demo*

*17.10.2018 20:30:51.874 \*INFO\* [JobHandler: /var/workflow/instances/server0/2018-10-17\_1/customWorkflow\_11:/content/we-retail/language-masters/en/women] org.redquark.demo.core.workflows.CustomWorkflow Date: 2018-10-17T19:17:00.000+05:30*

# Day 16: Creating JMX Beans in AEM

***Java Management Extensions*** (JMX) is a Java technology that provides an architecture to manage resources dynamically at runtime. [JMX](https://docs.oracle.com/javase/tutorial/jmx/mbeans/index.html) is used mostly in enterprise applications to make the system configurable or to get the state of the application at any point in time. Those resources are represented by objects called ***MBeans*** (for Managed Bean).  
  
In AEM also, sometimes we need to use this architecture to manage resources dynamically. Although AEM provides a rich set of out of the box MBeans, we also create our own MBeans as and when required.  
  
In this post, we will be creating an MBean that will take a set of input parameters and return the status of all the ***bundles/services/components*** in the AEM instance in a JSON format.  
  
Let us look at the steps of creating such an MBean.

**Create an MBean**

* The first step is to create the layout of our MBean. By layout, I mean the input parameters that the MBean requires to function.
* All the parameters are specified by the specified parameters. In our case, we need the server's hostname, port number and the entity which we want to examine (bundles/services/components).
* Create an interface named ***SystemInfo*** and paste the following code in it

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.jmx; |
|  |  |
|  | import com.adobe.granite.jmx.annotation.Description; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* This interface exposes the input parameter for the MBean |
|  | \*/ |
|  | @Description("Input parameters for getting the System information") |
|  | public interface SystemInfo { |
|  |  |
|  | @Description("Enter the protocol, hostname and port of the server") |
|  | String getBundles(String protocol, String hostName, String port); |
|  |  |
|  | @Description("Enter the protocol, hostname and port of the server") |
|  | String getComponents(String protocol, String hostName, String port); |
|  |  |
|  | @Description("Enter the protocol, hostname and port of the server") |
|  | String getServices(String protocol, String hostName, String port); |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/b124317a73f1c6686be8bf3dcfc7e486/raw/b0f735ad8ea59a378e49ca9ec5a3d0a33284e17c/SystemInfo.java)[**SystemInfo.java**](https://gist.github.com/ani03sha/b124317a73f1c6686be8bf3dcfc7e486#file-systeminfo-java) hosted with ❤ by [**GitHub**](https://github.com/)

Here we have three methods that take host, port and protocol and gives us the JSON response of the bundles, services and components

* Now let us create the implementation class of the ***SystemInfo*** interface and paste the following code in it

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.jmx; |
|  |  |
|  | import java.io.BufferedReader; |
|  | import java.io.IOException; |
|  | import java.io.InputStreamReader; |
|  | import java.net.URL; |
|  | import java.net.URLConnection; |
|  |  |
|  | import javax.management.DynamicMBean; |
|  | import javax.management.NotCompliantMBeanException; |
|  |  |
|  | import org.apache.commons.codec.binary.Base64; |
|  | import org.osgi.service.component.annotations.Component; |
|  | import org.redquark.demo.core.utils.Network; |
|  | import org.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | import com.adobe.granite.jmx.annotation.AnnotatedStandardMBean; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* Implementation class for the MBean |
|  | \*/ |
|  | @Component(immediate = true, service = DynamicMBean.class, property = {"jmx.objectname = org.redquark.demo.core.jmx:type=System Info MBean"}) |
|  | public class SystemInfoImpl extends AnnotatedStandardMBean implements SystemInfo { |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(SystemInfoImpl.class); |
|  |  |
|  | /\*\* |
|  | \* Parameters to read user input |
|  | \*/ |
|  |  |
|  | public SystemInfoImpl() throws NotCompliantMBeanException { |
|  | super(SystemInfo.class); |
|  | } |
|  |  |
|  | @Override |
|  | public String getBundles(String protocol, String hostname, String port) { |
|  |  |
|  | log.info("Logging the bundles in JSON"); |
|  |  |
|  | String url = protocol + "://" + hostname + ":" + port + "/system/console/bundles/.json"; |
|  |  |
|  | String result; |
|  |  |
|  | if (protocol.equalsIgnoreCase("http")) { |
|  | result = makeHttpCall(url); |
|  | } else { |
|  | result = Network.readJson(url); |
|  | } |
|  | return result; |
|  | } |
|  |  |
|  | @Override |
|  | public String getComponents(String protocol, String hostname, String port) { |
|  |  |
|  | log.info("Logging the components in JSON"); |
|  |  |
|  | String url = protocol + "://" + hostname + ":" + port + "/system/console/components/.json"; |
|  |  |
|  | String result; |
|  |  |
|  | if (protocol.equalsIgnoreCase("http")) { |
|  | result = makeHttpCall(url); |
|  | } else { |
|  | result = Network.readJson(url); |
|  | } |
|  | return result; |
|  | } |
|  |  |
|  | @Override |
|  | public String getServices(String protocol, String hostname, String port) { |
|  |  |
|  | log.info("Logging the services in JSON"); |
|  |  |
|  | String url = protocol + "://" + hostname + ":" + port + "/system/console/services/.json"; |
|  |  |
|  | String result; |
|  |  |
|  | if (protocol.equalsIgnoreCase("http")) { |
|  | result = makeHttpCall(url); |
|  | } else { |
|  | result = Network.readJson(url); |
|  | } |
|  | return result; |
|  | } |
|  |  |
|  | private String makeHttpCall(String requestURL) { |
|  |  |
|  | URL url; |
|  | try { |
|  | url = new URL(requestURL); |
|  | URLConnection uc; |
|  | uc = url.openConnection(); |
|  |  |
|  | uc.setRequestProperty("X-Requested-With", "Curl"); |
|  |  |
|  | String userpass = "admin" + ":" + "admin"; |
|  | String basicAuth = "Basic " + new String(new Base64().encode(userpass.getBytes())); |
|  | uc.setRequestProperty("Authorization", basicAuth); |
|  |  |
|  | BufferedReader reader = new BufferedReader(new InputStreamReader(uc.getInputStream())); |
|  | StringBuilder builder = new StringBuilder(); |
|  | String line = null; |
|  | while ((line = reader.readLine()) != null) { |
|  | builder.append(line); |
|  | builder.append(System.getProperty("line.separator")); |
|  |  |
|  | } |
|  | String result = builder.toString(); |
|  |  |
|  | return result; |
|  |  |
|  | } catch (IOException e) { |
|  | e.printStackTrace(); |
|  | } |
|  |  |
|  | return ""; |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/6fca1a04ad85d4588b0709288770d0a9/raw/97e8cd01bff013c10d9463d2efe0a9fb7958a7f5/SystemInfoImpl.java)[**SystemInfoImpl.java**](https://gist.github.com/ani03sha/6fca1a04ad85d4588b0709288770d0a9#file-systeminfoimpl-java) hosted with ❤ by [**GitHub**](https://github.com/)

Here we are registering this class of service ***DynamicMBean***which exposes a dynamic management interface. We are using property ***jmx.objectname***which defines the name of our MBean. We are reading the passed parameters and creating an appropriate URL to get the JSON response.

* Now deploy the code and navigate to [http://<host>:<port>/system/console/jmx](http://localhost:4502/system/console/jmx). There you will see all the registered MBean. Search for your MBean.

|  |
| --- |
|  |
| System Info MBean |

* Open the MBean, you will see three methods each for bundles, components and services.
* Open the ***getBundles()*** methods, and configure it. After clicking on ***invoke,***you will see the JSON response returned.

|  |
| --- |
|  |
| JSON Response returned |

# Day 17: Working with QueryBuilder API

Search functionality is the backbone of almost all the enterprise web applications and AEM is no exception. AEM provides the [QueryBuilder](https://helpx.adobe.com/experience-manager/6-3/sites/developing/using/reference-materials/javadoc/com/day/cq/search/QueryBuilder.html) API to perform search operations in the JCR. Query Builder works in the following way -

* Queries are converted into XPath by the Query Engine. There they are again converted into JCR SQL2.
* JCR SQL2 query is executed by the Oak Engine.
* Appropriate Oak Index will be picked up by the query. A query for which the index is not found becomes the traversal queries (traverses the whole repository) which should be avoided by creating an appropriate index.
* Query Builder does not have its own index. It uses Oak index.

**Query Builder**

A query builder accepts a query description (in the form of predicates), creates the query and searches the JCR based on the predicates passed. Without further ado, let's see the code.

Here we are creating the Sling Servlet that will use the Query Builder to search all the assets under /content/dam matching to the search term passed.

* Create a class named QueryBuilderServlet and paste the following code in it.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.servlets; |
|  |  |
|  | import java.util.HashMap; |
|  | import java.util.Map; |
|  |  |
|  | import javax.jcr.Session; |
|  | import javax.servlet.Servlet; |
|  |  |
|  | import org.apache.sling.api.SlingHttpServletRequest; |
|  | import org.apache.sling.api.SlingHttpServletResponse; |
|  | import org.apache.sling.api.resource.ResourceResolver; |
|  | import org.apache.sling.api.servlets.HttpConstants; |
|  | import org.apache.sling.api.servlets.SlingSafeMethodsServlet; |
|  | import org.osgi.framework.Constants; |
|  | import org.osgi.service.component.annotations.Component; |
|  | import org.osgi.service.component.annotations.Reference; |
|  | import org.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | import com.day.cq.search.PredicateGroup; |
|  | import com.day.cq.search.Query; |
|  | import com.day.cq.search.QueryBuilder; |
|  | import com.day.cq.search.result.Hit; |
|  | import com.day.cq.search.result.SearchResult; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* This servlet uses the QueryBuilder API to fetch the results from the JCR |
|  | \*/ |
|  | @Component(service = Servlet.class, property = { Constants.SERVICE\_DESCRIPTION + "=Query Builder servlet", |
|  | "sling.servlet.methods=" + HttpConstants.METHOD\_GET, "sling.servlet.paths=" + "/bin/demo/querybuilder" }) |
|  | public class QueryBuilderServlet extends SlingSafeMethodsServlet { |
|  |  |
|  | /\*\* |
|  | \* Generated serialVersionUID |
|  | \*/ |
|  | private static final long serialVersionUID = 2610051404257637265L; |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(QueryBuilderServlet.class); |
|  |  |
|  | /\*\* |
|  | \* Injecting the QueryBuilder dependency |
|  | \*/ |
|  | @Reference |
|  | private QueryBuilder builder; |
|  |  |
|  | /\*\* |
|  | \* Session object |
|  | \*/ |
|  | private Session session; |
|  |  |
|  | /\*\* |
|  | \* Overridden doGet() method which executes on HTTP GET request |
|  | \*/ |
|  | @Override |
|  | protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) { |
|  |  |
|  | try { |
|  |  |
|  | log.info("----------< Executing Query Builder Servlet >----------"); |
|  |  |
|  | /\*\* |
|  | \* This parameter is passed in the HTTP call |
|  | \*/ |
|  | String param = request.getParameter("param"); |
|  |  |
|  | log.info("Search term is: {}", param); |
|  |  |
|  | /\*\* |
|  | \* Get resource resolver instance |
|  | \*/ |
|  | ResourceResolver resourceResolver = request.getResourceResolver(); |
|  |  |
|  | /\*\* |
|  | \* Adapting the resource resolver to the session object |
|  | \*/ |
|  | session = resourceResolver.adaptTo(Session.class); |
|  |  |
|  | /\*\* |
|  | \* Map for the predicates |
|  | \*/ |
|  | Map<String, String> predicate = new HashMap<>(); |
|  |  |
|  | /\*\* |
|  | \* Configuring the Map for the predicate |
|  | \*/ |
|  | predicate.put("path", "/content/dam"); |
|  | predicate.put("type", "dam:Asset"); |
|  | predicate.put("group.p.or", "true"); |
|  | predicate.put("group.1\_fulltext", param); |
|  | predicate.put("group.1\_fulltext.relPath", "jcr:content"); |
|  |  |
|  | /\*\* |
|  | \* Creating the Query instance |
|  | \*/ |
|  | Query query = builder.createQuery(PredicateGroup.create(predicate), session); |
|  |  |
|  | query.setStart(0); |
|  | query.setHitsPerPage(20); |
|  |  |
|  | /\*\* |
|  | \* Getting the search results |
|  | \*/ |
|  | SearchResult searchResult = query.getResult(); |
|  |  |
|  | for(Hit hit : searchResult.getHits()) { |
|  |  |
|  | String path = hit.getPath(); |
|  |  |
|  | response.getWriter().println(path); |
|  | } |
|  | } catch (Exception e) { |
|  |  |
|  | log.error(e.getMessage(), e); |
|  | } finally { |
|  |  |
|  | if(session != null) { |
|  |  |
|  | session.logout(); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/43d1b954487aca6a2bd2fe518d121b59/raw/36d9b3d1804b7c013f87600c6b9b0f625a93f678/QueryBuilderServlet.java)[**QueryBuilderServlet.java**](https://gist.github.com/ani03sha/43d1b954487aca6a2bd2fe518d121b59#file-querybuilderservlet-java) hosted with ❤ by [**GitHub**](https://github.com/)

* Here, we are reading the query parameter ***param*** and based on this query parameter we will search the JCR in the path ***/content/dam***
* We are searching only the assets, which is defined by the property ***type=dam:Asset***
* Now we are creating the ***Query*** object using the ***predicate*** map and the ***session*** object.
* At last, we are getting the ***SearchResult*** from the query builder and then using the ***Hit*** object, we are getting the path of each result.
* Deploy the code and executing the servlet by hitting the request - [http://<host>:<port>/bin/demo/querybuilder?param=we.retail](http://localhost:4502/bin/demo/querybuilder?param=we.retail).
* You will get the following the result set on the browser window

***1. /content/dam/we-retail-screens/we-retail-instore-logo.png***

***2. /content/dam/projects/we-retail/cover***

***3. /content/dam/we-retail-screens/italy.png***

***4. /content/dam/we-retail-screens/usa.png***

***5. /content/dam/we-retail/en/activities/running/marathon-shoes.jpg***

***6. /content/dam/we-retail/en/activities/running/running-woods-woman.jpg***

***7. /content/dam/we-retail/en/activities/running/running\_4.jpg***

***8. /content/dam/we-retail/en/activities/running/fitness-woman.jpg***

***9. /content/dam/we-retail/en/activities/running/running\_1.jpg***

***10. /content/dam/we-retail/en/activities/running/running\_3.jpg***

***11. /content/dam/we-retail/en/activities/running/running\_5.jpg***

***12. /content/dam/we-retail/en/activities/running/running\_2.jpg***

***13. /content/dam/we-retail/en/activities/running/running-desert-woman.jpg***

***14. /content/dam/we-retail/en/activities/running/running-couple-mountain.jpg***

***15. /content/dam/we-retail/en/activities/snowboarding/man-snowboarder.jpg***

***16. /content/dam/we-retail/en/activities/snowboarding/extreme-championship-girls.jpg***

***17. /content/dam/we-retail/en/activities/snowboarding/freeride-downhill.jpg***

***18. /content/dam/we-retail/en/activities/hiking-camping/alpinists-rochefort-ridge.jpg***

***19. /content/dam/we-retail/en/activities/hiking-camping/hiker-anapurna.jpg***

***20. /content/dam/we-retail/en/activities/hiking-camping/trekker-ama-dablam.jpg***

# Day 18: Working with Granite Datasources in AEM

In AEM development, you might have requirements to populate the same data at multiple places. For e.g., suppose you have a requirement to create a text, an image and a video component and you need to provide background colour to these components. The background colours are the same throughout your application code.  
A naive way to achieve this is to create the same colour nodes under all the component's ***cq:dialog***. Here we are not getting any code reusability. Wouldn't it be great if we create all the colour nodes once and reuse it anywhere we want? [Granite Datasource](https://helpx.adobe.com/experience-manager/6-3/sites/developing/using/reference-materials/granite-ui/api/jcr_root/libs/granite/ui/docs/server/datasource.html) is the answer to this question.  
  
A datasource is the factory to provide a collection of [Resource](http://sling.apache.org/apidocs/sling8/org/apache/sling/api/resource/Resource.html) and normally used to dynamically populate data in the dialog fields such as dropdowns.  
In this post, we are going to create a dynamic datasource and we will use it to populate a dropdown in our component.

**Create a Datasource**

* Navigate to CRXDE and create a component with the following configuration

|  |
| --- |
|  |
| Create a component |

* No create a node ***cq:dialog*** of type ***nt:unstructured*** under this component with the following configuration

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?> |
|  | <jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0" xmlns:cq="http://www.day.com/jcr/cq/1.0" xmlns:jcr="http://www.jcp.org/jcr/1.0" xmlns:nt="http://www.jcp.org/jcr/nt/1.0" |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="Datasource Demo" |
|  | sling:resourceType="cq/gui/components/authoring/dialog"> |
|  | <content |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <layout |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/layouts/tabs" |
|  | type="nav"/> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <tab |
|  | jcr:primaryType="nt:unstructured" |
|  | jcr:title="Properties" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <layout |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/layouts/fixedcolumns"/> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <columns |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/container"> |
|  | <items jcr:primaryType="nt:unstructured"> |
|  | <title |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/textfield" |
|  | class="field-whitespace" |
|  | fieldDescription="Enter the title" |
|  | fieldLabel="Title" |
|  | name="./title"/> |
|  | <colors |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="granite/ui/components/foundation/form/select" |
|  | fieldDescription="Select a color" |
|  | fieldLabel="Color" |
|  | name="./color"> |
|  | <datasource |
|  | jcr:primaryType="nt:unstructured" |
|  | sling:resourceType="/apps/demoproject/components/datasource/lists/datalist.html" |
|  | data\_path="colors"/> |
|  | </colors> |
|  | </items> |
|  | </columns> |
|  | </items> |
|  | </tab> |
|  | </items> |
|  | </content> |
|  | </jcr:root> |

[**view raw**](https://gist.github.com/ani03sha/b9a93fe4a7db772632c1f152d2732b43/raw/38731ce2458dc5350ebb60a9732651ac56c80cfc/datasourceDemo-_content.xml)[**datasourceDemo-\_content.xml**](https://gist.github.com/ani03sha/b9a93fe4a7db772632c1f152d2732b43#file-datasourcedemo-_content-xml) hosted with ❤ by [**GitHub**](https://github.com/)

* Note that in the ***/apps/demoproject/components/content/datasourceDemo/cq:dialog/content/items/tab/items/columns/items/colors/datasource*** we have ***datasource*** node whose ***sling:resourceType*** property is equal to the path of the HTML file where we have written the code of calling Java class.
* Create the following structure in your project folder

|  |
| --- |
|  |
| Datasource and Data nodes |

* ***/apps/demoproject/components/datasource*** node has the HTML file in which we are calling the Java backend class
* While ***/apps/demoproject/components/common/data*** node has the nodes which represent data. In our case, we have colour nodes. Note the folders ***common*** and ***data*** are of type ***sling:Folder.***
* Add following code in the ***datalist.html*** file

***<sly data-sly-use.data="org.redquark.demo.core.datasource.Datasource" />***

* Create a class named ***Datasource*** and paste the following code in it.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.datasource; |
|  |  |
|  | import java.util.HashMap; |
|  | import java.util.LinkedHashMap; |
|  | import java.util.Map; |
|  |  |
|  | import javax.jcr.Node; |
|  | import javax.jcr.NodeIterator; |
|  |  |
|  | import org.apache.commons.collections4.Transformer; |
|  | import org.apache.commons.collections4.iterators.TransformIterator; |
|  | import org.apache.sling.api.resource.Resource; |
|  | import org.apache.sling.api.resource.ResourceMetadata; |
|  | import org.apache.sling.api.resource.ResourceResolver; |
|  | import org.apache.sling.api.resource.ResourceUtil; |
|  | import org.apache.sling.api.resource.ValueMap; |
|  | import org.apache.sling.api.wrappers.ValueMapDecorator; |
|  | import org.osgi.service.component.annotations.Activate; |
|  | import org.redquark.demo.core.constants.AppConstants; |
|  | import org.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | import com.adobe.cq.sightly.WCMUsePojo; |
|  | import com.adobe.granite.ui.components.ds.DataSource; |
|  | import com.adobe.granite.ui.components.ds.SimpleDataSource; |
|  | import com.adobe.granite.ui.components.ds.ValueMapResource; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \*/ |
|  | public class Datasource extends WCMUsePojo { |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(Datasource.class); |
|  |  |
|  | @Activate |
|  | public void activate() throws Exception { |
|  |  |
|  | try { |
|  |  |
|  | final ResourceResolver resolver = getResourceResolver(); |
|  |  |
|  | String dataPath = ResourceUtil.getValueMap(getResource().getChild("datasource")).get("data\_path", String.class); |
|  |  |
|  | log.info("Data path is: {}", dataPath); |
|  |  |
|  | Resource resource = resolver.getResource(AppConstants.DATASOURCE\_PATH + dataPath); |
|  |  |
|  | log.info("Resource: {}", resource); |
|  |  |
|  | Map<String, String> data = new LinkedHashMap<>(); |
|  |  |
|  | Node currentNode = resource.adaptTo(Node.class); |
|  |  |
|  | NodeIterator nodeIterator = currentNode.getNodes(); |
|  |  |
|  | while(nodeIterator.hasNext()) { |
|  |  |
|  | Node node = nodeIterator.nextNode(); |
|  |  |
|  | if (!node.hasProperty("value")) { |
|  | data.put(node.getName(), node.getProperty("name").getString()); |
|  | } else if (node.hasProperty("name")) { |
|  | data.put(node.getProperty("value").getValue().getString(), |
|  | node.getProperty("name").getValue().getString()); |
|  | } else { |
|  | data.put(node.getProperty("value").getValue().getString(), node.getName()); |
|  | } |
|  | } |
|  |  |
|  | @SuppressWarnings({ "unchecked", "rawtypes" }) |
|  | DataSource ds = new SimpleDataSource(new TransformIterator<>(data.keySet().iterator(), new Transformer() { |
|  |  |
|  | @Override |
|  | public Object transform(Object o) { |
|  |  |
|  | String dropValue = (String) o; |
|  |  |
|  | ValueMap vm = new ValueMapDecorator(new HashMap<>()); |
|  |  |
|  | vm.put("value", dropValue); |
|  | vm.put("text", data.get(dropValue)); |
|  |  |
|  | return new ValueMapResource(resolver, new ResourceMetadata(), "nt:unstructured", vm); |
|  | }; |
|  | })); |
|  |  |
|  | getRequest().setAttribute(DataSource.class.getName(), ds); |
|  |  |
|  | } catch (Exception e) { |
|  |  |
|  | log.error(e.getMessage(), e); |
|  | } |
|  |  |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/0bb2b8a198270f77b3eed901e4cfcc40/raw/50d5a7c5691f3a2a21398f5bb7bb975507f9b839/Datasource.java)[**Datasource.java**](https://gist.github.com/ani03sha/0bb2b8a198270f77b3eed901e4cfcc40#file-datasource-java) hosted with ❤ by [**GitHub**](https://github.com/)

* Here ***datapath*** is the value of the property ***data\_path*** stored under the ***colors*** node of the component's ***cq:dialog.***In our case, it will read as "colors".
* Then we are the resource object of the node ***/apps/demoproject/components/common/data/colors***
* Then we are storing the ***colors*** in the ***Map<String, String>*** via node iteration.
* Then comes the most important logic of creating ***com.adobe.granite.ui.components.ds.DataSource*** object. Here we are converting the Map created previously into the ***DataSource*** object ***ds.***
* After setting the ***ds***, we are setting it in the request object.
* Now go to any page of your website and drag & drop the ***datasourceDemo***component and open the dialog. You will then see the colours are populated in the dropdown.

|  |
| --- |
|  |
| Edit dialog |

# Day 19: Replication API in Action

Replication is an important concept in AEM as it enables an enterprise to author the content in the Author AEM instance and after it is final replicate to the Publish instance which is visible to the end-users of the website.  
  
AEM provides a rich [Replication API](https://helpx.adobe.com/experience-manager/6-4/sites/developing/using/reference-materials/javadoc/com/day/cq/replication/package-summary.html) which replicates the content from AEM Author to the AEM Publish instance. Though AEM provides out of the box Replication agents which are super easy to configure and provides much abstraction for the replication process, sometimes we are required to achieve this functionality programmatically such as when asset metadata is changed, or a page is versioned any many other cases.  
  
In this post, we will see how can we replicate content programmatically. For this, we will be creating a Sling Servlet (how? see [here](https://aem.redquark.org/2018/10/day-05-working-with-sling-servlets-in_10.html)) and passing the content path which we want to replicate as a query parameter. So without wasting much time, let's get our hands dirty with some code.

**Replication by code**

For using the Replication API, we will be using an API ***com.day.cq.replication.Replicator***to replicate the content.

* Create a class named ***ReplicationServlet*** and paste the following code in it.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.servlets; |
|  |  |
|  | import java.util.LinkedHashSet; |
|  | import java.util.Map; |
|  | import java.util.Set; |
|  |  |
|  | import javax.jcr.Node; |
|  | import javax.jcr.Session; |
|  | import javax.servlet.Servlet; |
|  |  |
|  | import org.apache.sling.api.SlingHttpServletRequest; |
|  | import org.apache.sling.api.SlingHttpServletResponse; |
|  | import org.apache.sling.api.resource.Resource; |
|  | import org.apache.sling.api.resource.ResourceResolver; |
|  | import org.apache.sling.api.servlets.HttpConstants; |
|  | import org.apache.sling.api.servlets.SlingSafeMethodsServlet; |
|  | import org.osgi.framework.Constants; |
|  | import org.osgi.service.component.annotations.Component; |
|  | import org.osgi.service.component.annotations.Reference; |
|  | import org.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | import com.day.cq.dam.api.Asset; |
|  | import com.day.cq.dam.api.DamConstants; |
|  | import com.day.cq.dam.commons.util.AssetReferenceSearch; |
|  | import com.day.cq.replication.ReplicationActionType; |
|  | import com.day.cq.replication.ReplicationException; |
|  | import com.day.cq.replication.Replicator; |
|  | import com.day.cq.wcm.api.Page; |
|  | import com.day.cq.wcm.api.PageManager; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \*/ |
|  | @Component(service = Servlet.class, property = { |
|  | Constants.SERVICE\_DESCRIPTION + "= Servlet to replicate the passed contents", |
|  | "sling.servlet.methods=" + HttpConstants.METHOD\_GET, "sling.servlet.paths=" + "/bin/demo/replication" }) |
|  | public class ReplicationServlet extends SlingSafeMethodsServlet { |
|  |  |
|  | /\*\* |
|  | \* Generated serialVersionUID |
|  | \*/ |
|  | private static final long serialVersionUID = 1860103923125331408L; |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(ReplicationServlet.class); |
|  |  |
|  | @Reference |
|  | private Replicator replicator; |
|  |  |
|  | @Override |
|  | protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) { |
|  |  |
|  | log.info("----------< Processing starts >----------"); |
|  |  |
|  | try { |
|  |  |
|  | String path = request.getParameter("contentPath"); |
|  |  |
|  | ResourceResolver resolver = request.getResourceResolver(); |
|  |  |
|  | Session session = resolver.adaptTo(Session.class); |
|  |  |
|  | replicateContent(session, path); |
|  |  |
|  | activatePageAssets(resolver, path); |
|  |  |
|  | log.info("----------< Processing ends >----------"); |
|  |  |
|  | response.getWriter().println("Replicated content"); |
|  |  |
|  | } catch (Exception e) { |
|  |  |
|  | log.error(e.getMessage(), e); |
|  | } |
|  | } |
|  |  |
|  | private void activatePageAssets(ResourceResolver resolver, String path) { |
|  |  |
|  | Set<String> pageAssetPaths = getPageAssetsPaths(resolver, path); |
|  |  |
|  | if (pageAssetPaths == null) { |
|  |  |
|  | return; |
|  | } |
|  |  |
|  | Session session = resolver.adaptTo(Session.class); |
|  |  |
|  | for (String assetPath : pageAssetPaths) { |
|  | replicateContent(session, assetPath); |
|  | } |
|  | } |
|  |  |
|  | private Set<String> getPageAssetsPaths(ResourceResolver resolver, String pagePath) { |
|  |  |
|  | PageManager pageManager = resolver.adaptTo(PageManager.class); |
|  |  |
|  | Page page = pageManager.getPage(pagePath); |
|  |  |
|  | if (page == null) { |
|  | return new LinkedHashSet<>(); |
|  | } |
|  |  |
|  | Resource resource = page.getContentResource(); |
|  | AssetReferenceSearch assetReferenceSearch = new AssetReferenceSearch(resource.adaptTo(Node.class), |
|  | DamConstants.MOUNTPOINT\_ASSETS, resolver); |
|  | Map<String, Asset> assetMap = assetReferenceSearch.search(); |
|  |  |
|  | return assetMap.keySet(); |
|  | } |
|  |  |
|  | private void replicateContent(Session session, String path) { |
|  |  |
|  | try { |
|  | replicator.replicate(session, ReplicationActionType.ACTIVATE, path); |
|  | log.info("Replicated: {}", path); |
|  | } catch (ReplicationException e) { |
|  | log.error(e.getMessage(), e); |
|  | e.printStackTrace(); |
|  | } |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/0afaf6b3f5750380d38bcf06b2d1387c/raw/e24481ce25363d2e366242a3738181e909e97c5d/ReplicationServlet.java)[**ReplicationServlet.java**](https://gist.github.com/ani03sha/0afaf6b3f5750380d38bcf06b2d1387c#file-replicationservlet-java) hosted with ❤ by [**GitHub**](https://github.com/)

* First, we are getting the path of the path to be replicated as a request parameter.
* All we need to do is to define a field of the type ***com.day.cq.replication.Replicator***, let OSGi injects the reference to Replicator service.
* The code is pretty simple as we are calling the replicate(Session session, ReplicationActionType type, String path) to replicate the content to the publish instance.
* But wait a minute? What about the assets that are referenced on the page? How do we replicate them? 🤔 We have an awesome class ***com.day.cq.dam.commons.util.AssetReferenceSearch***that will get the reference of the Assets and replicate them as well. Pretty cool stuff, eh? 🙋.
* We pass page paths and assets paths and they are replicated one by one.
* After deploying the code, go to the browser and hit [http://<host>:<port>/bin/demo/replication?contentPath=/content/we-retail/language-masters/en/user](http://localhost:4502/bin/demo/replication?contentPath=/content/we-retail/language-masters/en/user) and see the content on the publish.
* The logs will look like this -

2018-10-19 22:34:10 - INFO - publish : Creating content for path /content/we-retail/language-masters/en/user  
2018-10-19 22:34:11 - INFO - publish : Creating content for path /content/dam/we-retail/en/experiences/arctic-surfing-in-lofoten/northern-lights.jpg  
2018-10-19 22:34:11 - INFO - publish : Sending POST request to http://localhost:4503/bin/receive?sling:authRequestLogin=1  
2018-10-19 22:34:11 - INFO - publish : sent. Response: 200 OK  
2018-10-19 22:34:11 - INFO - publish : ------------------------------------------------  
2018-10-19 22:34:11 - INFO - publish : Sending message to localhost:4503  
2018-10-19 22:34:11 - INFO - publish : >> POST /bin/receive HTTP/1.0  
2018-10-19 22:34:11 - INFO - publish : >> Action: Activate  
2018-10-19 22:34:11 - INFO - publish : >> Path: /content/we-retail/language-masters/en/user  
2018-10-19 22:34:11 - INFO - publish : >> Handle: /content/we-retail/language-masters/en/user  
2018-10-19 22:34:11 - INFO - publish : >> Referer: about:blank  
2018-10-19 22:34:11 - INFO - publish : >> ...spooling 2606 bytes...  
2018-10-19 22:34:11 - INFO - publish : --  
2018-10-19 22:34:11 - INFO - publish : << HTTP/1.1 200 OK  
2018-10-19 22:34:11 - INFO - publish : << Date: Fri, 19 Oct 2018 17:04:11 GMT  
2018-10-19 22:34:11 - INFO - publish : << X-Content-Type-Options: nosniff  
2018-10-19 22:34:11 - INFO - publish : << Content-Type: text/plain;charset=utf-8  
2018-10-19 22:34:11 - INFO - publish : << Content-Length: 30  
2018-10-19 22:34:11 - INFO - publish : <<   
2018-10-19 22:34:11 - INFO - publish : << ReplicationAction ACTIVATE ok.  
2018-10-19 22:34:11 - INFO - publish : Message sent.  
2018-10-19 22:34:11 - INFO - publish : ------------------------------------------------  
2018-10-19 22:34:11 - INFO - publish : Replication (ACTIVATE) of /content/we-retail/language-masters/en/user successful.  
2018-10-19 22:34:11 - INFO - publish : Sending POST request to http://localhost:4503/bin/receive?sling:authRequestLogin=1  
2018-10-19 22:34:11 - INFO - publish : sent. Response: 200 OK  
2018-10-19 22:34:11 - INFO - publish : ------------------------------------------------  
2018-10-19 22:34:11 - INFO - publish : Sending message to localhost:4503  
2018-10-19 22:34:11 - INFO - publish : >> POST /bin/receive HTTP/1.0  
2018-10-19 22:34:11 - INFO - publish : >> Action: Activate  
2018-10-19 22:34:11 - INFO - publish : >> Path: /content/dam/we-retail/en/experiences/arctic-surfing-in-lofoten/northern-lights.jpg  
2018-10-19 22:34:11 - INFO - publish : >> Handle: /content/dam/we-retail/en/experiences/arctic-surfing-in-lofoten/northern-lights.jpg  
2018-10-19 22:34:11 - INFO - publish : >> Referer: about:blank  
2018-10-19 22:34:11 - INFO - publish : >> ...spooling 568041 bytes...  
2018-10-19 22:34:11 - INFO - publish : --  
2018-10-19 22:34:11 - INFO - publish : << HTTP/1.1 200 OK  
2018-10-19 22:34:11 - INFO - publish : << Date: Fri, 19 Oct 2018 17:04:11 GMT  
2018-10-19 22:34:11 - INFO - publish : << X-Content-Type-Options: nosniff  
2018-10-19 22:34:11 - INFO - publish : << Content-Type: text/plain;charset=utf-8  
2018-10-19 22:34:11 - INFO - publish : << Content-Length: 30  
2018-10-19 22:34:11 - INFO - publish : <<   
2018-10-19 22:34:11 - INFO - publish : << ReplicationAction ACTIVATE ok.  
2018-10-19 22:34:11 - INFO - publish : Message sent.  
2018-10-19 22:34:11 - INFO - publish : ------------------------------------------------  
2018-10-19 22:34:11 - INFO - publish : Replication (ACTIVATE) of /content/dam/we-retail/en/experiences/arctic-surfing-in-lofoten/northern-lights.jpg successful.

# Day 20: Working with Users and Groups in AEM

Hello everyone, welcome to the last day of this basic developer series. Today we are going to look into the Users/Groups API that can be used to get the information about all the users and groups present in the JCR.  
  
For this, we will be using ***org.apache.jackrabbit.api.security.user.UserManager*** API which looks for all the users and groups in the JCR.

**User Component**

To list all the users and groups on the page, we will be creating a ***cq:Component*** and it will search in the JCR to get all the data.

* Create component User with the following configuration

|  |
| --- |
|  |
| User Component |

* Rename the ***user.jsp*** to ***user.html*** and paste the following code in it.

|  |  |
| --- | --- |
|  | <h1>Users/Groups in the AEM JCR are : </h1> |
|  | <sly |
|  | data-sly-use.object="org.redquark.demo.core.cqcomponents.UserComponent" /> |
|  | <div> |
|  | <h2>Users</h2> |
|  | <ol data-sly-list="${object.users}"> |
|  | <li>${item}</li> |
|  | </ol> |
|  | <h2>Groups</h2> |
|  | <ol data-sly-list="${object.groups}"> |
|  | <li>${item}</li> |
|  | </ol> |
|  | </div> |

[**view raw**](https://gist.github.com/ani03sha/c31204ae97910b980af164ba1cba72aa/raw/0fb5747b4be8c692a9928033510d5899aaefdf9e/user.html)[**user.html**](https://gist.github.com/ani03sha/c31204ae97910b980af164ba1cba72aa#file-user-html) hosted with ❤ by [**GitHub**](https://github.com/)

* Here we are iterating the list of users and groups and displaying their IDs.
* Now create a class named UserComponent and paste the following code in it.

|  |  |
| --- | --- |
|  | package org.redquark.demo.core.cqcomponents; |
|  |  |
|  | import java.util.Iterator; |
|  | import java.util.LinkedList; |
|  | import java.util.List; |
|  |  |
|  | import javax.jcr.Session; |
|  |  |
|  | import org.apache.jackrabbit.api.JackrabbitSession; |
|  | import org.apache.jackrabbit.api.security.user.Authorizable; |
|  | import org.apache.jackrabbit.api.security.user.UserManager; |
|  | import org.apache.sling.api.resource.ResourceResolver; |
|  | import org.slf4j.Logger; |
|  | import org.slf4j.LoggerFactory; |
|  |  |
|  | import com.adobe.cq.sightly.WCMUsePojo; |
|  |  |
|  | /\*\* |
|  | \* @author Anirudh Sharma |
|  | \* |
|  | \* This component lists all the users and groups present in the JCR |
|  | \*/ |
|  | public class UserComponent extends WCMUsePojo { |
|  |  |
|  | /\*\* |
|  | \* Logger |
|  | \*/ |
|  | private static final Logger log = LoggerFactory.getLogger(UserComponent.class); |
|  |  |
|  | private List<String> users, groups; |
|  |  |
|  | private Session session; |
|  |  |
|  | @Override |
|  | public void activate() throws Exception { |
|  |  |
|  | try { |
|  |  |
|  | log.info("----------< Processing starts >----------"); |
|  |  |
|  | ResourceResolver resourceResolver = getResourceResolver(); |
|  |  |
|  | session = resourceResolver.adaptTo(Session.class); |
|  |  |
|  | UserManager userManager = ((JackrabbitSession) session).getUserManager(); |
|  |  |
|  | Iterator<Authorizable> userIterator = userManager.findAuthorizables("jcr:primaryType", "rep:User"); |
|  | Iterator<Authorizable> groupIterator = userManager.findAuthorizables("jcr:primaryType", "rep:Group"); |
|  |  |
|  | users = new LinkedList<>(); |
|  | groups = new LinkedList<>(); |
|  |  |
|  | while (userIterator.hasNext()) { |
|  |  |
|  | log.info("Getting user"); |
|  |  |
|  | Authorizable user = userIterator.next(); |
|  |  |
|  | if (!user.isGroup()) { |
|  | log.info("User found: {}", user.getID()); |
|  | users.add(user.getID()); |
|  | } |
|  | } |
|  |  |
|  | while (groupIterator.hasNext()) { |
|  |  |
|  | log.info("Getting group"); |
|  |  |
|  | Authorizable group = groupIterator.next(); |
|  |  |
|  | if (group.isGroup()) { |
|  | log.info("Group found {}", group.getID()); |
|  | groups.add(group.getID()); |
|  | } |
|  | } |
|  | } catch (Exception e) { |
|  |  |
|  | log.error(e.getMessage(), e); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | /\*\* |
|  | \* @return the users |
|  | \*/ |
|  | public List<String> getUsers() { |
|  | return users; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* @return the groups |
|  | \*/ |
|  | public List<String> getGroups() { |
|  | return groups; |
|  | } |
|  |  |
|  | } |

[**view raw**](https://gist.github.com/ani03sha/60fa9ccd4582accad2823dbbccbfc645/raw/72508d8c9492733cabea89dba9c377b252902f00/UserComponent.java)[**UserComponent.java**](https://gist.github.com/ani03sha/60fa9ccd4582accad2823dbbccbfc645#file-usercomponent-java) hosted with ❤ by [**GitHub**](https://github.com/)

* Here we are creating two lists containing users' and groups' IDs
* Deploy the code and drag & drop the component on the page. All the IDs will be listed as follows

|  |
| --- |
|  |
| Users and Groups |