Table of Contents

AEM Introduction

Adobe Experience Manager (AEM) is an enterprise-grade web content management system with a wide array of powerful features.

With AEM people in your organization can:

- Author and publish websites. This is achieved using two specialized environments:
  - **Author**
    Here you can enter and manage the content for your website.
    Use either of the two UIs (classic and touch-optimized) to author and administer content on a range of devices (desktop, laptop, tablet, etc).
  - **Publish**
    This environment makes the content available to your intended audience.
- Develop your website; complete with customized components, enforcement of corporate design, user access control (for editing and publishing rights) and specific views for a range of devices (e.g. mobile).
- Administer your environments to ensure that the configuration is optimized to your requirements.
- Define and implement workflows for a wide range of tasks including the creation, editing, review and publishing of content.
- Manage a repository of digital assets such as images, videos, documents and dynamic media, then integrate these assets into your website.
- Use search queries to find content no matter where it is stored in your organization.
- Set up social community tools like blogs, user groups, and calendars.
- Organize your digital assets and web pages using tagging.
- Plan, design, launch, and optimize marketing campaigns.
- Integrate an eCommerce system that will control product data, shopping carts, checkout and order fulfillment, while AEM controls the data display and marketing campaigns.

What is AEM

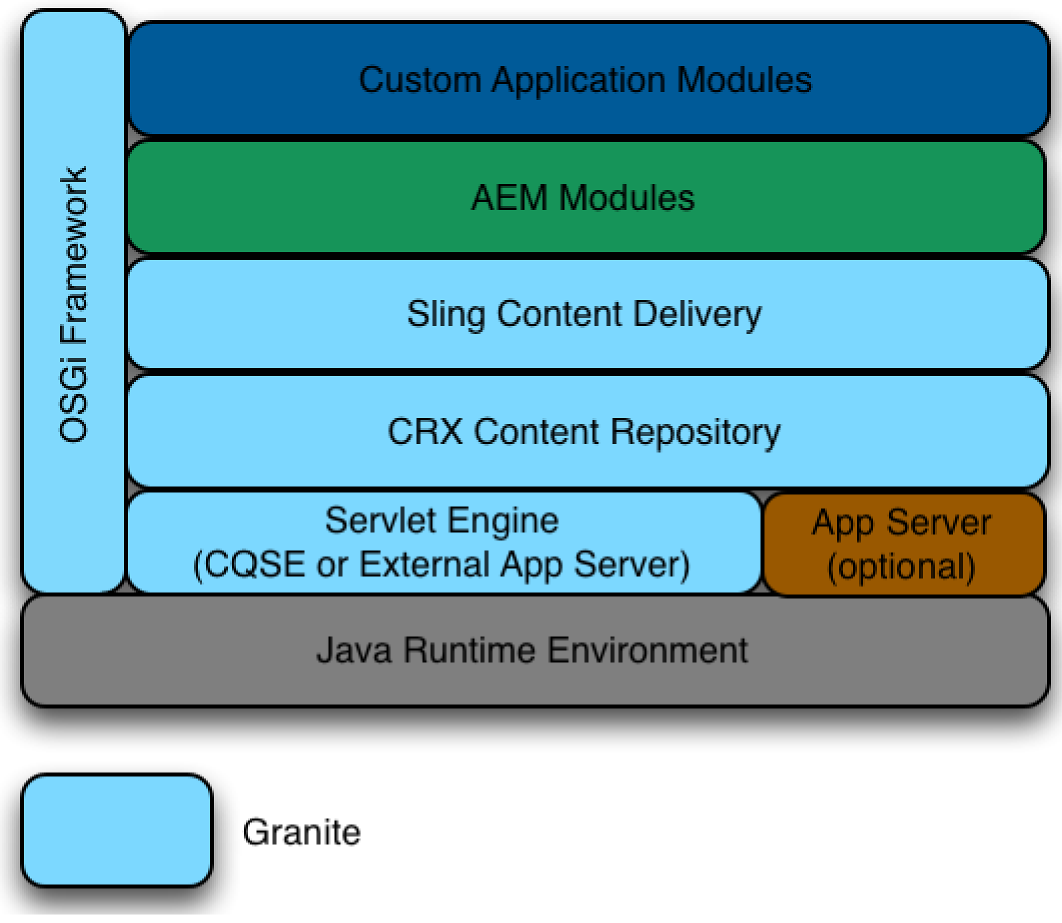
Adobe Experience Manager (AEM) is a java based content management system that is offered from Adobe. It was previously called Day CQ5, but was acquired from Adobe in 2010. AEM is based on a content repository and uses the JCR to access the content in the repository. AEM uses the Apache Sling framework to map request url to the corresponding node in the content repository. It also uses the OSGI framework to internally allow modular application development.

What are the advantages of AEM over another CMS

One big advantage of AEM over another CMS is how it integrates with other products from Adobe and with the Adobe Marketing Cloud. AEM comes built in with features like workflows to control content in the CMS, the us   e of search queries to find anything you are looking for, setting up social collaboration, tagging content, and a way to manage your digital content.
AEM also includes a way to manage mobile applications, mobile websites, e-commerce, and marketing campaign management.

The AEM Stack OR Eplain AEM Architecture.

The AEM Stack can be divided as follows, from lowest to highest level.



JAVA Platform: Adobe Experience Manager (AEM) is a Java web application and therefore requires a server-side Java Runtime Environment (JRE). At least JRE 1.6 is required, though JRE 1.7 is recommended.

Granite Platform:

Granite is Adobe's open web stack. It forms the technical foundation on which AEM is built.

OSGI FRAMEWORK

OSGi is a dynamic software component system for Java. In an OSGi-based system, an application is composed of an assemblage of components, called bundles in OSGi terminology, which can be dynamically installed, started, stopped and uninstalled at runtime, without shutting down and restarting the entire application.In a running AEM instance, bundle management is provided through the AEM Web Console at http://:/system/console/bundles.

SERVLET ENGINE:

In a quickstart installation, the built-in CQSE servlet engine runs as a bundle within the OSGi framework. In a war file installation servlet handling is delegated to a third-party application server.AEM includes a built-in servlet engine (CQSE) which runs as a bundle within the OSGi framework when AEM is deployed via the standalone quickstart jar file.

JCR CONTENT REPOSITORY

All data within AEM is stored in the built-in CRX content repository, which is an implementation of the Java Content Repository Specification (JCR).

All data in AEM is stored in its built-in content repository.

- The repository built into AEM is called CRX.
- CRX is Adobe's implemention of the Content Repository Specification for Java Technology 2.0, an official standard published through the Java Community Process as JSR-238 (version 1.0 was known as JSR-170)

SLING CONTENT DELIVERY

AEM is built using Sling, a Web application framework based on REST principles that provides easy development of content-oriented applications. Sling uses a JCR repository, such as Apache Jackrabbit, or in the case of AEM, the CRX Content Repository, as its data store. Sling has been contributed to the Apache Software Foundation

AEM modules

Adobe Experience Manager runs on Granite platform, within OSGi framework. Individual AEM modules are WCM, DAM, Workflow, etc.

Customer Applications

Customer Applications run on AEM. Customer specific applications (websites, etc. also run within OSGi framework).

AEM servlet

Servlets can be registered as OSGi services. For a Servlet registered as an OSGi service to be used by the Sling Servlet Resolver, the following restrictions apply:

- Either the **sling.servlet.paths or the sling.servlet.resourceTypes** service
- reference property must be set. If neither is set, the Servlet service is ignored.

- If the sling.servlet.paths property is set, all other sling.servlet.* properties are ignored.
- Otherwise a Resource provider is registered for the Servlet for each permutation resource types, selectors, extensions and methods.

Components

Components are reusable modules that implements a specific application logic to render the content of our website.

The definition of a component can be broken down as follows:

- AEM components are based on Sling.
- AEM components are (usually) located under:
  - HTL: /libs/wcm/foundation/components
  - JSP: /libs/foundation/components
- Project/Site specific components are (usually) located under:
  - /apps/<myApp>/components

Components provide the logic (code) to render content. They include both templates and specific components such as Text with Image, Column Control and Subtitle amongst others.

Out-of-the-Box Components Within AEM
AEM comes with a variety of out-of-the-box components that provide comprehensive functionality including:

- Paragraph System (parsys)
- Page (responsivegrid - touch-enabled UI only)
- Text
- Image, with accompanying text
- Toolbar

Templates

A CQ template enables you to define a consistent style for the pages in your application. A template comprises of nodes that specify the page structure.

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

Template is basically a blueprint for creating a page and describes what are the components that can be used within the page. It has the same hierarchy as page but with no content. For creating a page we need a template.

Sling:resourceType – It is a path which locates the script to be used for rendering the content. Path used can be absolute or relative.

Sling:resourceSuperType – It is used to achieve inheritance. When it is set, it inherits the specified component to this component.

1. Defn of allowedPaths, allowedChildren, allowedParents ranking?

allowedPaths - Path of a page that is allowed to be based on this template.

allowedChildren - Path of a template that is allowed to be a child of this template.

allowedParents - Path of a template that is allowed to be a parent of this template.

ranking - Rank of the template. Used to display the template in the User Interface.

2. Can we create a page without template?

Yes, we can create a page without template with the following procedure:

1. Under content, create a node of type **cq:page.**
2. Under that node create a node with name **Jcr:content** and type **cq:pagecontent**
3. Set the node properties **Sling Resource Type** as the **path of component.**
4. Double click on first node and the desired page will open.

3. What is Dialog, Design Dialog and cq:dialog?

dialog	**design-dialog**
Dialog will change the content at the page level.	Design dialog will change the content at the template level.
authored in edit mode.	authored in design mode.
node name should be dialog.	node name should be design_dialog.
stored under pages jcr:content node.	stored under design page located under /etc/design/default.
we can get value from properties object.	we can get design dialog value from currentStyle object.
jcr:primaryType is cq:Dialog.	jcr:primaryType is cq:Dialog.

cq:dialog - It is the dialogs for the touch-optimized UI(editor.html).

- It uses the Granite UI framework.
- Node name is cq:dialog.
- jcr:primaryType = nt:unstructured, sling:resourceType = cq/gui/components/authoring/dialog for cq:dialog

4. Difference between parsys and iparsys.

parsys – It is a placeholder called "**Paragraph System**", where we can drag and drop or add other components or scripts at page level.

iparsys - The inherited paragraph system is a paragraph system that also allows you to inherit the created paragraphs from the parent. it is similar to parsys except that it allows to inherits parent page "paragraph system" at template level. You can also cancel paragraph inheritance at a level at any time. It has two checkbox options to cancel/disable the inheritance.

- **Cancel Inheritance** - If selected, the components in this paragraph system are not passed down to the child pages.
- **Disable Inheritance -** If selected, components of this paragraph system on this page are not inherited from the parent page.

5. Why we need to include global.jsp if we are creating a component in jsp?

The JSP script file global.jsp is used to provide quick access to specific objects (i.e. to access content) to any JSP script file used to render a component. Therefore global.jsp should be included in every component rendering JSP script where one or more of the objects provided in global.jsp are used.

6. What is parbase?

Parbase is a key component as it allows components to inherit attributes from other components, similar to subclasses in object oriented languages such as Java, C++, and so on. For example, when you open the /libs/foundation/components/text node in the CRXDE Lite, you see that it has a property named sling:resourceSuperType, which references the parbase component. The parbase here defines tree scripts to render images, titles, and so on, so that all components subclassed from this parbase can use this script.

7. What is xtype ?

xtype is your widget, which you place in your dialog to capture author data. xtypes such as textfield, numberfield, selection etc.

8. What is difference between SlingAllMethodsServlet and SlingSafeMethodsServlet?

Sling All Methods Servlet

Servlet implementation that responds to all HTTP methods. These are primary used to surface endpoints that respond to POST (and GET) Requests.

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

Sling Safe Methods Servlet

Servlet implementation that responds to "Safe" HTTP Requests. These are primary used to surface endpoints that respond to GET Requests.

- Helper base class for **read-only Servlets** used in Sling.
- This class extends the GenericServlet and itself does not support for the POST, PUT and DELETE methods.

9. What is the difference between c:import, sling:include and cq:include?

1. <c:import url="layout-link.jsp" />
I assume this is the import tag of the Standard Tag Library. This tag is documented at http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/c/import.html and does not know about Sling directly.

But assuming that this tag is using a RequestDispatcher to dispatch the request, this tag will also pass Sling and the Sling resource resolver.

2. <sling:include path="layout-link.jsp" />
This is the include tag of the Sling JSP Tag library. This tag knows about Sling and also supports Request Dispatcher Options.

3. <cq:include script="layout-link.jsp" />
This tag is Communiqué specific extension of the Sling JSP Tag library include tag. IIRC it supports callings scripts in addition to just including renderings of resources.

OSGI Life Cycle

OSGi is a framework which allows modular development of applications using java. A large application can be constructed using small reusable components(called bundles in terms of OSGi) each of which can be independently started, stopped, and also can be configured dynamically while running without requiring a restart. Following are the states of OSGI life cycle:

Installed – The bundle has been successfully installed.

Resolved – All Java classes that the bundle needs are available. This state indicates that the bundle is either ready to be started or has stopped.

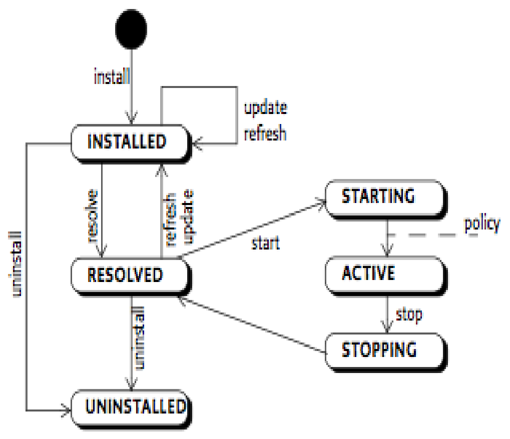
Starting – The bundle is being started, the BundleActivator.start method will be called, and this method has not yet returned. When the bundle has an activation policy, the bundle will remain in the STARTING state until the bundle is activated according to its activation policy.

Active – The bundle has been successfully activated and is running; its Bundle Activator start method has been called and returned.

Stopping – The bundle is being stopped. The BundleActivator.stop method has been called but the stop method has not yet returned.

Uninstalled – The bundle has been uninstalled. It cannot move into another state.

Figure 4.28 *State diagram Bundle*



Example code

```
package com.adobe.training.core;

import org.osgi.framework.BundleActivator;

import org.osgi.framework.BundleContext;


import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Activator implements BundleActivator {

private final Logger logger = LoggerFactory.getLogger(getClass());
```

```
    public void start(BundleContext context) throws Exception {
       logger.info("#################Bundle Started#################");
    }
              public void active(BundleContext context) throws Exception {
       logger.info("#################Bundle Started#################");
    }
    public void stop(BundleContext context) throws Exception {
       logger.info("#################Bundle Stopped#################");
    }
}
```

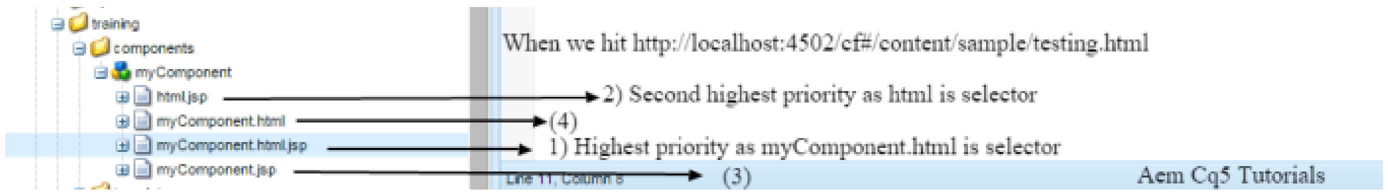
Sling Resource Resolver

- When end customer is accessing the page,url is mapped with respect to JCR node.If we want to access the JCR node url must be decomposed then JCR is accessed using sling.
- Click on JCR node property sling:resourceType there we have a path pointing to template.If we navigate to that template path we have JCR node property sling:resourceType there we have a path pointing to component.
- Before call to component is made,path will be appended based on priorities: firstly the priority will be given to name of the component,selector and extension,second the priority will be given to selector and extension,third the priority will be given to selector and lastly the priority will be given to extension.Finally the call to the component will be made.

Sling Resource Resolution

protocol	**host**	**content path**	**selector(s)**	**extension**		**suffix**		**param(s)**
http://	www.mywebsite.com	products/product1	.printable.a4	html	/	a/b	?	x=12
http://	**localhost:4502**	**content/aemTutorials/sighlyPage**	**.test**	html			?	x=12

Sling Resource Resolution Preference


When we hit http://localhost:4502/cf#/content/sample/testing.html

2) Second highest priority as html is selector

(4)

1) Highest priority as myComponent.html is selector

(3)

So the final priority order will be myComponent.html.jsp–>html.jsp–>myComponent.jsp –>myComponent.html

Below is the priority of resolution in Descending order(Highest to lowest):-

1. Selector +Extension .jsp –> **test.myComponent.html.jsp**
2. Selector.jsp –> **test.jsp**
3. Extension.jsp –>**html.jsp**
4. Node Name.jsp –> **myComponent.jsp**
5. method name.jsp(Based on type of request we make Either GET or POST) –> **GET.jsp or POST.jsp**

Note:-If no match is found then it will look for sling:resourceSuperType if available on the component, then it goes to parent component and again run above 5 rules for script resolution.

Client Library

What are client libraries? Why we need client libraries when we can include css/js directly?

Client libraries in aem is one of the most widely used features provided by Adobe. It is a client-side mechanism managing JS & CSS to page.

The standard way to include a client library(i.e., css/js file) is-

...

<head>

...

<script type="text/javascript" src="/etc/clientlibs/granite/jquery/source/1.8.1/jquery-1.8.1.js"></script>

...

</head>

...

While this approach works in AEM, it can lead to problems when pages and their constituent components become complex. In such cases there is complexity that multiple copies of the same JS library may be included in the final HTML output. To avoid this and to allow logical organization of client-side libraries AEM uses client-side library folders. A client-side library folder is a repository node of type **cq:ClientLibraryFolder.**

AEM provides Client-side Library Folders, which allow you to store your client-side code in the repository, organize it into categories, and define when and how each category of code is to be served to the client.

Creating Client Library Folders

Create a cq:ClientLibraryFolder node to define JS and CSS libraries and make them available to HTML pages. Use the categories property of the node to identify the library categories to which it belongs.

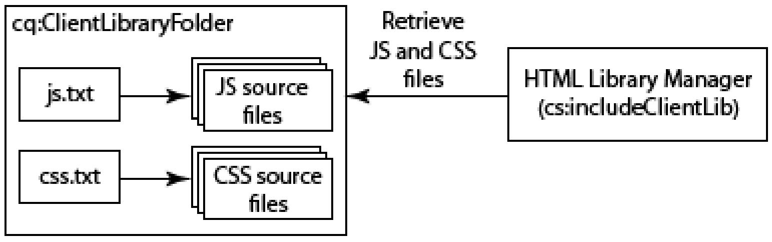
The node contains one or more source files that, at runtime, are merged into a single JS and/or CSS file. The name of the generated file is the node name with either the .js or .css file name extension. For example, the library node named cq.jquery results in the generated file named cq.jquery.js or cq.jquery.css.

Client library folders contain the following items:

- The JS and/or CSS source files to merge.
- Resources that support CSS styles, such as image files.

  Note: You can use subfolders to organize source files.

- One js.txt file and/or one css.txt file that identifies the source files to merge in the generated JS and/or CSS files.


cq:ClientLibraryFolder
js.txt
JS source files
css.txt
CSS source files
Retrieve JS and CSS files
HTML Library Manager (cs:includeClientLib)

Client Library folder structure and its importance:-

jcr:primaryType – The “clientlib” node must be of type cq:ClientLibraryFolder, so that aem understand that this is client lib folder.

categories – It is used to categorize or identify respective client library folder from where js and css files needs to be included. Value of this field is a String array i.e you could use an existing name here, which means that when we include client library in component jsp file using a category name that is available in more than one client library folder, then js and css files of all such folder will be send in the response. If you want to include files from a particular clientlib folder then keep the category name unique.

dependencies – This is a list of other client library categories on which this library folder depends. For example, given two cq:ClientLibraryFolder nodes F and G, if a file in F requires another file in G in order to function properly, then at least one of the categories of G should be among the dependencies of F.

embed – AEM will merge all embedded clientlibs into the current clientlib. This is usually used for minimizing requests and for accessing clientlibs. For example if Clientlib A embed Clientlib B which embed Clientlib C, then Clientlib A will be be loaded by embedding Clientlib B code.

Replication

Replication agents are central to Adobe Experience Manager (AEM) as the mechanism used to:

- Publish (activate) content from an author to a publish environment.
- Explicitly flush content from the Dispatcher cache.
- Return user input (for example, form input) from the publish environment to the author environment (under control of the author environment).

What is Replication in CQ?
Replication is used for 3 main operations in CQ:
1. To move content from CQ Author to CQ Publish. This is called as Forward Replication or Activating content.
2. To explicitly flush the contents into Dispatcher Cache
3. To synchronize user input (Ex: Blog comments) between Publish Instances. This is done via the concept of Reverse Replication.

What are Replication Agents?
- Replication Agents are used to perform Replication activities.
- Replication Agents are configured on Author and Publish.
- Generally you configure the following Replication Agents on the Author:
  * Forward Replication Agents
  * Reverse Replication Agents
- You configure the following Replication Agent on the Publish:
  * Dispatcher Flush Agent

What is Default Agent?

By Default, a Default Forward Replication agent is configured on the author instance. This will point to a Publish instance running on port 4503. If you are running the Publish instance on a different port you need modify this on the Default Agent Configuration.

How do you access Replication Agents on an author instance?

Go to http://localhost:4502/miscadmin#/etc/replication and you can see all the Replication Agents configured on this CQ instance. (Assuming Author is running on 4502 port).

Forward Replication

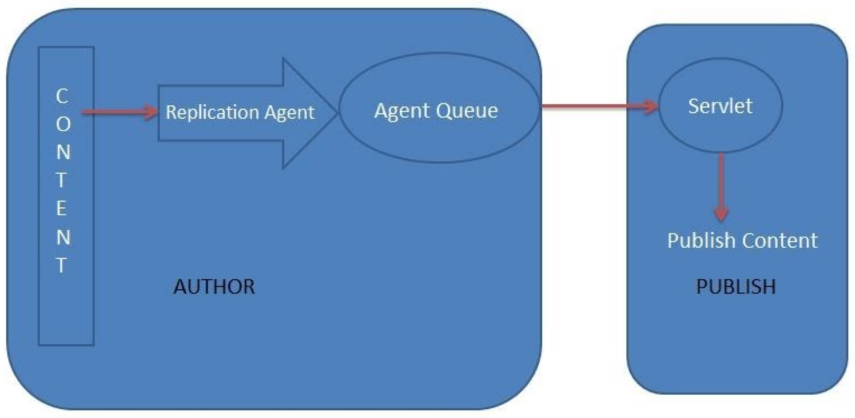
- Replicating content from author to publish instances (or dispatcher) is known as Forward Replication.

- Replication agents to all publish instances to which content should be replicated to, should first be configured on the author instance(s).

- When content is activated (for example a page is activated), the content is placed in the queue of all forward replication agents configured.

- The replication agents then package the content and forwards it to the appropriate Publish instance using the protocol configured on the replication agent (default is HTTP).

- A servlet in the Publish instance receives the request and it publishes the received content.



Forward Replication Process

REPLICATION PROCESS:

- First, the author requests that certain content to be published (activated).
- The request is passed to the appropriate default replication agent.
- Replication agent packages the content and places it in the replication queue.
- The content is lifted from the queue and transported to the publish environment using the configured protocol.
- A servlet in the publish environment receives the request and publishes the received content, the default servlet is HTTP://LOCALHOST:4502/BIN/RECEIVE.

Reverse Replication

Overview:

Reverse Replication is process of replicating user data back to author instance from publish instance. This may include any user related data, comments, form data etc.

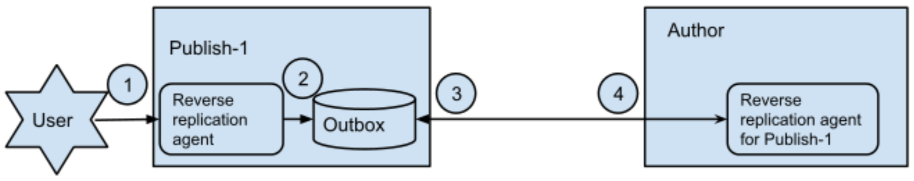
Architecture:

In reverse replication, Publish instance reverse replication agent puts user data in "outbox" (An repository location where data is temporarily hold). Author instance have another agent matching agent which polls data from publish at regular interval. If data is found in outbox of publish instance, it is sync in author instance. This way synchronisation of data between publish to author is controlled.

How to achieve reverse replication & Where it is used frequently ?

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

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


Publish-1
Author
User
1
2
3
4
Reverse replication agent
Outbox
Reverse replication agent for Publish-1

1. User Post data to publish instance.
2. Reverse Replication agent check if this data is subjected to reverse replication and then puts it in it's outbox.
3. Author reverse replication agent for that publish instance, checks if there is any data to pull from outbox of publish instace.
4. If data is available, Author pulls data and synchronize it with author.

Replication Agents - Configuration Parameters

When configuring a replication agent from the Tools console, four tabs are available within the dialog:

Settings

- **Name**

  A unique name for the replication agent.

- **Description**

  A description of the purpose this replication agent will serve.

- **Enabled**

  Indicates whether the replication agent is currently enabled.

  When the agent is **enabled** the queue will be shown as:

  - **Active** when items are being processed.
  - **Idle** when the queue is empty.
  - **Blocked** when items are in the queue, but cannot be processed; for example, when the receiving queue is disabled.

- **Serialization Type**

  The type of serialization:

    - **Default**: Set if the agent is to be automatically selected.
    - **Dispatcher Flush**: Select this if the agent is to be used for flushing the dispatcher cache.
- **Retry Delay**

  The delay (waiting time in milliseconds) between two retries, should a problem be encountered.

  Default: 60000
- **Agent User Id**

  Depending on the environment, the agent will use this user account to:

    - collect and package the content from the author environment
    - **create and write the content on the publish environment**

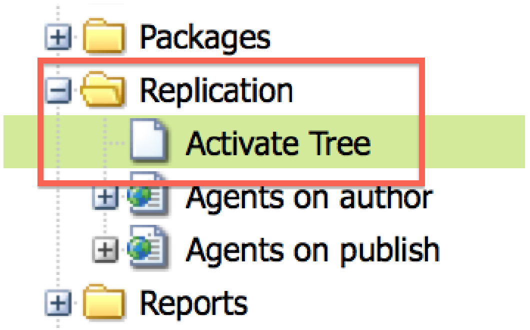
Tree Activation

Overview: Through Activation you can only activate one resource at a time or number of limited resource through multi select. However some time you have a situation where more than one files or a resource tree needs to be activated (For example new section of website is going live).

Using Tree Activation:

- From site admin click on Tools -> Replication
- Then double click on activate tree.

  You can also directly go there by **HOST:PORT/etc/replication/treeactivation.html**



- You can then select Start Path. All resource under that path and subpath will be activated. If Possible do not select path with a lot of children's in it. That might cause performance issue.
- You can also various options for activation. Based on option selected all pages will get activated.
- Here is what various options mean

Tree Activation Options

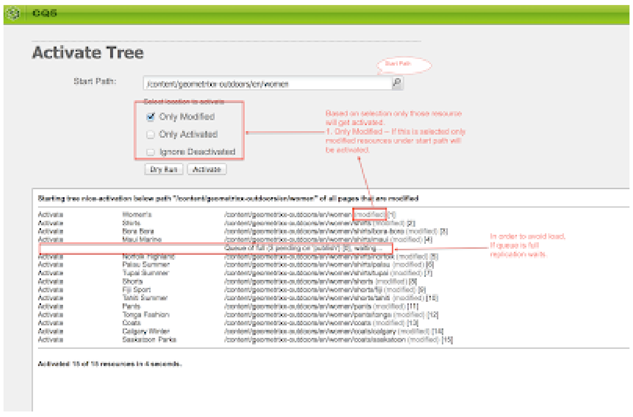
For Resource Option:

- **Only Modified:** only activate pages that have been modified.

- **Only Activated:** only activate pages that have (already) been activated. Acts as a form of reactivation.
- **Ignore Deactivated**: ignore any pages which have been deactivated.

For Activation:

- Select **Dry Run** if you want to check which pages would be activated. This is only an emulation, no pages will be activated.
- Select **Activate** if you want to activate the pages.


CQ5
Activate Tree
Start Path
Start Path:
Select location to activate
Only Modified
Only Activated
Ignore Deactivated
Dry Run
Activate
Based on selection only those resource will get activated.
1. Only Modified – If this is selected only modified resources under start path will be activated.
In order to avoid load, If queue is full replication waits.

```
/content/geometrixx-outdoors/en/women

Starting tree nice-activation below path "/content/geometrixx-outdoors/en/women" of all pages that are modified
Activate    Women's            /content/geometrixx-outdoors/en/women (modified) [1]
Activate    Shirts             /content/geometrixx-outdoors/en/women/shirts (modified) [2]
Activate    Bora Bora          /content/geometrixx-outdoors/en/women/shirts/bora-bora (modified) [3]
Activate    Maui Marine        /content/geometrixx-outdoors/en/women/shirts/maui (modified) [4]
                               Queue of full (3 pending on 'publish') [5], waiting...
Activate    Norfolk Highland   /content/geometrixx-outdoors/en/women/shirts/norfolk (modified) [5]
Activate    Palau Summer       /content/geometrixx-outdoors/en/women/shirts/palau (modified) [6]
Activate    Tupai Summer       /content/geometrixx-outdoors/en/women/shirts/tupai (modified) [7]
Activate    Shorts             /content/geometrixx-outdoors/en/women/shorts (modified) [8]
Activate    Fiji Sport         /content/geometrixx-outdoors/en/women/shorts/fiji (modified) [9]
Activate    Tahiti Summer      /content/geometrixx-outdoors/en/women/shorts/tahiti (modified) [10]
Activate    Pants              /content/geometrixx-outdoors/en/women/pants (modified) [11]
Activate    Tonga Fashion      /content/geometrixx-outdoors/en/women/pants/tonga (modified) [12]
Activate    Coats              /content/geometrixx-outdoors/en/women/coats (modified) [13]
Activate    Calgary Winter     /content/geometrixx-outdoors/en/women/coats/calgary (modified) [14]
Activate    Saskatoon Parka    /content/geometrixx-outdoors/en/women/coats/saskatoon (modified) [15]

Activated 15 of 15 resources in 4 seconds.
```

AEM servlet

Servlets can be registered as OSGi services. For a Servlet registered as an OSGi service to be used by the Sling Servlet Resolver, the following restrictions apply:

- Either the **sling.servlet.paths or the sling.servlet.resourceTypes** service
- reference property must be set. If neither is set, the Servlet service is ignored.
- If the sling.servlet.paths property is set, all other sling.servlet.* properties are ignored.
- Otherwise a Resource provider is registered for the Servlet for each permutation resource types, selectors, extensions and methods.

How many ways call from ajax to servlet

OR

How are different ways to register servlet in aem

You can register a Servlet using the two Standard approaches:

1. Registering the servlet by path

```
@SlingServlet(
paths={"/bin/customservlet/path"} )
```

```
@Properties({
@Property(name="service.pid",
value="com.day.servlets.SampleServlet",propertyPrivate=false),
@Property(name="service.description",value="SampleDescription",
propertyPrivate=false),
@Property(name="service.vendor",value="SampleVendor", propertyPrivate=false)
})
public class SampleServletname extends SlingAllMethodsServlet
{
@Override
protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response)
throws ServletException, IOException
{
}
}
```

2. Register servlet by ResourceType

```
@SlingServlet(
resourceTypes = "sling/servlet/path",
selectors = "json",
extensions = "html",
methods = "GET")
public class MyServlet extends SlingSafeMethodsServlet {
@Override
protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response)
throws ServletException, IOException {
...
}
}
```

Another way to write in aem 6.4 version
@Component(service = Servlet.class, immediate = true, property =
{Constants.*SERVICE_DESCRIPTION* + "=myservlet servlet ","sling.servlet.paths=" +
"/bin/sling/ve" })

Registering the servlet by path vs ResourceType

Registering the servlet by **resourceType** is more preferential than **path** , because

- use of resourceType is that the Sling Engine will take care of permissions for you. Users who cannot access a particular resource will not be able to invoke the servlet. Hence register by resourcetype is more secure.
- While defining a path , you must be specific what all paths are allowed to be used in the ServletResource OSGi service. If you define something randomly, servlet might not be functional. Only a limited paths are allowed and the rest are blocked unless you open them up. This is resolved using resourceType.

SCR annotations in OSGI

@component - The @Component annotates an implementation class and is used to declare it as a component type. It is the only required annotation. If this annotation is not declared for a Java class, the class is not declared as a component.

@service - The @Service annotation defines whether and which service interfaces are provided by the component. This is a class annotation.

@reference - The @Reference annotation defines references to other services made available to the component by the Service Component Runtime.

@property - The @Property annotation defines properties which are made available to the component through the **ComponentContext.getProperties()** method. These tags are not strictly required but may be used by components to defined initial configuration.

Additionally properties may be set here to identify the component if it is registered as a service, for example the service.description and service.vendor properties

Difference between OSGI component and service

Component - If you want the life of your object to be managed by the OSGi container, you should declare it as a component. Using annotations, you could make a POJO a OSGi component by annotating it with **@Component.** With this, you will get the ability to start, stop and configure the component using the felix web console. All objects managed by OSGi container are components. You qualify components as services. This means that all services are components but not vice-versa.

Service- OSGi components can be made as OSGi service by marking it with @Service annotation. When you mark a component as service, you could refer (call) this service from other osgi components. Components can refer/call (using container injection – @Reference) other services but not components. In other words, a component cannot be injected into another component / service. Only services can be injected into another component.

--------------------------------How to create servlet by using ajax---------------------------------

----------------------------HTML-JQuery-Ajax-----------------------------------

```
<!doctypehtml>
<html>
<head>
<scriptsrc="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script> </head>
<body>
SearchField: <inputtype="text"id="fullname"/><br>
<inputtype="button"value="submit"id="bttHello"/>
<script>
```

```
$(document).ready(function() {
alert("first");
$('#bttHello').click(function(){
alert("second");
var fullname1=$('#fullname').val();
alert("third");
$.ajax({
type:'GET',
data: {fullname: fullname1},
url:'/bin/sling/ve',
success: function(result){
alert(result);
},
error: function(request,error) {
alert("some error");
}
});
});
});
</script>
</body>
</html>
```

----------------------------------**servlet**--

```
package com.aem.community.core.servlets;
import java.io.IOException;
import java.io.PrintWriter;
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.servlets.SlingAllMethodsServlet;
import org.osgi.framework.Constants;
import org.osgi.service.component.annotations.Component;
```

```
@Component(service = Servlet.class, immediate = true, property =
{Constants.SERVICE_DESCRIPTION + "=myservlet servlet ",
"sling.servlet.paths=" + "/bin/sling/ve" })

public class Myservlet extends SlingAllMethodsServlet {

protected void doGet(final SlingHttpServletRequest req, final SlingHttpServletResponse resp)
throws ServletException, IOException {
                PrintWriter out=resp.getWriter();
                out.println("request is coming");
        String name3=req.getParameter("fullname");
          out.println("name::::::::::::"+name3);
        } }
```

Overlay & Override Component

Overlay Component: Creating a custom component by copying a foundation component to your project and modifying it based on the need. For example you copy image component from "/libs/foundation/components/image" to your site folder "/apps/testsite/components" by doing so you are creating a new component with is exactly same as Image component. After copying you can make changes to component based on your requirements. But the problem with this approach would be that if you are upgrading CQ then new version of CQ might have new implementation of "/libs/foundation/components/image" component than those changes will not be reflected in your "/apps/testsite/components/image" component, so you have to manually make those changes in custom component.

Extend/Override Component: Creating a custom component manually by creating all necessary nodes and setting value of **"sling:superResourceType"** property as "/libs/foundation/components/image". By doing this you inherit all the feature of image component, even after upgrade you still inherit the features of image component.

----------------------------------Ajaxcall-servlet and service---
--Html---

```
Hello
<!doctypehtml>
<html>
<head>
<scriptsrc="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
</head>
<body>
```

```
SearchField: <inputtype="text"id="fullname"/><br>
City:<inputtype="text"id="city"/><br>
<inputtype="button"value="submit"id="bttHello"/>
<script>
$(document).ready(function() {
alert("first");
$('#bttHello').click(function(){
alert("second");
var fullname1=$('#fullname').val();
var city1=$('#city').val();
alert("third");
$.ajax({
type:'GET',
data: {fullname: fullname1,city2:city1},
url:'/bin/sling/ve',
success: function(result){
alert(result);
},
error: function(request,error) {
alert("some error");
}
});
});
});
</script>
</body>
</html>
```

---servlet--

```
package AEMNEW12.core.servlets;
import java.io.IOException;
import java.io.PrintWriter;
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.servlets.SlingAllMethodsServlet;
import org.osgi.framework.Constants;
import org.osgi.service.component.annotations.Component;
import org.osgi.service.component.annotations.Reference;
import AEMNEW12.core.service.Login;
@Component(service = Servlet.class, immediate = true, property =
{Constants.SERVICE_DESCRIPTION + "=myservlet servlet ","sling.servlet.paths=" +
"/bin/sling/ve" })
public class Myservlet extends SlingAllMethodsServlet {
        @Reference
        Login login;
        protected void doGet(final SlingHttpServletRequest req, final SlingHttpServletResponse
resp) throws ServletException, IOException {
                PrintWriter out=resp.getWriter();
                out.println("request is coming");
        String name3=req.getParameter("fullname");
        String city3=req.getParameter("city2");
        String result= login.login(name3, city3);
        out.println("result:::::::::::::::"+result);
        }
}
```

--Interface--

```
package AEMNEW12.core.service;
publicinterface Login {
        public String login(String name, String password);
}
```

--interface Impl class---

```
package AEMNEW12.core.service;
import org.osgi.service.component.annotations.Component;
@Component(service =Login.class)
publicclass LoginImpl implements Login{
        public String login(String name, String password) {
                // TODO Auto-generated method stub
                if(name.equals("AEM"))              {
                return"AEM";
                }else
                {
                        return"password mistach";
                }
        }
}
```

Resource Resolver

The **ResourceResolver** defines the service API which may be used to resolve Resource objects. The resource resolver is available to the request processing servlet through the **SlingHttpServletRequest.getResourceResolver()** method. A resource resolver can also be created through the ResourceResolverFactory.

The **ResourceResolver** is also an **Adaptable** to get adapters to other types. A JCR based resource resolver might support adapting to the JCR Session used by the resolver to access the JCR Repository.

ResourceResolverFactory

The ResourceResolverFactory defines the service API to get and create ResourceResolvers. As soon as the resource resolver is not used anymore, ResourceResolver.close() should be called. All resource resolvers returned by the same resource resolver factory must use the same search path

ResourceResolverFactory is like any other service that is available in OSGI framework. It's most frequent use is accessing resource resolver. You can see its detailed use in your felix console, /system/console/configMgr. Search for "Apache Sling Resource Resolver Factory".

This is the way we create the object of RS by RSF but this not recommended to use

ResourceResolver serviceResourceResolver = resourceResolverFactory.getServiceResourceResolver(authenticationInfo);

Resource someResource = serviceResourceResolver.getResource("/path/to/resource");

What is Resource.

Resources are pieces of content on which Sling acts

The Resource is also an Adaptable to get adapters to other types. A JCR based resource might support adapting to the JCR Node on which the resource is based.

What is pagemanager?

The page manager provides methods for page level operations.

Program: How to get get node and properties by using servlet

```
package com.aem.community.core.servlets;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
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.ResourceResolver;
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;
import com.day.cq.wcm.api.Page;
import com.day.cq.wcm.api.PageFilter;
import com.day.cq.wcm.api.PageManager;
@Component(service = Servlet.class, immediate = true, property = {
   Constants.SERVICE_DESCRIPTION + "=practice servlet ",
   "sling.servlet.paths=" + "/bin/sling/vee" })
public class FetchPageProperties extends SlingAllMethodsServlet{
   private static final Logger log= LoggerFactory.getLogger(FetchPageProperties.class);
          protected void doGet(final SlingHttpServletRequest req,
                 final SlingHttpServletResponse resp) throws ServletException, IOException {
                   resp.setContentType("text/html");
              PrintWriter out=resp.getWriter();
              String FullName=req.getParameter("fullname");
              out.println(FullName);
             /*  ResourceResolver resourceResolver=req.getResourceResolver();
              Resource resource=resourceResolver.getResource("/content/AEMMaven12/en");
```

```
                Node node=resource.adaptTo(Node.class);
                log.info("request1");
               try {
                        NodeIterator itr=node.getNodes();
                        while(itr.hasNext())
                        {
                                Node childNode=itr.nextNode();
                                String nodeName=childNode.getName();
                                String nodePath=childNode.getPath();

        out.println("nodeNAme::::::::"+nodeName+"nodePath:::::::::::::"+nodePath);
                    log.info("request2"+childNode);
                                if(childNode.hasProperty("jcr:title"))
                                {
                              log.info("request3");
                        String title=childNode.getProperty("jcr:title").getValue().getString();
                        String pageTitle=childNode.getProperty("pageTitle").getValue().getString();
                        String
resouceType=childNode.getProperty("sling:resourceType").getValue().getString();


        out.println("title"+title+"pageTile"+pageTitle+"resouceType"+resouceType);
                                }
                        }
                  } catch (RepositoryException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                  }
*/
                ResourceResolver resourceResolver=req.getResourceResolver();
                PageManager pageManager=resourceResolver.adaptTo(PageManager.class);
                Page page=pageManager.getPage("/content/AEMMaven12");
                List<String> lis=new ArrayList<String>();
                Iterator<Page> rootPage=page.listChildren(new PageFilter(),true);
                while(rootPage.hasNext())
                {
                   Page childPage=rootPage.next();
                    String path=childPage.getPath();
                    String name=childPage.getPageTitle();
                    String tilee=childPage.getName();
                    out.println("path::::::::::::"+path);
                    lis.add(path);
                }
                }
}
```

Sling Model

What are sling models?

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

Why sling models?

Using sling models, you can reduce coding efforts,code is more maintainable using sling model i.e,

No need to write redundant code.

Annotations used:

Inject :- email property is always looked from Resource(after adapting to ValueMap), if this property value is not present then it returns null. If this property is in itself not available then it throws exception.

@Inject @Optional :- It is not mandatory to have firstname property in Resource(Backend fields made optional).

@Named :- Use it if jcr node property name is different than class variable.

@Default :- use it if you want to assign default values, If empty then assign default value as ‘NA’.

@PostConstructor:- annotation defines that this method will run after injecting all field and it is used to write business logics.

@Model :- Retrieving values from JCR node through inject in sling model

@via :- If injections should be based on java bean property of adaptable, it can be indicated by @via annotation

Working: As per user requirements,I created a component and a dialog.In dialog, I created a multifield so I used sling model.

In eclipse,I created a pojo class(model class).To make this pojo class a component I used annotation **@model(adaptables=Resource.class).**Then for the member variables I used **@inject** annotation to fetch the values from repository.Then I wrote a post construct method using **@postconstruct** annotation where I implemented my business logic,it returns a list object and we send it to html with getter method.

In crxdelite,UI team gives the template here I need to integrate my slightly code.I use **sly tag** with **use attribute** create an reference object and call sling model helper class.Next the reference object is called,all the methods are got,now all values are present in object.Finally iterate using the **list attribute.**

--------------------HTML-------------------------------------

```
Hello
<sly data-sly-use.sampleInfo="com.aem.community.core.models.Sample">
${sampleInfo.text}<br>
${sampleInfo.psd}<br>
<img src="${sampleInfo.link}" alt="link"><br>
<img src="${sampleInfo.link1}" alt="link1"><br>
${sampleInfo.backgroundColor}<br>
<img src="${sampleInfo.blrImage}" alt="blrimage"><br>
${sampleInfo.backgroundoption}<br>
${sampleInfo.alignment}
${sampleInfo.message}
</sly>
```

----------------------------sling model--------------------------

```
package com.aem.community.core.models;
import javax.annotation.PostConstruct;
import javax.inject.Inject;
import javax.inject.Named;
import javax.jcr.Node;
import javax.jcr.NodeIterator;
import javax.jcr.RepositoryException;

import org.apache.sling.api.SlingHttpServletRequest;
import org.apache.sling.api.resource.Resource;
import org.apache.sling.api.resource.ResourceResolver;
import org.apache.sling.models.annotations.Default;
import org.apache.sling.models.annotations.DefaultInjectionStrategy;
import org.apache.sling.models.annotations.Model;
import org.apache.sling.models.annotations.injectorspecific.SlingObject;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@Model(adaptables =Resource.class,
defaultInjectionStrategy=DefaultInjectionStrategy.OPTIONAL)
public class Sample {
private static final Logger log= LoggerFactory.getLogger(Sample.class);
        private String message;
        @Inject
        private String text;


        @Inject
        private String psd;

          @Inject
        private String link;

        @Inject
        private String link1;

        @Inject
        private String backgroundColor;

        @Inject
        private String blrImage;


        @Inject
        private String backgroundoption;

        @Inject
        private String alignment;
        @Inject @Named("sling:resourceType") @Default(values="nod resourcetype")
        private String resourceType;
        @PostConstruct
        protected void init() throws RepositoryException
        {
                    message="welcome";
                    message+="resourceType :::::::::::::::::"+resourceType;
         }
        public String getMessage() {
                    return message;
        }

        public String getText() {
                    return text;
        }


        public String getLink() {
                    return link;
        }


        public String getPsd() {
                    return psd;
        }

        public String getLink1() {
                    return link1;
        }

        public String getAlignment() {
                    return alignment;
        }
        public String getBackgroundColor() {
                    return backgroundColor;
        }

        public String getBlrImage() {
                    return blrImage;
        }

        public String getBackgroundoption() {
                    return backgroundoption;
        	}
        }
```

Fetch property value by using backend sling model

--------------------------HTML---------------------------------

```
Hello
<sly data-sly-use.info="com.aem.community.core.models.FetchDialogvalue">
<div data-sly-list.childInfo="${info.listpojo}">
		${childInfo.title}
</div>
${info.message}
</sly>
```

-----------------sling model-------------------

```
package com.aem.community.core.models;
import java.util.ArrayList;
import java.util.List;
import javax.annotation.PostConstruct;
import javax.jcr.Node;
import javax.jcr.NodeIterator;
import javax.jcr.RepositoryException;
import org.apache.sling.api.SlingHttpServletRequest;
import org.apache.sling.api.resource.Resource;
import org.apache.sling.api.resource.ResourceResolver;
import org.apache.sling.models.annotations.DefaultInjectionStrategy;
import org.apache.sling.models.annotations.Model;
import org.apache.sling.models.annotations.injectorspecific.SlingObject;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory
import com.aem.community.core.pojo.ReadPojo;
@Model(adaptables={Resource.class,SlingHttpServletRequest.class},defaultInjectionStrategy=Def
aultInjectionStrategy.OPTIONAL)
public class FetchDialogvalue {
private static final Logger log= LoggerFactory.getLogger(FetchDialogvalue.class);
```

```
        private String message;
        @SlingObject
        SlingHttpServletRequest req;
        List<ReadPojo> listpojo=new ArrayList<ReadPojo>();
        @PostConstruct
        protected void init() throws RepositoryException
        {
                message="welcome";
                 ResourceResolver rr=req.getResourceResolver();
                  Resource
r=rr.getResource("/content/AEMMaven12/en/jcr:content/root/responsivegrid/ss/accordian");
                  Node node=r.adaptTo(Node.class);
                  NodeIterator nitr=node.getNodes();
                  while(nitr.hasNext())
                  {
                        ReadPojo p=new ReadPojo();
                        Node nn=nitr.nextNode();
                        String name=nn.getName();
                        String path=nn.getPath();
                        log.info("name::::::::"+name);
                        log.info("path:::::::::"+path);
                        p.setTitle(nn.getProperty("contentTitle").getValue().getString());
                        listpojo.add(p);
                        }
                  }
                        public List<ReadPojo> getListpojo() {
                                return listpojo;
                        }
                       public String getMessage() {
                return message;
        }
}
```

-----------------normal java poojo class------------------

```
package com.aem.community.core.pojo;

public class ReadPojo {
        private String title;
        public String getTitle() {
                return title;
        }
        Public void setTitle(String title) {
                this.title = title;
        }
}
```

Tags

Tag is a keyword or metadata and allows to search content quickly.Tags can be applied to both pages and assets. Tags allows us to organize and categorize site. Method of classifying content with in website. Tags can be attached to page and asset. Main Motto of the tags to improve the search capability of site.

Tags can be an effective means of **organizing your AEM pages or assets**.

Steps:

- First create namespace
- create tag
- After creating tags are attached to the page and asset.

Reference Link: https://helpx.adobe.com/in/experience-manager/6-4/sites/authoring/using/tags.html

DAM

What Is Renditions

Rendition is a process of uploading images to **DAM**(**D**igital **A**sset **M**anagement), It allows you to create renditions of an image, which include different sizes, and versions of the same **asset.** These renditions are helpful in creating thumbnails or smaller image views of large and high-resolution images which can be used in the content of the website. It makes the website load images faster.

Reference Link:

https://www.linkedin.com/pulse/image-rendition-aem-prabhukarruppusamy-ramasamy

Sightly

What is sightly

sightly is new template language developed by adobe. it is the recommended language to use for developing components using aem instead of jsp.

why we need sightly

it is simplifies the development of project. It allows html developer without java knowledge to participate in aem

Security: slightly is automatically filter and escape all text being out put to the presentation layer to prevent from cross side script attack.
In future slightly purposely limited. All the complex login must be place in help class. Which our html file can be easily invoke.

Example
<p data-sly-use. info="scrpit.js">
${info.text}
<p>

Sightly Mechanism

It consists of:

1. Expression
2. sly elements
3. attributes
4. comments

1. **Expression -** ${} like ${!currentPage.jcr:title}

It renders title of html page and fetch the value from jcr. we can use logical operator in expression section.

2. **Sly elements -** slightly provide own elements which is called slightly element. It removes from output while executing the test statement. use it for all element that are not the part of markup.

Example- <div data-sly-include="header.html"></div>

out put will be like <div> header.html</div>

<sly data-sly-include="header.html"></sly>

Output : header.html

3. **Attributes:**

- **Text:** Replace the content of html element with specified text

  Ex: `<div data-sly-text="#{currentPage.title}">title page</div>`

- **List :** Repeats the children element.

```
Ex:   <div data-sly-list.childEle="${currentPage.listChilder}">
                ${childEle.title}
        </div>
```

- **Repeat:** Repeat the same elements

  Ex: `<sly data-sly-repeat.child="currentpage.listChilder">${child.title}</sly>`

- **Test:** To hide or show an HTML element.

  Ex: `<div data-sly-test="${wcmmode.edit}">welcome edit mode</div>`

- **Include:** which is include other file with in a particular file.

  Ex: `<sly data-sly-include="my.file"></sly>`

- **Resource:** Include the components

Ex: `<sly dat-sly-resource="${'par' @ resourceType="foundation/component/parsys"}`

- **use:** invoke the help file when you write business logic and complex login in my helper class. that invoke in your html file of component.

```
Ex:   <sly data-sly-use.info="logic.js">
          ${info.title}

    </sly>
```

- **Call :** use to call/invoke the ClientLibray in your component. Client library is place where we have all the client side resource

  exmple css and js file image etc.

  Ex: <sly data-sly-call=${client.all @ categories="vv"}

Include Client Library using sightly
Touch Ui

```
<sly data-sly-use.clientlib="/libs/granite/sightly/templates/clientlib.html"

data-sly-call="${clientlib.all @ categories='clientlib1,clientlib2'}"/>
```

Classic Ui

`<cq: includeClintLib categories=”clientlib1”>`

How to include file and component within component using sightly tags

Files

Classic Ui

```
<cq: include script=”my.jsp”/>
```

Touch Ui

```
<sly data-sly-include=”my.html”/>
```

Components

Classic Ui

```
<cq: include path=”wish” resourceType=”app/content/components/Demo”>
```

Touch Ui

```
<slydata-sly-resource="${ new Path @
resourceType='my/components/content/pods/testcomponent' }"></sly>
```

How to call other service or component to osgi ?
Example

```
Interface A
{
Void m1();
}

Class B Implements A
{
Void m1()
{
System.out.println(“hello”);
}
```

Calling to servlet

```
@Reference
A a;
a.m1();
```

OR

Calling in Jsp

```
<%
com.lara.A a= sling.getService(com.lara.A. Class);
%>
```

Experience fragment

EF has been introduced with aem 6.3 version.it allows the content authors to reuse content across channels include site, page and third-party system.

Create variation for display the data different mobile device or website

Reference Link: https://helpx.adobe.com/in/experience-manager/6-4/sites/authoring/using/experience-fragments.html

WORKFLOW

Definition- work flows consist of series of step that is executed in specific order or A series of task to produce a desire outcome.
Each step performs a distinct activity such as activating a page or sending an email message. Work flows can interact with assets in the repository, user accounts, and Experience Manager services.

Custom ParticipantStepChooser : The **ParticipantStepChooser** allows assigning a user or a group to execute a specific branch of the workflow model and also allows to use a user’s session to perform a certain task like activating pages or assets. This might be useful when you need to write some business logic while allocating a particular Workflow Step to a predefined set of users/groups based on some condition- kind of Dynamic Participant Chooser.

Custom Process Step: The Workflow Process interface is intended to implement an item of job where you are writing some Business logic based on your requirement on the workflow payload or step is executed automatically by system

Lifecycle: It is created when a new workflow is started and ends when the end node is processed.

The following actions are possible on a workflow instance:

- Terminate
- Suspend
- Resume
- Restart

Completed and terminated instances are archived.

Inbox : Each user has its own workflow inbox in which the assigned WorkItems are accessible.

Launcher : Allows you to define a workflow to be launched if a specific node has been updated.

Transition : Defines the link between two consecutive steps.

WorkItem : A workflow instance can have one or many WorkItems at the same time (depending on the workflow model).

Payload : is path where you have to trigger workflow,you have to give path of that.

Process step-Syntax

```
@Component(service=WorkflowProcess.class, property= {"process.label=Training Workflow"})

public class DemoWorkflow implements WorkflowProcess {

@Override
            public void execute(WorkItem item, WorkflowSession session, MetaDataMap map)
throws WorkflowException {

              --------------   Business logic  ----------------

}

}
```

Participant step- Syntax

When creating a Dynamic Participant Step, you can use **application logic** to determine to whom the workflow item is assigned. For example, your participant chooser can select the user that has the fewest work items.

This shows you how to create a custom Dynamic Participant Step and use it in an Experience Manager workflow. In this workflow example, content is reviewed and approved using a custom Dynamic Participant Step.

```
@Component(service=ParticipantStepChooser.class, property = {"chooser.label=Sample
Implementation of dynamic participant chooser"})

public class ParticipantStep implements ParticipantStepChooser

{


public String getParticipant(WorkItem workItem, WorkflowSession wfSession, MetaDataMap
metaDataMap) throws WorkflowException   {

                    --------------   Business logic  ----------------

  }
}
```

Reference Link

https://helpx.adobe.com/experience-manager/using/Workflows.html

https://helpx.adobe.com/in/experience-manager/6-3/sites/developing/using/workflows-models.html

DAM workflow of process step

```
package com.aem.community.core.workflow;

import javax.jcr.Node;
import javax.jcr.PathNotFoundException;
import javax.jcr.RepositoryException;
import javax.jcr.Session;
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;

@Component(service =WorkflowProcess.class, property={"process.label=DamWorkflow"})
public class DamWorkflow implements WorkflowProcess{
private static final Logger LOGGER = LoggerFactory.getLogger(DamWorkflow.class);
        public void execute(WorkItem item, WorkflowSession sesion, MetaDataMap map)
                        throws WorkflowException {
                Session localSession=sesion.adaptTo(Session.class);
        String urlPayload=item.getWorkflowData().getPayload().toString();
LOGGER.info("INSIDE EXECUTE OF DEMO PROCESS STEP"+urlPayload);
        try {
                Node node=localSession.getNode(urlPayload);
                Node jcrNode=node.getNode("jcr:content");
                  LOGGER.info("jcrnode"+jcrNode);
                Node metadata=jcrNode.getNode("metadata");
                  LOGGER.info("metada"+metadata);
                if(metadata.hasProperty("dam:MIMEtype"))
                {
                    LOGGER.info("Inside dam:MIMEtype");
if("image/jpeg".equals(metadata.getProperty("dam:MIMEtype").getValue().toString()))
{
              LOGGER.info("Inside JPEG");
         metadata.setProperty("mervin","bb");
}
    }
       } catch (PathNotFoundException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
         } catch (RepositoryException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
    }
  }
}
```

Editable Templates

ET was introduced in aem 6.2
With the template editor, creating and maintaining templates is no longer a developer-only task. Developers are still required to setup the environment, create client libraries and create the components to be used, but once these basics are in place the author has the flexibility to create and configure templates without a development project or iteration.

The template console allows your (template) authors to:

- Create a new template (either new or by copying an existing template).
- Manage the lifecycle of the template.

Static Template	Editable Templat
1. st is define and configured by the developer	1. Et create and edit by templated autho
2. same structer as the page	2. definge structer, initial content and content
policies.	

3. No dynmic connection between content and page   3. Maintain a dynmic connection between content and page.

https://blog.3sharecorp.com/creating-editable-templates-adobe-experience-manager

MSM

Multi Site Manager (MSM) allows aem developers to create copy of existing site and automatically update the copy when changes are done to the source site.

Multi Site Manager (MSM) enables you to use the same site content in multiple locations. MSM uses its Live Copy functionality to achieve this:

- With MSM you can:
  - Create content once (source), then
  - Copy this content to, and re-use this content in, other areas (live copies) of the same or other sites.
- MSM then maintains the (live) relationships between your source content and its live copies so that:
  - When you make changes to the source content, the source and live copies are synchronized (to apply these changes to the live copies too).
  - You can make adjustments to the content of the live copies by disconnecting the live relationship for individual sub pages and/or components. By doing this, changes to the source will no longer be applied to the live copy.

Link for reference: https://helpx.adobe.com/experience-manager/6-3/sites/administering/using/msm-livecopy.html#CreatingaBlueprintConfiguration

Blueprint

Blueprints target the rollout of complete multilingual website projects and are a tool to control multiple rollout configs and live copies. Blueprints allow you to control multiple live copies and centrally consistent rollout configs for the blueprint's live copies. A blueprint rollout will push modifications to all it's live copies. The default Blueprint Template assumes that the source web site has the following characteristics:

- The web site has a root page.
- The immediate child pages of the root are language branches of the web site. The Name of each page is a language code. When creating a Live Copy, the languages are presented as optional content to include in the copy.

- Each language page contains one or more child pages. When creating a Live Copy, child pages are presented as a chapter that you can include in the copy.

To create a blueprint:

- 1. Open the Tools console.
- 2. Select Tools, then MSM Control Center. Click New... in the top-middle toolbar.
- 3. In the Create Page dialog, define the blueprint:
    - **Title**, the page title,
    - **Name**, the page (node) name
    - Select the Blueprint Template

  click **create**
- 4. Open the newly created page and click the **Edit** button beside **Settings**.
- 5. In the Blueprint Settings dialog, define the blueprint:
    - **Name**, the blueprint name that was defined earlier can be changed.
    - **Description**, e.g. This is my blueprint
    - **Source Path:** set the path of the blueprint, e.g. /content/geometrixx
    - **Thumbnail Image** (optional): this thumbnail will appear in the live copy creation.

Live Copy

Live Copy is the creation of a new site based on the content and structure of an existing site. Changes or additions to the main site can then cascade down throughout Live Copies. When used to its full potential, this is a very powerful tool that can simplify management across a large number of sites with little effort.

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

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

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

- 1. Open the Websites console.
- 2. Select the folder or page below which you want to locate the Live Copy pages.
- 3. Click New > New Live Copy.
- 4. In the Source selection tab, define the Live Copy:
    - **Title**: The title of the root Live Copy page
    - **Name**: The name of the page node
    - **Live Copy From**: Browse to select the page to use as the source of the Live Copy

    - **Exclude sub pages**: Select this option to exclude the child pages from the Live Copy.
- 5. On the Sync config tab, specify one or more Rollout Configs to use for the Live Copy
- 6. Click **Create**.

Rollout Configurations

Rollout is a process that propagates the changes made from the source (Blueprint) to the target (live copy). When you roll out a site, Adobe Experience Manager copies the Blueprint (source) to the live copy (target).

If the components are *locked*, then whenever the source content changes, Adobe Experience Manager automatically updates the target content.

MSM enables you to specify sets of rollout configurations that are used generally, and when required you can override them for specific live copies. MSM provides several locations for specifying the rollout configurations to use. The location determines whether the configuration applies to a specific live copy.

Rollout configurations can be found by navigating to etc/msm/rolloutconfigs.

Language Copy

A language copy is a copy of an existing site that is to be translated to another language. It provides a way to create a copy of a site (or part of a site) that is specially designed for translation. It is one time copy of the content.

The biggest advantage of using MSM over internationalization is ease to add new languages to websites.

Runmodes

we have different run modes in cq. for different envt for example default run modes author and publish.Then for project requirements we have other envt like dev, Qa, prod support. we achieve this by different configuration respectively.

Why Run Modes?

- Uniquely identify an environment and instances
- Unique configurations based on environment
- OSGI Component Creation for a specific environment
- Bundle Creation for a specific environment

There are two types of run modes:

- Primary Run Mode
- Secondary Run Mode

Primary Run Modes are:

- **Author**: This instance is used for the complete development and authoring purpose.
- **Publish**: This is the actual environment which can be accessed by end users.

Secondary Run Modes are:

- Dev Server
- QA Server
- UAT Server
- Prod Server

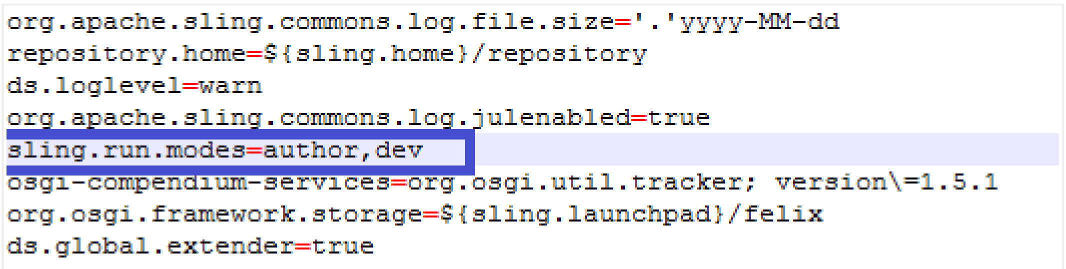
Where we define Run mode?

1. By Changing in sling:properties file

Steps to follow:

Go to **crx-quickstart/conf** directory of aem instance and add the below line:

sling.run.modes= author,dev



```
org.apache.sling.commons.log.file.size='.'yyyy-MM-dd
repository.home=${sling.home}/repository
ds.loglevel=warn
org.apache.sling.commons.log.julenabled=true
sling.run.modes=author,dev
osgi-compendium-services=org.osgi.util.tracker; version\=1.5.1
org.osgi.framework.storage=${sling.launchpad}/felix
ds.global.extender=true
```

Fig- Set the run mode in sling.properties file

2. Using the -r option: When you start the AEM instance by command prompt then set **“-r”** option in the command.

Example: **java -jar jar-name -r dev,sameplecontent**

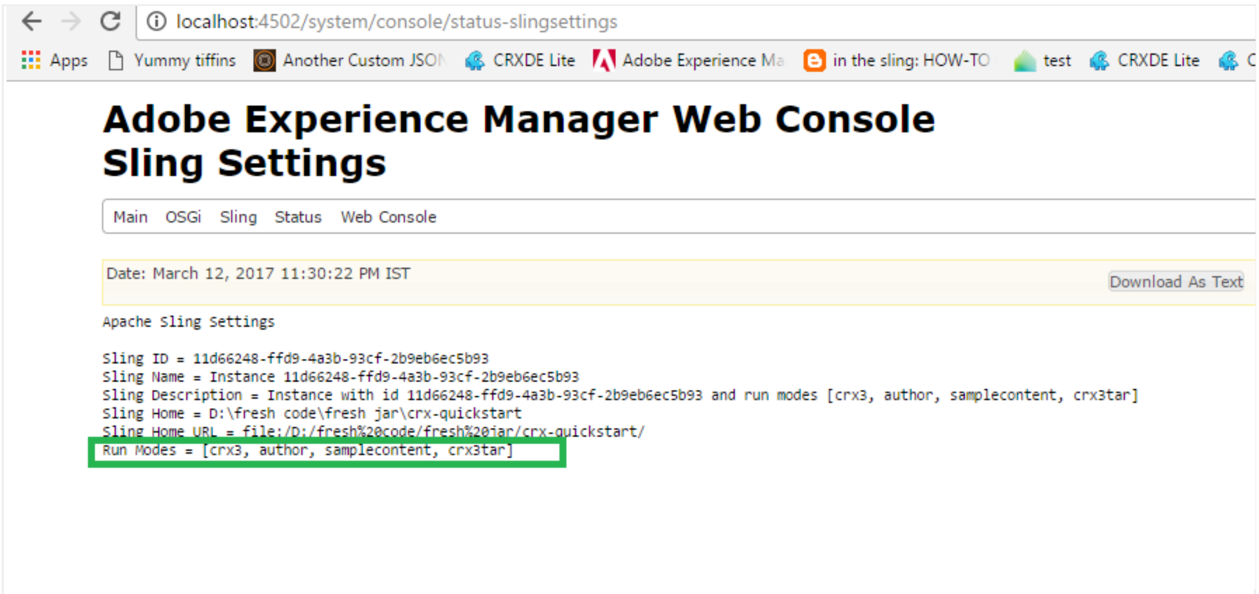
3. By Renaming Jar: cq5-<*run-mode*>-p<*port-number*>

Example: publish instance : cq5-publish -p4503.jar

Author instance : cq5-author-p4502.jar

How to check the run mode of a running AEM instances?

- Go to Felix Console.
- Go to **Status** tab in Navigation and click on **sling settings** option**.**
- Here you can see the Run Modes



```
Date: March 12, 2017 11:30:22 PM IST

Apache Sling Settings

Sling ID = 11d66248-ffd9-4a3b-93cf-2b9eb6ec5b93
Sling Name = Instance 11d66248-ffd9-4a3b-93cf-2b9eb6ec5b93
Sling Description = Instance with id 11d66248-ffd9-4a3b-93cf-2b9eb6ec5b93 and run modes [crx3, author, samplecontent, crx3tar]
Sling Home = D:\fresh code\fresh jar\crx-quickstart
Sling Home URL = file:/D:/fresh%20code/fresh%20jar/crx-quickstart/
Run Modes = [crx3, author, samplecontent, crx3tar]
```

Fig - Check the run mode of running AEM instance

- You can directly go to http://localhost:4502/system/console/status/slingsettings

Problem Statement:

In Felix Configurations, there is a configuration named as **DAY CQ MAIL SERVICE**. Now lets say we have a requirement of using smtp.port=25 in author, Dev,US server but smtp.port=465 in Author,QA,UK Server. (Here US and UK are the location of servers)

Solution:

Create Run Modes:

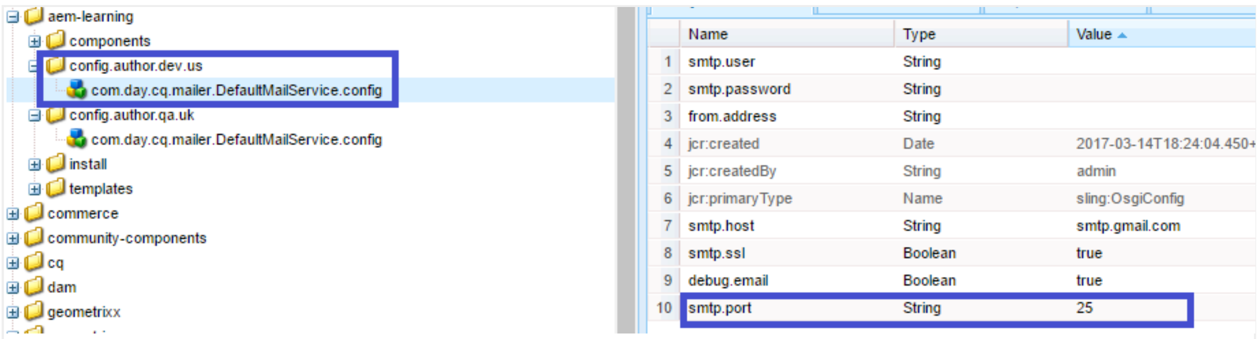
- Go to Crxde.
- Go to /apps/my-project

Create Configurations for Author,Dev Server which is in US:

- Create a folder (**sling:folder**) with the name config.author.dev.us
- Create a **sling:osgiConfig** type of node named com.day.cq.mailer.DefaultMailService.config.

Note:Node name should be the same name as the Persistent Identity (PID) of the configuration in the OSGi Console.

- Add the following properties in this config for Dev Server:

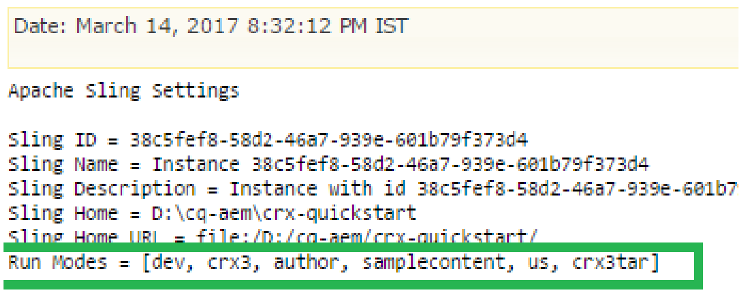


```
aem-learning
  components
  config.author.dev.us
    com.day.cq.mailer.DefaultMailService.config
  config.author.qa.uk
    com.day.cq.mailer.DefaultMailService.config
  install
  templates
commerce
community-components
cq
dam
geometrixx
```

	Name	Type	Value
1	smtp.user	String	
2	smtp.password	String	
3	from.address	String	
4	jcr:created	Date	2017-03-14T18:24:04.450+
5	jcr:createdBy	String	admin
6	jcr:primaryType	Name	sling:OsgiConfig
7	smtp.host	String	smtp.gmail.com
8	smtp.ssl	Boolean	true
9	debug.email	Boolean	true
10	smtp.port	String	25

Fig - Adding the mail service configuration for DEV,US run mode

- Run an AEM instance on Author,Dev,US Environment by using any of the option explained above.
- Check the run mode of running AEM instance



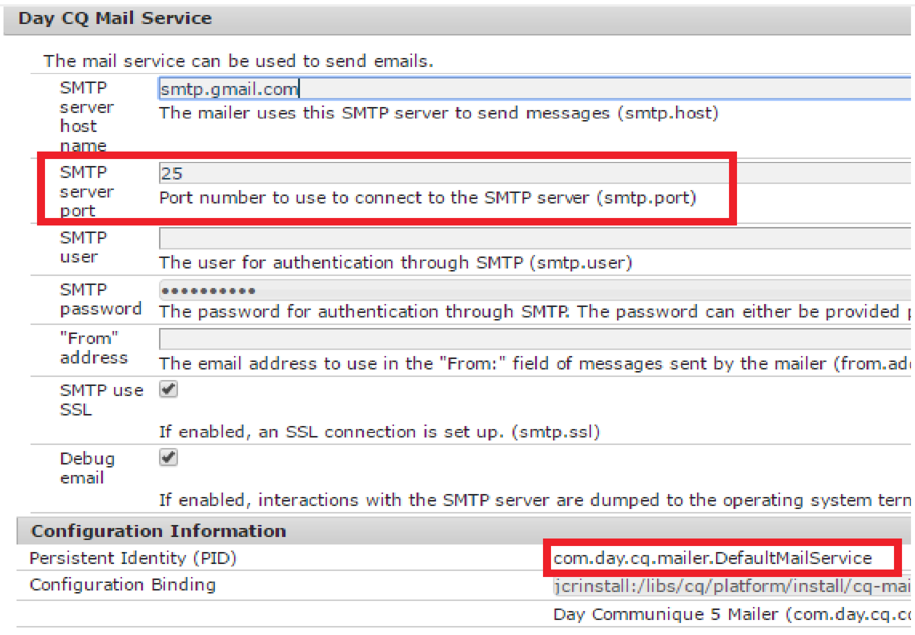
```
Date: March 14, 2017 8:32:12 PM IST

Apache Sling Settings

Sling ID = 38c5fef8-58d2-46a7-939e-601b79f373d4
Sling Name = Instance 38c5fef8-58d2-46a7-939e-601b79f373d4
Sling Description = Instance with id 38c5fef8-58d2-46a7-939e-601b7
Sling Home = D:\cq-aem\crx-quickstart
Sling Home URL = file:/D:/cq-aem/crx-quickstart/
Run Modes = [dev, crx3, author, samplecontent, us, crx3tar]
```

Fig- AEM instance is in AUTHOR,DEV,US run mode

- Now Check the mail Configurations



Day CQ Mail Service

The mail service can be used to send emails.

SMTP server host name	smtp.gmail.com The mailer uses this SMTP server to send messages (smtp.host)
SMTP server port	25 Port number to use to connect to the SMTP server (smtp.port)
SMTP user	 The user for authentication through SMTP (smtp.user)
SMTP password	•••••••••• The password for authentication through SMTP. The password can either be provided
"From" address	 The email address to use in the "From:" field of messages sent by the mailer (from.ad
SMTP use SSL	✓ If enabled, an SSL connection is set up. (smtp.ssl)
Debug email	✓ If enabled, interactions with the SMTP server are dumped to the operating system tern

Configuration Information

Persistent Identity (PID)	com.day.cq.mailer.DefaultMailService
Configuration Binding	jcrinstall:/libs/cq/platform/install/cq-mai Day Communique 5 Mailer (com.day.cq.c

Fig- Check the configuration in felix console for DEV,US run mode aem instance

Dispatcher

Dispatcher is Adobe Experience Manager's caching and/or load balancing tool. Using AEM's Dispatcher also helps to protect your AEM server from attack.

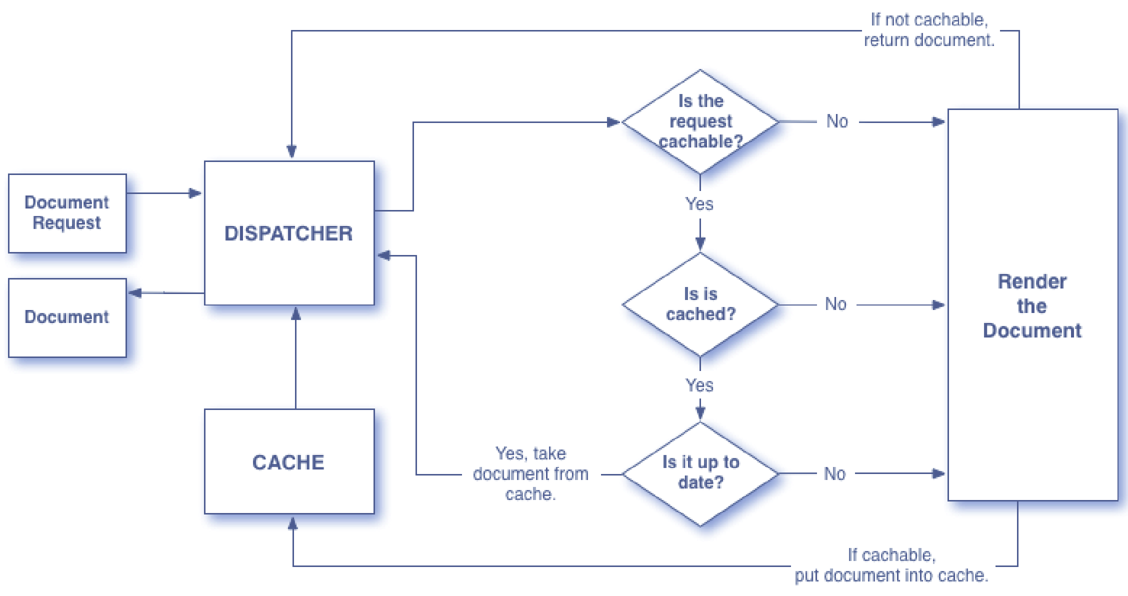
The Dispatcher helps realize an environment i.e both fast and dynamic. It works as a part of a static HTML server, with the aim of :

- Storing as much of the site content as is possible, in the form of static website.
- Accessing the layout as little as possible.

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

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

  How Dispatcher returns Documents



Determining whether a document is subject to caching

The Dispatcher checks the request against the list of cacheable documents. If the document is not in this list, the Dispatcher requests the document from the AEM instance.

The Dispatcher always requests the document directly from the AEM instance in the following cases:

- If the request URI contains a question mark "?". This usually indicates a dynamic page, such as a search result, which does not need to be cached.
- The file extension is missing. The web server needs the extension to determine the document type (the MIME-type).
- The authentication header is set (this can be configured)

Determining if a document is cached

The Dispatcher stores the cached files on the web server as if they were part of a static website. If a user requests a cacheable document the Dispatcher checks whether that document exists in the web server's file system:

- if the document is cached, Dispatcher returns the file.
- if it is not cached, the Dispatcher requests the document from the AEM instance.

Determining if a document is up-to-date

To find out if a document is up to date, the Dispatcher performs two steps:

1. It checks whether the document is subject to auto-invalidation. If not, the document is considered up-to-date.

2. If the document is configured for auto-invalidation, the Dispatcher checks whether it is older or newer than the last change available. If it is older, the Dispatcher requests the current version from the AEM instance and replaces the version in the cache.

The Benefits of Load Balancing

Load Balancing is the practice of distributing the computational load of the website across several instances of AEM.